


AngularFaces and BootsFaces put JSF on Steroids

Riccardo Massera, Stephan Rauh
February 15, 2016 at 10.00:00

Print article
XML (all feeds)

Suggest JSFCentral features.

Sign up for our newsletter!
Email address:
Sign me up!

The Editor's Desk
Podcasts
Inside Facelets
In the Trenches

Google Custom Se
Search

In this series of articles, Riccardo Massera and Stephan Rauh introduce you to the BootsFaces and AngularFaces JSF frameworks. In the first article, they will explain how these frameworks can be useful in your JSF Projects and highlight their main features. The second article will get you started developing a simple project. The last article will cover more advanced features like the new AJAX engine of BootsFaces and the advanced search expressions.

Discussions on Reddit and Hackernews nail it: JSF is dead. It's ugly, it's boring, it's not responsive, it's a waste of server resources, and it doesn't do mobile. Plus, many developers claim they can't wrap their head around it. In 2015, virtually everyone, if given a choice, would prefer HTML5 and JavaScript.

Only that there's that cash cow of ours. We have huge JSF applications. Replacing them with JavaScript is usually not an option. Plus, JSF is mature and stable, something most JavaScript frameworks can’t claim. JavaScript country is in the Gold Rush era, and the dust has yet to settle.

AngularFaces and BootsFaces to the rescue.

Introducing BootsFaces

BootsFaces is a project started by Riccardo Massera at the end of 2013. The mission of the project is to bring the responsive design of Bootstrap to the JSF world.

Riccardo noticed there was something odd about the JSF panorama. There were an abundance of JSF frameworks, each focusing on offering good looking widgets. However, they didn't handle website layout well. Neither did they offer the responsive layout needed by the large variety of target platforms. In particular, until recently, mobile platforms were supported poorly by many JSF libraries. Granted, there’s PrimeFaces, a framework going to great lengths to support mobile platforms, but you will need a second version of the library if you want mobile support, with a different markup and navigation system.

Another thing we didn't (and still don't) like about existing frameworks is the verbose and complex HTML markup they generate. Even if you don't care about performance and network traffic, the complex HTML code does not help web designers easily control the look and feel of a page, except when using a standard theme.

So Riccardo started the BootsFaces project, a new JSF framework based on Bootstrap 3 and the jQuery UI to simplify front-end enterprise application development. In fact, the unique selling point of BootsFaces, that sets it apart from other libraries, is that it uses a standard—Bootstrap—to deliver responsive applications.

For the sake of completeness, we should mention that in the meantime PrimeFaces added its own proprietary responsive design (starting with PrimeFaces 5.1, which came out in October 2014).

The first goal of the BootsFaces team was to help JSF developers create a well-designed and responsive front-end quickly, without forcing them to dive knee-deep into the gory details of CSS and HTML5. Plus, BootsFaces aims to save you from the tedious and error-prone task of writing a lot of markup and JavaScript code.

Using Bootstrap has proven to be the winning choice. It is a very well designed and flexible front-end framework that continues to gain momentum. It is based on all best practices of CSS3 and HTML5 design, and well tested and crafted with great care.

Plus, BootsFaces is small.

Actually, the memory footprint of BootsFaces used to be even smaller, but as often happens, users started to demand more and more features, which caused the footprint of BootsFaces to grow considerably. Still, with less than 650 KB, BootsFaces 0.8.0 is one of the smaller JSF component libraries. The latest version on Maven Central also contains the Bootswatch themes, which makes the jar file bigger, but that's an optional addition. Smaller versions of the file are available at http://www.bootsfaces.net.

A strong focus on layout

To achieve its goals, BootsFaces introduced new JSF components mapping to the most popular features of Bootstrap: components for defining the grid layout and components making it easy to implement a well-organized and good looking navigation. These early components are simple but useful widgets.

This way BootsFaces manages to give JSF applications a fresh new look, and to make them a lot more fun, which is to say: more productive. It also has the additional benefit of being able to target mobile platforms with JSF.

A simple example

It's time to get our feet wet and start coding:

```
<h:form>
        <b:panel title="Please log in" look="primary">
            <b:panelGrid colSpans="6,6">
                        <b:inputText id="username" value="#{loginBean.username}"
                                    label="username" renderLabel="true"
                                    inline="true"
```

```
                        required="true" requiredMessage="Please type your username" />
                    <h:message for="previous" />
                    <b:inputSecret id="password" value="#{loginBean.password}"
                        label="password" renderLabel="true"
                        inline="true"
                        required="true" requiredMessage="Please type your password" />
                    <h:message for="previous" />
            </b:panelGrid>
            <b:commandButton value="login" onclick="ajax:loginBean.login()"
                    update="@form" look="primary" />
        </b:panel>
</h:form>
```

The code above renders a nice Panel with a simple login form.

As you can see, you don't need a lot of code to get a nice looking login page. You don't have to worry about CSS and Javascript. Needless to say there are also standard features like displaying the labels in red if a validation fails. Recently we've also added search expressions like @previous (inspired by PrimeFaces), and we've added a new approach to AJAX using the standard JavaScript event attributes. In our example, that's the onclick attribute of the command button.

By the way, you can make the code more compact using the rapid prototyping features of AngularFaces. In particular, you can omit labels and error messages because they are generated automatically. We'll come back to AngularFaces in a minute.

Integration with other frameworks

After the initial releases, the strategy of the BootsFaces team has been to attract more and more developers. In particular, BootsFaces prefers playing well with existing frameworks over becoming a mutually exclusive alternative. Nowadays, developers can combine BootsFaces with AngularFaces, PrimeFaces, and OmniFaces. Developers have even reported that they can successfully run their application with RichFaces, but until now, we haven't included RichFaces in the list of officially supported JSF libraries.

For instance, one of the first tasks Stephan Rauh worked on was to make his pet framework, AngularFaces, compatible with BootsFaces. Today, both frameworks cooperate seamlessly, putting all the power of both AngularJS and Bootstrap at your fingertip.

Meanwhile, the focus of the team has also been on strengthening the framework and adding nice features already existing in other frameworks. For instance, as you saw in the example, BootsFaces 0.8.0 brings a powerful search expressions API inspired by PrimeFaces (and Thomas Andraschko in particular) and a new approach to AJAX support.

At the time of this writing, BootsFaces consists of 45 components, give or take a few, covering most of the fundamental building blocks of a modern JSF application. It doesn't cover every possible JSF widget, nor does it want to do so. We are fully aware that there are powerful libraries like PrimeFaces that deliver a host of advanced components.

For example, what good is it if the BootsFaces team were to write a barcode component if there's already a superior barcode component in PrimeFaces? So, the general idea is to use BootsFaces for the layout and the basic JSF widgets, and PrimeFaces for the advanced widgets. Of course you can also

use HTML5, provided you are already on JSF 2.2. There's a large number of useful Bootstrap components out there, so you can probably gradually replace server-side components with client-side widgets.

Adding AngularJS to the equation

AngularFaces is different. It's not just another component framework. It doesn't provide components at all. It's a plugin to many popular JSF component libraries, such as PrimeFaces, BootsFaces, and of course Mojarra and MyFaces, enabling you to leverage the power of AngularJS in your JSF application. In the long run, that's a major revolution: AngularFaces allows you to move JSF components from the server side to the client. Or you can write an Angular-driven single-page-application with a JSF backend. Or you can do both. But, with AngularFaces, you still have the option to take the conservative approach: simply replace the server-side validations with client-side validations. Plus, it adds decent support for targeting international audiences, for rapid prototyping, and much more.

Making two incompatible containers work together

AngularFaces has attracted a lot of attention among the JSF community. In part, that's because AngularFaces manages to do something everybody considered impossible: the application contains two containers, AngularJS and JSF, each with its own life cycle. Plus, both frameworks claim exclusive ownership of the HTML DOM. AngularFaces allows these containers to coexist peacefully. It's even possible to combine JSF AJAX with AngularJS.

AJAX vs. AngularJS

At first glance, combining JSF AJAX with AngularJS seems to be very attractive. Often you don't need to transfer HTML code, but you can update the AngularJS model and just have data updates between server and client. That’s a lot less network traffic.

Unfortunately, this approach adds a lot of complexity to your JSF application. So we don’t recommend it, because the goal of AngularFaces is to make your application simpler. The AngularFaces way is to replace AJAX with AngularJS. The JSF application implements a very simple request-response paradigm, like "in the old days" (which are merely five or ten years ago). The fancy, interactive stuff is done with AngularJS. You end up with a series of single page applications. Or, to use a buzzword, you end up with a couple of *microservices*. The smaller scale of the AngularJS applications reduces the complexity of the JavaScript code. The JSF toolchain reduces the complexity even further, by allowing you to use JSF templates and advanced JSF widgets.

Using AngularJS components in JSF applications

Another fruitful strategy is to replace JSF components with AngularJS components. This way, you don't necessarily have to implement your business logic on the client side, but you can still benefit from powerful AngularJS widgets like ng-table or smarttable.

Reality check

AngularFaces has gained a stable, even growing, community of developers using it actively. It has found a comfortable niche to live in. However, the community is a lot smaller than the BootsFaces community. Most developers shy away from the complexity of AngularFaces. Actually, that's largely a prejudice: there's less additional complexity than many architects and developers believe, in particular if you drop JSF AJAX.

Be that as it may, we invest much more time in BootsFaces than in AngularFaces because more developers are interested in BootsFaces. But then, AngularFaces is basically a JSF plugin with a limited scope. As soon as it is stable, there's little work to do. Most changes are triggered by exterior changes, such as new AngularJS versions. Speaking of which: AngularFaces currently supports AngularJS 1.3. Most likely, the next AngularJS version to be supported will be Angular 2. We intend to skip AngularJS 1.4 and 1.5.

Why should you use AngularFaces?

AngularFaces is a low-risk option to start the migration from a server-side JSF application to a client-side JavaScript application. Alternatively, you can use AngularFaces to make JSF programming simpler. It's a bridge technology, but chances are you'll like living on this bridge. AngularFaces allows you to use the best of two worlds.

Even if you aren't interested in AngularJS, AngularFaces is worth a look: as mentioned before, it offers rapid prototyping features, brings JSR303 validation to the client, simplifies internationalization and allows for a relaxed, HTML-like markup. For example, you have to prefix the <form> tag with h: in JSF. AngularFaces allows you to omit the prefix in many cases, making the JSF markup much more concise.

Conclusion

AngularFaces and BootsFaces are two different approaches to improving JSF. While AngularFaces concentrates on moving logic from the server to the client, BootsFaces concentrates on the mobile platforms and brings a new, fresh look to JSF.

If you are intrigued by BootsFaces, stay tuned. The next article will get you started with BootsFaces with a hands-on showcase of its main features, showing you how to build a simple project. Later we'll write an article covering the new AJAX engine of BootsFaces and the advanced search expressions.

Resources

- Showcase and manual of the BootsFaces project
- Showcase and manual of the AngularFaces project
- The original article describing why Stephan started the AngularFaces project. While the technical approach described in the article has been abandoned, the ideas still hold.

Riccardo Massera is the founder of BootsFaces. He has over 15 years of professional experience in Web development and was fortunate to witness the dawn of the Internet, authoring one of the first personal web pages in Italy, and later working for one of the first Italian Internet Service Providers. He is now a full-stack developer, working with both server side and client side, with a focus on the latter. He likes working with Bootstrap, jQuery and AngularJS, building projects with Gradle, developing Java Enterprise applications, and he loves Virtualization Technologies. When not coding, Riccardo likes traveling around the world and learning new languages.

Stephan Rauh is a senior consultant at OPITZ CONSULTING GmbH (http://www.opitz-consulting.com/en/home.php). Chances are you already know Stephan's blog, http://www.BeyondJava.net. During the last couple of years, he has spent a lot of time with JSF and AngularJS. He is the author of AngularFaces and a regular committer to the BootsFaces project. He will be talking about BootsFaces and

AngularFaces at the Javaland 2016 conference (see https://www.doag.org/konferenz/konferenzplaner/konferenzplaner_details.php?locS=0&id=499959&vid=509860) and—maybe—also at the JAX 2016 conference.

Site version 2.00 Report web site problems