BootsFaces: JavaServer Faces in Zeiten von Angular, React und Co.

Stephan Rauh, Opitz Consulting Deutschland GmbH

Wir schreiben das Jahr 2019: Alle Welt verwendet moderne JavaScript-Frameworks für Benutzeroberflächen. JavaServer-Faces(JFS)-Entwickler werden nur müde belächelt oder gar bedauert. JSF ist aus der Mode gekommen. Es gilt als altbacken, kompliziert und außerdem noch langsam. Genau in dieser Zeit tritt unser Open-Source-Projekt BootsFaces an, die Lücke zu schließen und Java-Entwicklern das Tor zu modernen, responsiven UIs aufzustoßen.

Allen Unkenrufen zum Trotz kennen die Downloadzahlen von BootsFaces nur eine Richtung: Sie stürmen nach oben – und das schon seit Jahren. Das Gleiche gilt für die Downloadzahlen von OmniFaces, die in unregelmäßigen Abständen veröffentlicht werden *[1]*. Bei den Kollegen vom PrimeFaces-Team frage ich nur unregelmäßig nach, sie sind aber erfolgreich genug, um von ihrer Cash-Cow PrimeFaces komfortabel zu leben und gleichzeitig in die JavaScript-Welt zu expandieren: PrimeNG, PrimeReact und seit Neuestem auch PrimeVue.

Nichtsdestotrotz wirken die Downloadzahlen geradezu vernachlässigbar, wenn wir sie mit Angular (sechs Millionen Downloads pro Monat) oder React.js (elf Millionen Downloads pro Monat) vergleichen. BootsFaces kommt auf rund 3.000 Downloads pro Monat. Selbst wenn man unterstellt, dass npm-stat.com anders zählt als Maven Central, ist der Unterschied beeindruckend. Warum ist BootsFaces trotzdem interessant? Was motiviert die Mannschaft hinter BootsFaces, weiterzumachen? Welche Antworten hat BootsFaces auf die Herausforderungen Angular, React und Vue.js?

Feedback der Community

Die zweite Frage ist schnell beantwortet: Die BootsFaces-Community motiviert uns immer wieder aufs Neue. Wir bekommen eine Menge positives Feedback, und das macht Spaß. Noch dazu, wenn es von Leuten kommt, die wir ansonsten nie kennengelernt hätten. Manchmal gewinnen wir dadurch sogar Einblick in fremde Kulturen

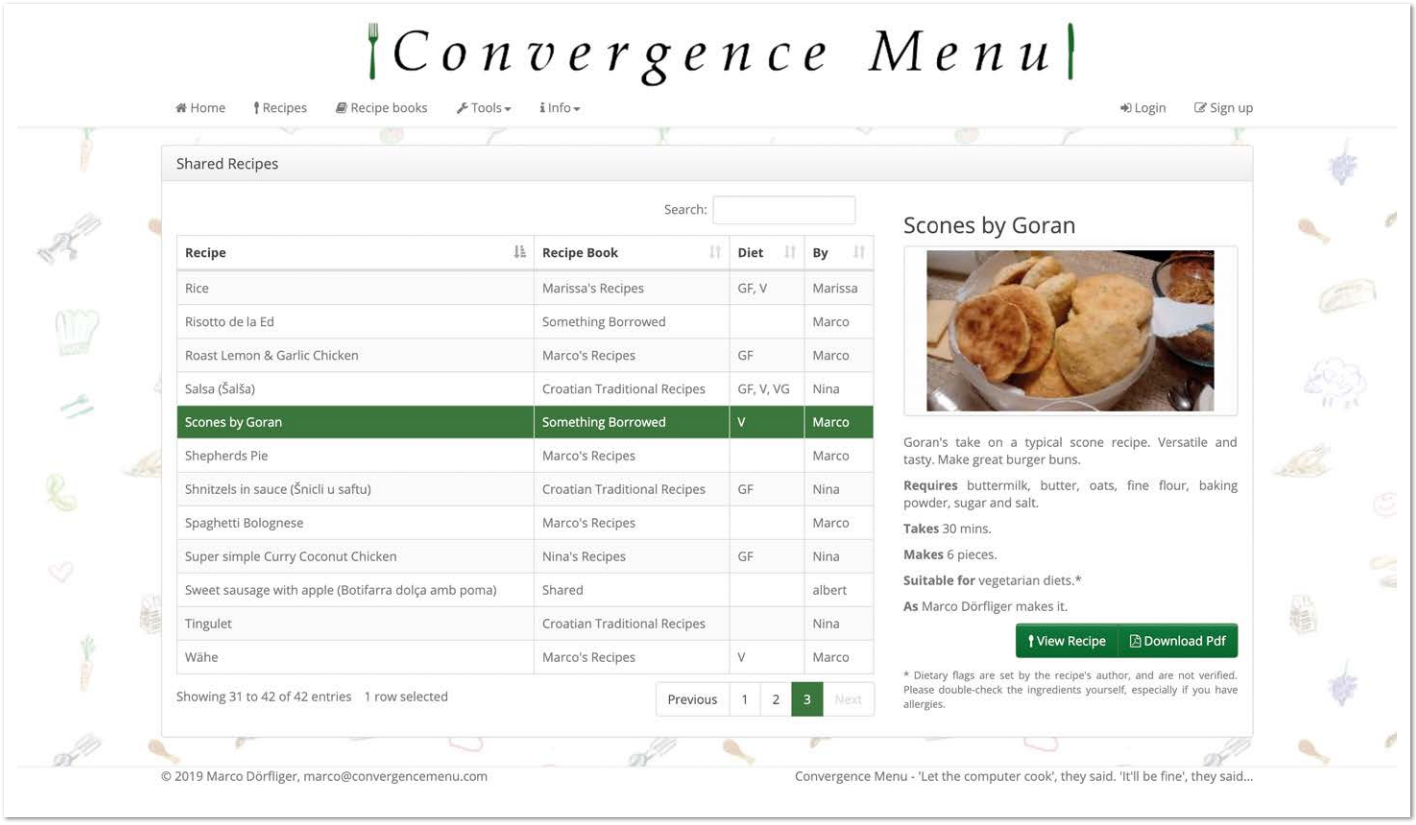

Abbildung 1: Das Convergence Menu von Marco Dörfliger zeigt viele populäre Elemente von BootsFaces, wie z.B. Layout, Tabellen, Menüs

oder sehen Projekte, die mit unserem Framework realisiert wurden. Das ist und bleibt spannend!
Für die Beantwortung der zweiten Frage möchte ich ein wenig ausholen.

Was ist BootsFaces?

BootsFaces ermöglicht es JSF-Entwicklern, das Layout-Framework Bootstrap zu verwenden. Bootstrap wiederum ist ein CSS-Framework, das mithilfe einiger Ideen aus dem Buchdruck beinahe automatisch für gutaussehende Webseiten und Webanwendungen sorgt. Wer einmal versucht hat, eine Visual-Basic-Anwendung Pixel für Pixel sauber auszurichten, weiß die Vorteile von Bootstrap zu schätzen.

Streng genommen ist BootsFaces spätestens seit JSF 2.2 überflüssig. CSS-Klassen und natives HTML konnte man schon immer in JSF verwenden, und seit JSF 2.2 können die nativen Eingabefelder von HTML auch in einer JSF-Anwendung eingesetzt werden („Pass-Through-Elemente"). Ich habe diesen Ansatz in einem großen Projekt benutzt. Der große Vorteil ist die Flexibilität. Wir konnten beliebige JavaScript-Komponenten aus dem Internet in unserer Anwendung verwenden. Das Ergebnis war eine schnelle und gutaussehende Anwendung mit ansprechender Ergonomie.

Der Nachteil war, dass unsere Quelltexte sehr umfangreich wurden. Sie waren auch nicht leicht zu lesen. Bootstrap nutzt sehr viele -Tags, die sich nur durch eine CSS-Klasse weiter hinten in der Zeile unterscheiden. Komplexere Elemente bestehen aus einer ganzen Reihe von HTML-Tags, die erst in der Kombination als modaler Dialog oder Reiterkarten erkennbar werden *(siehe Listing 1)*.

Mission von BootsFaces

Spätestens seit JSF 2.2 lautet unser Motto daher: JSF mit Bootstrap soll einfacher werden. Wenn wir schon dabei sind: Auch JSF selbst soll einfacher werden, egal ob mit oder ohne Bootstrap. BootsFaces bietet einen einfacheren Zugang zu AJAX, einen ganzen Schwung neuer Search Expressions, eine optionale, HTML-artige Syntax und vieles mehr. Natürlich auch eine Reihe von UI-Komponenten. Schauen wir uns unser Beispiel in der BootsFaces-Variante an *(siehe Listing 2)*.

Acht Zeilen anstelle der ursprünglich benötigten 24 Zeilen sind ein deutlicher Vorteil. Noch wichtiger: Diese acht Zeilen zeigen die Rolle des jeweiligen Tags gleich am Anfang der Zeile, statt sie – wie in der HTML-Lösung – in den class-Attributen zu verstecken. Die verbesserte Lesbarkeit kommt Entwicklern in der Wartungsphase ihrer Anwendung zugute. Die meisten Programme werden nur einmal geschrieben, aber später während einer oft hektischen Fehlersuche häufig gelesen. Deswegen legen wir großen Wert auf die Lesbarkeit der Quelltexte.

UI-Komponentenbibliothek

Die Komponentenbibliothek von BootsFaces umfasst 82 Widgets. Wie wir gerade gesehen haben, ersparen viele dieser Widgets Ihnen eine Menge Schreibarbeit, indem sie aus kompaktem XHTML-Code mehrere Zeilen HTML und JavaScript erzeugen. Gleichzeitig ist es eine Besonderheit von BootsFaces, dass es – im Vergleich zu anderen JSF-Komponentenbibliotheken – relativ kompakten HTML-Code generiert. BootsFaces kann sich das leisten, weil es jünger ist als die meisten Mitbewerber. Wir setzen konsequent auf HTML5 und können damit viele Features nutzen, die anderen JSF-Frameworks aus Kompatibilitätsgründen verwehrt bleiben.

```
<div class="tab-panel" role="tabpanel">
  <ul class="nav nav-tabs" role="tablist">
     <li role="presentation" class="active">
        <a role="tab" data-toggle="tab"
           href="#j_id_c_pane">
           First tab
        </a>
     </li>
     <li role="presentation">
        <a role="tab" data-toggle="tab"
           href="#j_id_e_pane">
           Another tab
        </a>
     </li>
  </ul>
  <div class="tab-content" role="tablist">
     <div id="j_id_c_pane" class="tab-pane active">
        First tab
     </div>
     <div id="j_id_e_pane" class="tab-pane">
        Second tab
     </div>
  </div>
</div>
```

Listing 1

```
<b:tabView>
  <b:tab title="First tab">
     First tab
  </b:tab>
  <b:tab title="Another tab">
     Second tab
  </b:tab>
</b:tabView>
```

Listing 2

```
<div class="container">
  <div class="row">
     <div class="col-sm-4">left-hand-side column</div>
     <div class="col-sm-8">right-hand-side column</div>
  </div>
</div>
```

Listing 3

```
<b:container>
  <b:row>
     <b:column colSm="4">
        left-hand-side column
     </b:column>
     <b:column small-screen="two-thirds">
        right-hand-side column
     </b:column>
  </b:row>
</b:container>
```

Listing 4

Layout-Komponenten

Historisch gesehen bilden die Layout-Komponenten den Kern von BootsFaces. Die meisten Bootstrap-Seiten bestehen aus Containern, Zeilen und Spalten. Ein einfaches HTML-Layout sieht mit Bootstrap wie in *Listing 3* beschrieben aus.

BootsFaces übernimmt dieses Schema und verbessert die Lesbar-

keit durch spezielle Tags. Sie sehen auf den ersten Blick, ob es sich um einen Container, eine Spalte oder eine Zeile handelt. Wenn Sie wollen, können Sie die etwas kryptischen Ausdrücke „sm-4“ durch Klartext ersetzen. Das nächste Beispiel zeigt in der oberen Hälfte die kompakte Schreibweise, die einen erfahrenen Webdesigner anspricht, und in der unteren Hälfte die wortreiche Syntax, die neuen Mitgliedern in Ihrem Team das Leben erleichtert *(siehe Listing 4)*.

Variationen der Syntax

Es hat ein paar Jahre gedauert, bis die Entwicklergemeinschaft die neuen Variationen der Syntax angenommen hat. Das Beispiel zeigt drei der neuen Möglichkeiten:

- BootsFaces erlaubt den Bindestrich als Alternative zum Binnenmajuskel. Einfacher formuliert: Viele JSF-Attribute bestehen aus mehreren Wörtern. Wie in Java weithin üblich, werden die Wortgrenzen durch einen Großbuchstaben abgegrenzt. Im Beispiel wäre das also „„smallScreen“ oder (weniger leicht ersichtlich) „colSm“. Wenn Sie viel mit Bootstrap oder generell viel mit CSS arbeiten, kennen Sie auch eine andere Konvention: Wortgrenzen werden in CSS-Klassen durch einen Bindestrich markiert. Das stellt einen BootsFaces-Entwickler vor eine unnötige Hürde. Mal gilt der Bindestrich, mal die Großschreibung. Es gibt eine klare, einfache Regel dafür – aber warum sollten wir unsere Anwender zwingen, sie zu lernen? Unsere Antwort: Wir erlauben einfach beides.
- In eine ähnliche Richtung geht die Vereinfachung des Attributnamens. Im Prinzip hat Bootstrap mit den T-Shirt-Größen schon eine ziemlich raffinierte Metapher gefunden. Wir bieten mit „small-screen“ trotzdem eine – hoffentlich – ausdrucksstärkere Alternative zu „col-sm“ an.
- Für den Wert der Spaltenbreite können Sie wahlweise eine Zahl oder einen englischen Klartext angeben. Unsere Idee dahinter ist, dass es für einen Laien sofort klar ist, was „one-third“ oder „half“ bedeutet. Die traditionelle Angabe „col-sm-4“ bzw. „col-sm-6“ erfordert etwas mehr Einarbeitung. Wie immer im Leben gibt es natürlich auch die andere Sichtweise: Erfahrene Web-Designer fühlen sich wahrscheinlich wohler mit der kompakten Schreibweise, die sie schon seit Jahren kennen. Also erlauben wir auch in diesem Fall beides.

Die Flexibilität von BootsFaces hat leider eine Kehrseite. Bisher haben wir keine der großen IDEs Eclipse, IntelliJ und NetBeans überzeugen können, speziellen Support für BootsFaces anzubieten. Daher werden fast alle Attribute der Komponenten doppelt angezeigt – einmal mit Bindestrich und einmal mit Großbuchstaben. Das hat in der Vergangenheit schon einige Entwickler verwirrt.

```
<container>
  <row>
     <column small-screen="one-third">
        left-hand-side column
     </column>
     <column small-screen="two-thirds">
        right-hand-side column
     </column>
  </row>
</container>
```

Listing 5

HTML-artige Syntax

Die fehlende Unterstützung der IDEs verhindert womöglich auch den Durchbruch der HTML-artigen Syntax. BootsFaces erkennt auch Komponenten ohne das Präfix „b:“ als BootsFaces-Komponente. Unser Beispiel wird damit noch eine Kleinigkeit kompakter und ausdrucksstärker *(siehe Listing 5)*.

Leider markieren die meisten IDEs diesen Quelltext als fehlerhaft und bieten keine Codevervollständigung mehr an. Das wiederum hindert die meisten Entwickler daran, das Feature zu verwenden.

Vielleicht ist es aber auch einfach so, dass die meisten Entwickler meine Allergie gegen XML nicht teilen. An und für sich würde ich auch gerne eine kompaktere, weniger zeremonielle Alternative zu XHTML anbieten. Beispielsweise könnte die Liste der Namespaces am Beginn jeder XHTML-Datei wegfallen, wenn man mit Default-Werten für die gängigen Namespaces arbeitet. Der JSF-Standard würde das durchaus erlauben. Leider ist es sehr schwierig, die JSF-Basisbibliotheken Mojarra und MyFaces entsprechend anzupassen – vor allem, wenn die Erweiterung mit beiden Bibliotheken funktionieren soll.

Zusätzliche Breakpoints

Der große Vorteil eines Frameworks wie Bootstrap ist, dass sich Anwendungen dynamisch an unterschiedliche Bildschirmgrößen anpassen. Eine BootsFaces-Anwendung kann je nach Displaygröße unterschiedlich aussehen. Wir haben dafür schon ein Beispiel gesehen.

Dieses Codefragment *(siehe Listing 6)* stellt die Seite in zwei nebeneinanderliegenden Spalten dar. Aber nur auf großen Bildschirmen. Genauer gesagt, nur auf Bildschirmen ab dem Breakpoint „sm“. Bei Bootstrap 3 (und damit bei BootsFaces 1.x) sind das 768 Pixel, bei Bootstrap 4 576 Pixel. Bei beiden Versionen von Bootstrap liegt der größte Breakpoint bei 1200 Pixeln.

So genial dieses Konzept ist, hat es doch einen großen Nachteil. Bisher haben alle meine Anwendungen andere Breakpoints erfordert, als sie von Bootstrap angeboten werden. Das liegt daran, dass Bootstrap hauptsächlich für mobile Geräte entworfen wurde – und zudem für mobile Geräte, die inzwischen vom Markt verschwunden sind.

Als JSF-Framework richtet sich BootsFaces in erster Linie an Desktop-Anwendungen. Dort sind mittlerweile viel größere Bildschirme üblich. Bei Laptops findet sich häufig die Breite 1440 Pixel. Full-HD (1920 Pixel) gehört längst zum Mainstream und seit einigen Jahren sind 4K-Bildschirme en vogue.

Ich würde mir daher ein paar zusätzliche und vor allem praxisnähere

```
<column small-screen="one-third">
  left-hand-side column
</column>
<column small-screen="two-thirds">
  right-hand-side column
</column>
```

Listing 6

Breakpoints wünschen. Material Design macht es vor: Dort gibt es Breakpoints bei 480, 600, 840, 960, 1280, 1440, 1600 und 1920 Pixeln. Ich würde noch 2560 Pixel und die 4K-Auflösung 3840 Pixel hinzunehmen.

Als einer der Committer von BootsFaces brauche ich es nicht bei Wünschen zu belassen. Ich kann die Dinge auch umsetzen. Als ersten Schritt habe ich den Breakpoint „XXL" mit 1400 Pixeln hinzugefügt. Seitdem warte ich vergeblich auf das Feedback der Community. Offensichtlich ist das Problem nicht so drängend, wie ich dachte. Oder wir haben es in der Dokumentation zu gut versteckt. Wie dem auch sei, in einer der nächsten Versionen plane ich, die Breakpoints „XXXL" mit 1600 Pixeln, „full-hd" mit 1920 Pixeln und eventuell noch „QHD" mit 2560 Pixeln sowie „4K" mit 3840 Pixeln zu ergänzen.

AJAX

Die ersten Versionen von JSF unterstützen AJAX noch nicht. Das kam erst relativ spät dazu. Der gewählte Ansatz in *Listing 7* ist sehr flexibel, aber auch sehr wortreich.

Das muss doch auch einfacher gehen!

BootsFaces greift ein paar Ideen von PrimeFaces und Angular auf und entwickelt sie konsequent weiter. Der AJAX-Aufruf wird jetzt direkt am zugehörigen JavaScript-Event definiert. Das ist kompakter, verbessert die Lesbarkeit und kommt damit den Kollegen, die Ihr Projekt in einigen Jahren pflegen müssen, entgegen *(siehe Listing 8)*.

In eine ähnliche Kerbe schlägt das Attribut „auto-update". Jedes Bildschirmelement mit diesem Attribut wird automatisch bei jedem AJAX-Request neu gezeichnet. Typische Beispiele sind die Fehlermeldungen in der Maske oder Eingabefelder, die rot markiert werden, wenn die Eingabe fehlerhaft war.

Automatic AJAX?

Es gibt auch ein paar Dinge, an denen wir gescheitert sind. Bei JSF muss der Programmierer explizit angeben, was nach einem AJAX Request neu gezeichnet wird. Das war seinerzeit ein genialer Schachzug, um AJAX nachträglich zu JSF zu ergänzen. Gleichzeitig ist das auch eine große Fehlerquelle. Daher stellt sich die Frage, ob es nicht einfacher geht. Computer sind sehr gut darin, solche Veränderungen zu erkennen. Viel besser als wir Menschen. Also sollten wir dem Computer diese Aufgabe überlassen.

Dieser Ansatz funktioniert. Die JSF-Bibliothek ICEFaces hat dieses Feature seinerzeit implementiert und ich habe es auch selbst schon erfolgreich viele Jahre in produktiven Anwendungen verwendet. Damals habe ich ein UI-Framework entwickelt, dass JSF in vielerlei Hinsicht verblüffend ähnelt. Nur, dass es etliche Jahre früher entwickelt wurde und wir nie die Chance hatten, es in der Open-Source-Gemeinde zu veröffentlichen.

Vor ein paar Jahren habe ich gemeinsam mit Thomas Andraschko dieses „automatic AJAX" mit unserer Bibliothek BabbageFaces in die JSF-Welt übertragen. Das Projekt erregte einiges Aufsehen, in der Praxis funktionierte es aber nie richtig. Es erfordert, dass man extrem sauberen HTML-Code schreibt. JSF ist ein XML-Dialekt und zwingt damit schon zu einem unnatürlich sauberen Programmierstil. Aber noch nicht einmal das genügte den Ansprüchen unseres Vergleichsalgorithmus. Sehr schade. Ich hätte diese Idee sehr gerne in BootsFaces verwendet und es den JSF-Entwicklern damit erspart, manuell die richtige ID für das Update der AJAX-Requests zu definieren.

JSF soll einfacher werden!

Anhand dieser Beispiele wird deutlich, was wir vom BootsFaces-Team mit unserem Ziel „JSF soll einfacher werden!" meinen. Wir haben uns über viele Eigenheiten von JSF in jahrelanger täglicher Arbeit geärgert und beschlossen, aktiv etwas zu tun. Unsere Hoffnung ist, diese Ideen später auch in den JSF-Standard einfließen zu lassen. Bei einem Thema ist uns das auch gelungen. Thomas Andraschko vom PrimeFaces-Team hat sich von unseren „Search Expressions" anspornen lassen, das Feature in den JSF-Standard einzubringen. Seit JSF 2.3 profitiert die ganze JSF-Community davon – und nicht mehr nur die Anwender von PrimeFaces und BootsFaces.

Search Expressions

Ursprünglich waren die Search Expressions ein Feature von PrimeFaces. Ich hätte die Search Expressions von PrimeFaces gerne direkt für BootsFaces verwendet. Das hätte gut gepasst, weil ich seinerzeit BootsFaces als reines Layout-Framework definieren wollte, das quasi als Erweiterung von PrimeFaces genutzt werden kann. Beides ist anders gekommen. Kaum ein Architekt ist bereit, BootsFaces und PrimeFaces im gleichen Projekt einzusetzen. Wir konnten die Search-Expressions-Engine aus lizenzrechtlichen Gründen nicht ohne Weiteres verwenden.

Die Alternative war, selbst ein Search-Expressions-Framework zu schreiben, dafür zu sorgen, dass das API zu PrimeFaces kompatibel ist, und einen deutlich größeren Funktionsumfang anzubieten. Das passte insofern gut in meine Strategie, als ich ohnehin ein paar neue Search Expressions geplant hatte:

- `@next` ist sehr praktisch, um die Fehlermeldung hinter dem Eingabefeld zu referenzieren
- `@previous` zeigt (in vielen Anwendungen) auf das Label vor dem Eingabefeld
- `@**:id` erlaubt eine rekursive Suche nach einer ID – im Prinzip wie im Dateisystem von Unix. JSF hat die Besonderheit, dass jedes Bildschirmelement eine ID hat und dass diese ID vom JSF-Framework verändert wird. Je nachdem, ob das Element in einer Tabelle, einem Reiter oder einem Formular steht, wird ein Präfix vor die ID, die wir Programmierer definiert haben, gesetzt. Die Wildcard-Suche löst das Problem. „@**:myField" findet alle Felder mit der JSF-ID „myField".

```
<h:commandButton value="Hover over me!">
  <f:ajax render="@form :messages"
     event="mouseover"
        listener="#{AJAXBean.hoverAction}"/>
</h:commandButton>
```

Listing 7

```
<b:commandButton  value="Hover over me!"
           update="@form :message"
           onmouseover="ajax:AJAXBean.hoverAction()"/>
```

Listing 8

Reality Check

Aus meiner Sicht sind die neuen Search Expressions ein großer Erfolg. Sie lösen ein Problem, mit dem ich als JSF-Entwickler massiv zu kämpfen hatte. Jedes Mal, wenn ich JSF-Code kopiert oder verschoben hatte, musste ich einige sehr technische IDs ändern. Üblicherweise in einem Moment, in dem ich den Kopf nicht wirklich für solche technischen Imponderabilien frei hatte. `@next`, `@previous` und `@**:id` lösen das Problem und, wie gesagt, ein großer Teil dieses Features hat sogar den Weg in den JSF-Standard gefunden.

Zu meinem großen Erstaunen gab es lange Zeit überhaupt kein Feedback der Community. Im besten Fall liegt das einfach daran, dass die Features klaglos funktionieren. Es könnte aber auch daran liegen, dass JSF-Entwickler einfach in anderen Bahnen denken als ich. Zum Beispiel so: „Der Quelltext muss nicht elegant, einfach und schön sein. Er muss einfach nur funktionieren." Noch eine andere Möglichkeit ist, dass frustrierte Entwickler gleich zu Angular, React oder Vue.js wechseln.

Aus meiner Perspektive ist das natürlich eine sehr interessante Option. So interessant, dass ich selber eine Angular-Schulung anbiete und – wenn ich die Wahl habe – normalerweise Angular oder Vue.js statt JSF verwende.

Nichtsdestotrotz bin ich weiterhin ein sehr aktiver und begeisterter Committer für ein JSF-Framework. Etwas pointiert ausgedrückt: Was bringt mich dazu, mich bei einem Legacy-Framework zu engagieren? Noch wichtiger: Warum sollten Sie sich im Jahr 2019 noch für JSF und BootsFaces interessieren, wo doch jeder Architekt und Entwickler, der etwas auf sich hält, im Frontend längst auf JavaScript und HTML5 setzt?

Was ist mit Angular, React und Vue.js?

Kein Zweifel, der Trend geht eindeutig Richtung JavaScript. Technologien wie JSF haben ihre besten Zeiten hinter sich. Die Dynamik der JavaScript-Community ist überwältigend. Das ist auch kein Wunder: Es gibt viel mehr JavaScript-Entwickler, als es jemals Java-Frontend-Entwickler gegeben hat. Wer sich ernsthaft und intensiv mit Frontend-Entwicklung beschäftigen möchte, dem rate ich schweren Herzens von BootsFaces ab. Heutzutage ist es schwer, als Full-Stack-Entwickler alle Bereiche von der Datenbank bis zum UX-Design abzudecken. Die Technologie hat sich auf allen Ebenen ausdifferenziert, wird komplexer, und es ist nicht abzusehen, dass sich das so bald ändert.

BootsFaces kann auf anderen Gebieten punkten. Es gibt sie noch, die Full-Stack-Entwickler. Entwickler, die gerne im Backend programmieren und trotzdem eine schöne Benutzeroberfläche mögen oder den Abstraktionslevel, den JSF bietet, zu schätzen wissen. Als JSF-Entwickler muss ich mich nicht zwingend intensiv mit HTML5 auseinandersetzen.

BootsFaces hat – im Vergleich zu anderen JSF-Komponentenbibliotheken – den Vorteil, einen besonders einfachen Zugang zu JavaScript und zu CSS zu bieten. Als Entwickler haben Sie die Wahl. Sie können den Komfort der Standardbibliothek verwenden, Sie können diese Standardbibliotheken mit CSS sehr einfach an Ihr Corporate Design anpassen oder Sie nutzen die nahtlose Integration mit JavaScript.

Ein Blick in die Glaskugel

Hinter den Kulissen tut sich bei BootsFaces einiges. Das kurzfristige Ziel ist es, auf Bootstrap 4 upzudaten. Wir haben schnell festgestellt, dass das einen verblüffend tiefgreifenden Umbau unseres Frameworks bedeutet. Das ist ein interessanter Unterschied zur normalen Anwendungsentwicklung: Dort gibt es einen relativ einfachen und unkomplizierten Migrationspfad. Als Framework-Entwickler wollen wir Ihnen aber den vollen Funktionsumfang von Bootstrap 4 zur Verfügung stellen und der ist nun einmal sehr viel größer – und in vielen Details auch anders – als das Feature Set der Vorgängerversion.

Daher haben wir uns für ein komplettes Redesign entschieden. Die Komponenten von BootsFaces werden jetzt nicht mehr in Java, sondern in der Template-Sprache von Angular definiert. Das Ergebnis ist um ein Vielfaches kürzer, sehr viel besser lesbar und – hoffentlich – leichter zu warten. Der Java-Code wird aus dem Angular-Template generiert.

Das eröffnet neue, spannende Möglichkeiten. Das Angular-Template definiert, wie der fertige HTML-Code aussehen soll. Es sagt nicht mehr, wie wir den HTML-Code erzeugen. Er muss nicht unbedingt von Java generiert werden!

Wir glauben, dass wir aus dem gleichen Quelltext auch eine Angular-Bibliothek, eine React-Bibliothek oder eine Vue.js-Bibliothek generieren können. Oder auch alle drei Varianten gleichzeitig.

Fazit

Wenn diese Idee funktioniert – und warum sollte sie nicht funktionieren! –, haben wir eine sehr elegante Antwort auf unsere ursprüngliche Frage gefunden. Welche Existenzberechtigung hat ein JSF-Framework wie BootsFaces im Jahr 2019? Ganz einfach: Erstens erfreut sich das totgesagte JSF auch heutzutage noch einer erstaunlichen Gesundheit und zweitens ist BootsFaces gleichzeitig auch eine Angular-Bibliothek, eine React-Bibliothek und eine Vue.js-Bibliothek – und damit voll und ganz auf der Höhe der Zeit! Zugegeben – das ist Zukunftsmusik. Um dorthin zu kommen, brauchen wir noch die Unterstützung von ein oder zwei Entwicklern.

Referenzen

[1] *http://balusc.omnifaces.org/2018/07/)*

Stephan Rauh
Stephan.Rauh@opitz-consulting.com

Stephan Rauh arbeitet als Senior Consultant bei der OPITZ CONSULTING Deutschland GmbH und befasst sich seit Jahren mit Angular, JSF und generell mit UI-Technologien. Er ist durch seinen Blog http://www.beyondjava.net und sein Open-Source-Framework BootsFaces bekannt geworden.