Hidden Gems in ADF - Features die kaum einer kennt aber vieles einfacher und besser machen

Markus Klenke
TEAM GmbH
Paderborn

Schlüsselworte
ADF, Entwicklung, New Features, Best Practices

Einleitung

Viele ADF Entwickler oder Projektteams stehen häufig vor der Herausforderung Anwendungen zu bauen, die auf der einen Seite mit der Flexibilität und der User-Experience von modernsten Javascript-basierten Webanwendungen mithalten sollen, auf der anderen Seite aber die Rigidität und Komplexität von komplexen Datenbank Anwendungen abbilden müssen.
Viel zu häufig kommen dann leider Aussagen wie „Das kann ADF nicht, dafür benötigt man jQuery“ oder „Geschäftslogik halte ich lieber in der Datenbank, ich will ja nicht alles x-mal entwickeln. Man unterschätzt als Entwickler sehr leicht den Fakt, dass es sich bei ADF um eine Full-Stack-Development Suite handelt, die von Oracle als Basisframework für viele SAAS-Anwendungen gilt. Daher wird das Framework auch durchgängig um Features erweitert (oder es wurde auch schon vor vielen Jahren erweitert), um diesen Anforderungen gerecht zu werden. Manchmal werden diese Implementierungen allerdings nur im Kleingedruckten eines neuen Releases beschrieben oder finden auf Grund der Vielzahl der Themen in der ADF Entwicklung keinen Platz in Grundlagenschulungen. Somit ist es verständlich aber auch schade, dass diese Features sehr selten in Projekten zum Einsatz kommen.
Der Vortrag soll Komponenten und Konzepte in ADF zeigen, die den meisten Entwicklern nicht bekannt sein dürften, welche aber die für die Konkurrenzfähigkeit einer ADF Anwendung gegenüber anderen Web-Frameworks mehr als nur hilfreich sind.

Der K(r)ampf mit der modernen UI

Wahrscheinlich wird jeder Endanwender einer Business-Anwendung im privaten Bereich mit unterschiedlichsten Geräten auf Webseiten zugreifen und hat dadurch ein Gefühl, wie sich diese Art von Anwendungen zu verhalten haben. Sei es die Geschwindigkeit (wenn man mal die Werbung ausblendet) als auch das Layout von Webanwendungen, vieles hat sich gerade im Zuge des Triumphzuges von Mobilgeräten geändert. Für Business-Anwendungs-Entwickler ist allerdings die Crux, dieses Verhalten mit dem effizienten Anzeigen, der unkomplizierter Eingabe und der komplexen Validierung von Daten zu verheiraten. Je länger mit dem Re-Design der Business-Anwendung gewartet wird, desto größer wird die Kluft zwischen modernen Webanwendungen und der Geschäftsanwendung und umso komplexer wird eine Migration des UI’s.

Grundsätzlich gibt es zwei allgemeine Ansätze, welche gefahren werden können. Der drastische Ansatz sieht vor, dass die Middle-Tier, also der Bereich der Geschäftslogik und Datenaufbereitung von der Oberfläche durch Web-Services (siehe Abschnitt WS) zu trennen und die Datendarstellung durch ein vollständig eigenes Framework auf JS Basis zu ersetzen. Doch ADF bietet viele Möglichkeiten diesen heftigen Schritt nicht zu gehen und trotzdem eine schöne und effiziente Weboberfläche bereitzustellen.

Der wichtigste Übergang von der klassischen Weboberfläche hin zu einer responsiven, flexiblen Anwendung ist sicherlich der Wegfall von Scrollbars, denn diese sorgen bei kleineren Monitoren für ein Chaos der Darstellung (vor allem weil das Framework nach bestem Wissen und Gewissen selbst an manchen Stellen weitere einfügt). Mit den neueren 12c Releases sind zwei Komponenten in die ADFFaces Bibliothek eingebunden worden, welche diese Problematik reduzieren können und der Anwendung direkt ein „moderneres Gewand“ geben.

Das af:masonryLayout sorgt dafür, dass Container, welche nicht mehr im horizontalen View-Port Platz finden würden, in einem Flow-Layout in die Vertikale wandern. Die vertikale Vergrößerung wird in mobilen Geräten dann durch das Framework über swipe-Interaktion scrollbar gemacht. Diese Komponente bietet sich vor allem in Dashboards und Übersicht-Seiten an, da dort häufig UI-Blöcke bzw. Container implementiert sind, welche direkt für die Komponente genutzt werden können.

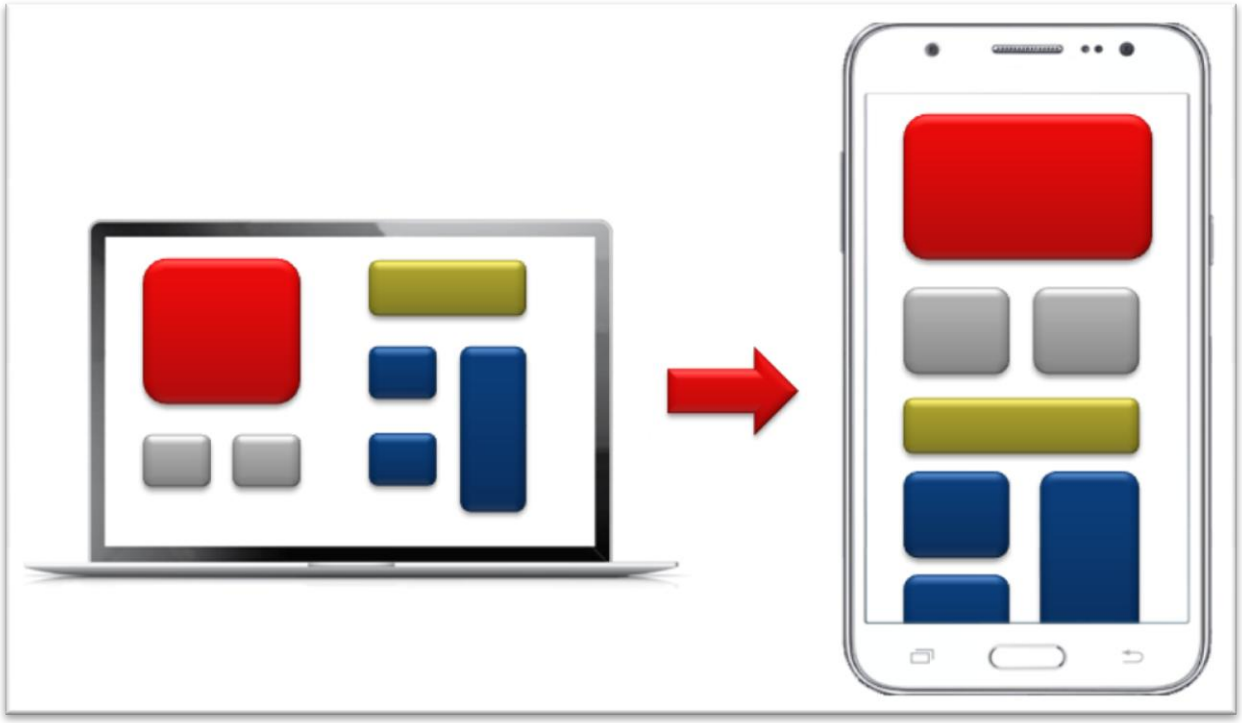

Abb. 1: Schematische Darstellung des Masonry-Layouts

Die Idee der Komponente ist, dass ein javascript angereicherter Container weitere Inhalte nach ihren CSS-Klassen darstellt. Es ist daher notwendig, dass die einzelnen Inhalte eines Masonry-Layouts spezielle (allerdings selbst definierbare) Style-Klassen erhalten. Man kann also mit wenigen Anpassungen und die Unterstützung des Frameworks sehr schnell und einfach interaktive Layouts / Dashboards erstellen.

Das af:masonryLayout ist allerdings hautpsächlich für Container-Strukturen im Konzept von PanelGroupLayouts gedacht. Um die Anwendung wirklich grundsätzlich auf unterschiedliche Monitore (oder andere Client-Attribute) anzupassen, kann das af:matchMediaBehavior tag verwendet werden. Dieses prüft über CSS Media Queries Einstellungen des Clients und setzt daraufhin Attribute in den Client-Komponenten des JSF Frameworks. Am einfachsten kann man das Konzept an einem Navigationsmenübaum beschreiben, wie er in vielen Business-Anwendungen vorkommt. Dieser stellt bei großen Monitoren eine gute Möglichkeit direkt von Anwendungsbereich zu Anwendungsbereich zu springen. Auf kleineren Monitoren und auch auf mobilen Endgeräten nimmt dieses Menü allerdings

zu viel Platz weg, der dann nicht mehr für fachlich wichtige Informationen genutzt werden kann. Es bietet sich in den meisten Fällen also an, dieses Menü über das mediaMatchBehaviour tag in einen Panel Drawer zu überführen. Diese Komponente erlaubt es ein Panel auf Klick einer Reiterkarte über die Hauptseite schweben zu lassen, welches dann das Menü anzeigt. Bei einem weiteren Klick (beispielsweise auf einen Link im Menü) wird dann die Navigation ausgeführt und das PanelDrawer wieder geschlossen. Wird der Browser wieder vergrößert wechselt die Komponente wieder zu dem bekannten Menü.

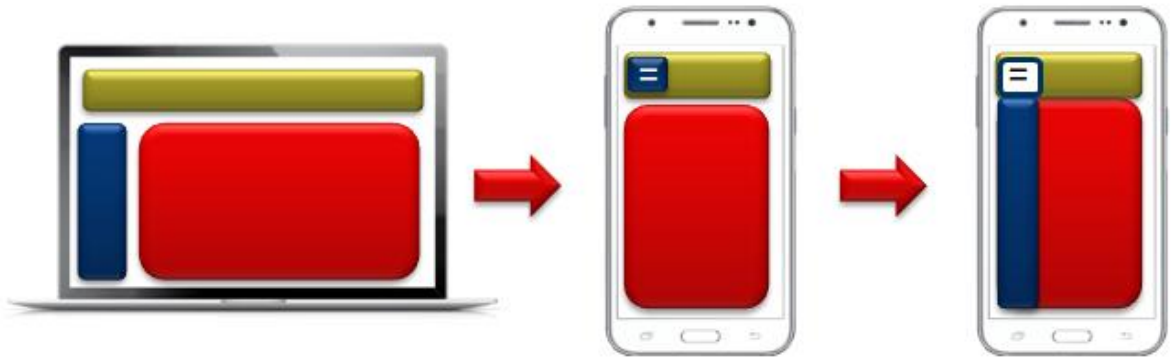

Abb. 2: Schematische Darstellung des MatchMediaBehavior Tags

Unabhängig von den JSF Komponenten die in neueren Versionen von ADF zu einem moderneren Gefühl von Webanwendung führen darf man nicht vergessen, dass ADF und die entsprechenden UI Komponenten ein vollständiges Javascript Interface bieten, so dass diverse Javascript Frameworks sehr einfach in den Lebenszyklus von ADF eingebunden werden können und durch Daten und Parameter von den ADF Faces gesteuert werden können. Außerdem können durch wenige CSS Klassen in einem ADF Skin Webseiten noch näher an den Standard aktueller beliebter Webseiten gebracht werden.

Wiederverwendbarkeit im Datenmodell

Über Good Practices und möglichst gute Hierarchien in den Business Components wurde schon viel geschrieben, Oracle bietet White-Papers an und die ADF EMG bietet Audit-Regeln an, um den Code so sauber wie möglich zu halten. Was jedoch verwunderlich ist, dass viele Themen im Bereich der Wiederverwendbarkeit und der Reduktion von Wiederholungen selten im Fokus liegen. So bieten die ADF Business Components viele Möglichkeiten, Dinge wie Validierung oder Datenkonvertierung nur ein einziges Mal zu definieren und diese Definitionen an vielen Objekten wieder zu verwenden. Um beispielsweise eine Standardisierung für Eingaben zu erhalten können ADFBC Domains erstellt werden. Diese sind Erweiterungen von Standarddatentypen, die implizit eine Eingabevalidierung oder Korrektur eines Datenfeldes durchführen. Ein einfaches Beispiel ist die Eingabe einer IBAN, welche ein vorgegebenes Format hat und technisch sogar validiert werden kann. Man könnte dieses Element natürlich auch auf der Oberfläche mit einem eigenen Validator versehen, müsste dies allerdings auf jedem Element machen. Und wenn die Daten über eine Rest-Schnittstelle übergeben werden, so verliert man diese Validierung gänzlich. Durch die Definition einer IBANDomain kann diese Validierung zentral definiert werden und durch eine einfache Dropdown an den Entity Objekten festgelegt werden. Gleichzeitig kann die Domäne auch für eine Verschönerung der Darstellung sorgen. Bei der IBAN als Beispiel kann entsprechend des Landes überprüft werden, ob die Bankleitzahl und die Kontonummer korrekt sind und gleichzeitig für die Bessere Lesbarkeit alle 4 Stellen ein Leerzeichen erstellt werden. Für das Speichern in der DB wird der String dann wieder aufgeräumt.

```
protected void validate() {
    Boolean valid = true;
```

```
    if(mData != null && mData.startsWith("DE")){
        String unbeautifiedIBAN = mData.replaceAll(" ", "").trim();
        valid = checkBLZ(unbeautifiedIBAN.substring(4,12));
        valid = checkKTO(unbeautifiedIBAN.substring(12));
    }
    if(!valid){
        throw new JboException("IBAN not valid");
    }
}
```

Listing 1: Schematische Darstellung des MatchMediaBehavior Tags

Ähnliches gilt für wiederkehrende fachliche Validierungslogik. Diese werden typischerweise an Entity Objekten erstellt. Wenn diese Entity-Objekte sehr gleichförmig sind, werden die gleichen Validierungen immer und immer wieder erstellt. Um diese Wiederholungen zu vermeiden können Business Logic Groups auf Basis-Entitäten erstellt werden, welche dann über die Vererbung von Entity Objekten diese Wiederholung überflüssig machen und entsprechend den Code aufgeräumt und schmal halten.

In der Demo werden die beschriebenen Technologien in ihrer Benutzung gezeigt und generelle Verbesserungsvorschläge in der Produktivität mit dem JDeveloper und ADF im Allgemeinen gegeben.

Kontaktadresse:
Markus Klenke
TEAM GmbH
Herrmann-Löns Straße, 88
D-33104 Paderborn

Telefon:	+49 (0) 5254-8008 55
Fax:	+49 (0) 5254-8008 19
E-Mail	mke@team-pb.de
Internet:	www.team-pb.de
Blog:	padora.blogspot.com
Twitter:	@MarkusKlenke