ApEx effektiv
Lösungen für Standardanforderungen mit Köpfchen

Stephan Engel
OPITZ CONSULTING Bad Homburg GmbH
Bad Homburg

Schlüsselworte:

ApEx, Application Express, Exception Handling, Fehlerhandling, Navigation, Seitennavigation, JQuery, Generische JavaScript Lösungen, PL/SQL, Best Practice, Tipps & Tricks, Praxisbeispiele

Einleitung

Application Express, kurz ApEx macht seinem Namen alle Ehre. In Windeseile entstehen datenbankzentrierte Webanwendungen. Hat man erst einmal begonnen mit ApEx zu entwickeln, so verfällt man schnell dem Charme des Tools und auf die erste Anwendung folgen schnell weitere. Dabei lässt sich feststellen, dass es einige Anforderungen gibt, die sich in nahezu allen Anwendungen wieder finden. Zu diesen Anforderungen zählen beispielsweise „Jeder Löschvorgang soll mit einer Sicherheitsabfrage bestätigt werden“, „Ich möchte dem Anwender sprechende Fehlermeldungen anzeigen“ oder „Ich möchte nicht, dass die Anwender mit dem Browser „Back“ Button navigieren muss. Alle diese Anforderungen lassen sich natürlich mit ApEx umsetzen, wobei die Umsetzung mehr oder weniger effizient gemacht werden kann. Der Vortrag zeigt Lösungen zur Umsetzung solche Standardanforderungen, wobei der Fokus auf möglichst generischen Ansätzen liegt. Dadurch wird sowohl die Wartbarkeit gesteigert als auch der Entwicklungsaufwand minimiert.

Fachliche Fehler aus Prozessen

Wer schon einmal eine ApEx Anwendung entwickelt hat, in der Geschäftslogik in PL/SQL ausgelagert ist, der kennt das Problem. Innerhalb eines Prozesses tritt ein Exception auf, die Prozedur endet abrupt und der Anwender bekommt die häufig nicht besonders schöne ApEx Fehlerseite zu sehen, möglicherweise mit für Ihn kryptischen Oracle Fehlermeldungen. Diesen Zustand gilt es zu vermeiden. Doch wie kann dies mit ApEx realisiert werden?

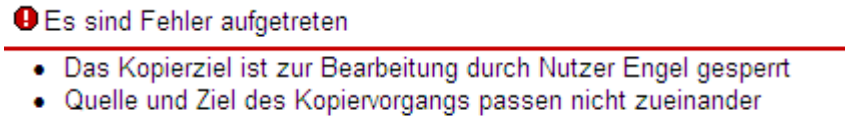

Abb. 1: Benutzerfreundliche Fehlermeldungen

Der typische Ansatz und das von ApEx vorgesehene Mittel für dieses Problem sind Validierungen. Diese bieten dem Benutzer in der Darstellung der Fehlermeldungen hohen Komfort, indem die Fehlermeldungen entweder direkt an den fehlerverursachenden Eingabeelementen oder in einem speziellen Bereich auf der Seite angezeigt werden. Der einzige Nachteil dieser Variante ist, dass sie unter Umständen hohe Implementierungsaufwände nach sich zieht. Hierzu ein Beispiel:

Eine PL/SQL Prozedur ist dafür zuständig, massenhaft Daten zu kopieren. Sie hat deshalb als Eingabeparameter eine Kopierquelle und ein Kopierziel. Bei dem Kopiervorgang kann aber einiges schief

gehen. So könnte beispielsweise das Kopierziel zur Bearbeitung durch einen anderen Benutzer gesperrt oder Quelle und Ziel sind zueinander inkompatibel sein, so dass gar nicht kopiert werden kann. Um diese Fehlersituationen zu erkennen, müssen verschiedene Daten ermittelt werden, die im eigentlichen Kopiervorgang an irgendeiner Stelle verfügbar sind. Dadurch können die Fehler im Prozess ohne großen Mehraufwand erkannt werden. Soll aber mit Validierungen gearbeitet werden, muss die Ermittlung der Daten, die im Prozess bereits verfügbar sind, nochmals ausprogrammiert werden. Dies ist sehr mühselig und erzeugt zudem redundanten Code.

Besser wäre es deshalb eine Möglichkeit zu schaffen, direkt aus dem Prozess heraus Fehlermeldungen in einer sauberen Form an die ApEx Oberfläche weiterzugeben. Wie kann dies realisiert werden?

Naheliegend ist, über PL/SQL in die Datenstruktur zu schreiben, welche von ApEx dazu verwendet wird, die Validierungsmeldungen einzusammeln und dann anzuzeigen. Dies ist jedoch nicht möglich. Allerdings ist die Datenstruktur, welche verwendet wird, um die Erfolgsmeldungen der Prozesse anzuzeigen, beschreibar. Leider handelt es sich hierbei nur um eine einfach Stringvariable, weshalb die Möglichkeiten der Darstellung relativ begrenzt sind. Für den ersten Schritt ist es aber durchaus ausreichend.

Generell ist es eine gute Praxis, Funktionalitäten in Packages zu kapseln. Dies gilt selbstverständlich auch hier. Um die erste Variante zu implementieren, wird ein Logging Package entwickelt, in dem eine Prozedur LOG_TO_APEX Fehlermeldungen in die Prozesserfolgsmeldung schreibt (Listing 1). Die Prozedur muss dabei so geschrieben sein, dass sie die Meldung niemals überschreibt, sondern immer nur Information anhängt. Der große Vorteil dieses Vorgehens ist, dass sich ApEx quasi selbst darum kümmert, wann die Meldungen angezeigt wird und wann nicht. So ist die Meldung beispielswiese nach einem einfachen Seitenrefresh noch vorhanden, wird aber eine andere Aktion wie beispielsweise ein Redirect oder eine Submit ausgelöst so verschwindet die Meldung wieder.

```
PROCEDURE log_to_apex (p_message IN VARCHAR2)
IS
BEGIN
   apex_application.g_print_success_message :=
                  apex_application.g_print_success_message
                  || ' '
                  || p_message;
END log_to_apex;
```

Listing 1: Fehlermeldungen in eine ApEx Variable schreiben

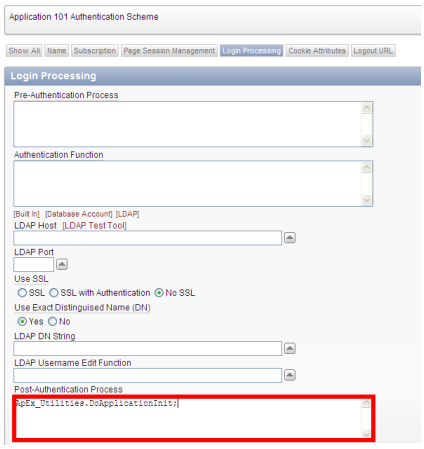

Abb. 2: Ein guter Platz um Collections zu initialisieren

Nachteilig an diesem Verfahren ist, dass es bei einer größeren Anzahl von Logmeldungen schnell unübersichtlich wird. Besser wäre es deshalb, wenn die Daten in tabellarischer Form aufbereitet wären. Zu diesem Zweck eignen sich sehr gut Berichte, die auf SQL Statements basieren. Als Zwischenspeicher für die Logmeldungen kommen in diesem Fall Tabellen, oder ApEx Collections in Frage. ApEx Collections haben den Vorteil, dass sie die Sessionverwaltung gleich mitbringen und so sichergestellt ist, dass die Fehlermeldungen von Benutzer A nicht dem Benutzer B angezeigt werden. Zudem muss man sich keine Gedanken darüber machen, wann veraltete Daten gelöscht werden, da ApEx automatisch dafür sorgt, dass nicht gelöschte Daten zusammen mit der Session abgeräumt werden. Allerdings muss dafür gesorgt werden, dass die Collection auch initialisiert wird.

Hierfür bietet sich ein POST Authentication Prozess an, der beim Authentifizierungschema der Anwendung hinterlegt werden kann (siehe Abb. 2).

Ein allgemeiner Bericht, der Fehlermeldungen in ApEx anzeigt, wird am besten auf der Seite 0 platziert. Auf dieser speziellen Seite können Komponenten platziert werden, die auf allen Seiten einer Anwendung erscheinen sollen. Damit der Fehlerbericht aber nicht immer erscheint, sondern nur wenn auch tatsächlich etwas angezeigt werden soll, empfiehlt es sich, eine globale Variable in ApEx anzulegen. Das Logging Package, mit dem fachliche Fehlermeldungen in die Collection geschrieben werden, wird noch dahingehend erweitert, dass es, wenn eine Logmeldung geschrieben wird, auch gleichzeitig diese Variable z.B. auf 1 setzt (Listing 2). Der Bericht wird dann nur dann angezeigt, wenn auch diese Variable einen Wert hat.

```
PROCEDURE log_to_apex (p_message IN VARCHAR2)
IS
BEGIN
    ApEx_Util.Set_Session_State('F104_SHOW_ERROR',1);
    APEX_COLLECTION.ADD_MEMBER(
        p_collection_name => 'ERROR_COLLECTION',
        p_c001            => P_Message);
END log_to_apex;
```

Listing 2: Fehlermeldungen in Collection schreiben

Mit etwas Vorarbeit ist es sogar möglich, dass die handgemachten Prozessfehlermeldungen kaum von den original ApEx Validierungsmeldungen zu unterscheiden sind. Damit dies funktioniert, muss man sich zunächst genauer mit dem Seitentemplate seiner Anwendung beschäftigen. Damit ApEx erkennen kann, an welcher Stelle auf einer Seite bestimmte Inhalte dargestellt werden sollen, werden im Seitentemplate Ersetzungsstrings definiert. Diese Strings werden zur Laufzeit von ApEx mit generiertem HTML Code ersetzt. Einer dieser Ersetzungsstrings hat den Namen #NOTIFICATION_MESSAGE#. An dieser Stelle werden Validierungsmeldungen erzeugt. Soll der Bericht mit den Fehlermeldungen an derselben Stelle wie die Validierungsmeldungen erscheinen, so muss im Seitentemplate eine Regionsposition an derselben Stelle wie der Ersetzungsstring #NOTIFICATION_MESSAGE# platziert werden, beispielsweise #REGION_POSITION_05#. Auf der Seite 0 kann dem Bericht dann diese Position zugewiesen werden. Nun müssen noch ein spezielles Regionstemplate und ein Berichtstemplate angelegt werden, welche dafür sorgen, dass die Fehlermeldungen in den Berichten so aussehen wie Validierungsmeldungen. Das Berichtstemplate lässt sich sogar soweit steuern, dass verschiedene Logklassen wie beispielsweise Info, Warnung und Fehler unterschiedlich dargestellt werden. Damit gehen die Gestaltungsmöglichkeiten sogar noch über die Möglichkeit von normalen Validierungsmeldungen hinaus.

Damit die Fehlermeldungen auch wieder verschwinden, muss zu guter Letzt auch noch ein Anwendungsprozess erstellt werden, der nach dem Laden der Seiten ausgeführt wird. Dieser Prozess soll ebenfalls nur ausgeführt werden, wenn die globale Variable, die bereits in der Bedingung für die Darstellung des Fehlerberichts verwendet wurde, einen Wert hat. Der Prozess leert den Inhalt der Collection und setzt der Wert der Variable auf NULL zurück, so dass der Bericht beim nächsten Mal nicht mehr angezeigt wird.

Löschen nur mit Sicherheitsabfrage

Eine häufige Anforderung, mit der Entwickler konfrontiert werden, ist, dass das Löschen eines Datensatzes nur nach vorheriger Sicherheitsabfrage erfolgen soll. Diese Anforderung wird in ApEx traditionell umgesetzt, indem statt eines direkten Submits der Seite zunächst eine JavaScript Funktion aufge-

rufen wird. Die JavaScript Funktion sorgt dafür, dass dem Anwender beim Betätigen des Buttons zunächst eine Dialogbox mit der Nachfrage präsentiert wird, ob der den Datensatz wirklich löschen will. Erst wenn der Anwender dies bestätigt erfolgt der eigentliche Submit der Seite und der Datensatz wird gelöscht. Der Nachteil dieser Lösung ist jedoch offensichtlich: Der Entwickler muss bei jedem Delete Button daran denken, die entsprechende Funktion aufzurufen. Eleganter wäre es, wenn dies an einer zentralen Stelle geschehen würde.

Um die Lösung zu verstehen, müssen wir jedoch zunächst einen kurzen Blick unter die Haube von ApEx und das http Protokoll werfen. Das http Protokoll kennt keine Aktion namens „Submit", sondern tauscht Daten mit dem Server mit den Befehlen get und post aus. Die Ausdrucksweise „Submit" kommt aus dem HTML Sprachgebrauch, wenn ein Formular seine Daten an den Server sendet. Ein solches HTML Formular wird auch von ApEx verwendet um die Daten mittels eines http POSTS an den Server zu senden. Um zu erkennen, welche Aktion der User auf der HTML Seite ausgeführt hat verwendet ApEx ein verstecktes HTML Element namens pRequest, das auf allen Seiten in dem Formular vorhanden ist. Wird ein Button in ApEx gedrückt und es soll die Verarbeitungslogik der Seite ausgeführt werden, kommt immer die JavaScript Funktion doSubmit(RequestWert), bzw. seit ApEx 4 die Funktion apex.submit(RequestWert) zum Einsatz. Diese Funktion setzt den Wert des HTML Elements pRequest z.B. auf den Wert DELETE und ruft dann die Funktion submit() des Formulars auf, wodurch der entsprechende POST Request des Formulars ausgelöst wird. Da nun alle Daten, inklusive des versteckten Elements pRequest an den Server gesendet werden, weiß ApEx anhand dieser Variable, welche Aktion ausgelöst wurde. Aber wo kann in diesen Prozess gefahrlos eingegriffen werden?

```
<script language="JavaScript" type="text/javascript">

  function submitOverride(event) {
    //Wenn der Request DELETE enthält, dann eine Meldung anzeigen
    if($('#pRequest').val().indexOf('DELETE')>-1) {
          $('#ConfirmDeleteMessage').dialog('open');
        }
    else {
    // Original submit ausführen
      this._submit();
    }

  }

 $(document).ready(function() {

    //Originalsubmit Funtkion in andere Variablen Speichern
    document.wwv_flow._submit = document.wwv_flow.submit;
    //Eigene Submitfunktion platzieren.
    document.wwv_flow.submit = submitOverride;

</script>
```

Listing 3: Eigenen Code in die Standardsubmitroutine injizieren

Die Lösung für dieses Problem liegt in der Tatsache, dass JavaScript eine Programmiersprache ist, in der Funktionen wie in anderen Sprachen Objekte behandelt werden. In JavaScript können Funktionen also beispielsweise einer Variablen zugewiesen werden (Listing 3). Dadurch ist es möglich, die Original Submit Funktion gegen eine eigene Submit Funktion zu tauschen, die am Ende wieder das Original

ausführt. Der Code zum Tauschen muss beim Laden der Seite ausgeführt werden. Zur Ausführung des Codes beim Laden der Seite eignet sich besonders gut JQuery mit seiner $document.ready Funktion.

Durch diesen Trick ist es möglich, sich in den ApEx Submit Prozess einzuklinken, ohne den Quellcode von ApEx anpassen zu müssen oder etwas an der eigentlichen Submit Logik von ApEx zu ändern. In der Funktion die wir nun als Standardsubmit Funktion eingeschmuggelt haben, können wir nun unseren eigenen JavaScript Code platzieren. Im Beispiel wird der aktuelle Wert des Elements pReqeust geprüft. Enthält dieser den String DELETE, so soll eine Message Box mit der Nachfrage zum Löschen angezeigt werden. Der große Vorteil dieser Variante ist, dass Sie sich auch nachträglich mit wenig Aufwand in bestehende Anwendungen integrieren lässt.

Wem die Default JavaScript Box nicht schön genug ist, der kann beispielsweise auf Dialogboxen von JQueryUI zurückgreifen. JQueryUI ist eine Erweiterung der JavaScript Bibliothek JQuery. Mit ApEx 4.0 sind diese beiden JavaScript Frameworks in den Standardlieferumfang von ApEx gewandert und werden auch von den internen Funktionen häufig genutzt. Die Dialogboxen können mit CSS gestylt werden. Neben ästhetischen Gesichtspunkten hat die Verwendung von JQueryUI Dialogen auch den Vorteil, dass die Anzahl und die Beschriftung der Buttons gesteuert werden kann (Abb. 3.)

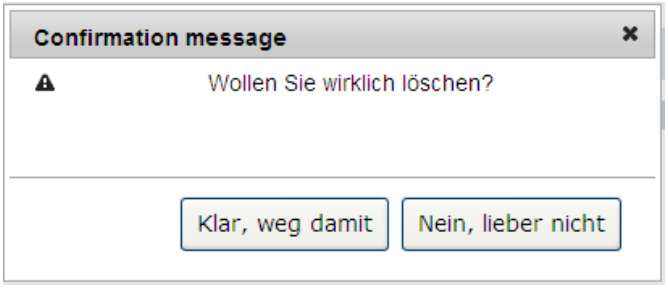

Abb. 3: Dialogbox mit individuellen Buttons

Prozesse nur ausführen, wenn Daten geändert wurden

Ein weiterer Anwendungsfall der mit der gleichen Technik abgedeckt werden kann ist die Anforderung, nur dann einen Prozess auszulösen, wenn sich auf der Seite Daten geändert haben. Hierzu könnte beispielsweise mit JQuery und einer weiteren Hilfsbibliothek eine Prüfsumme über alle Elemente der Seite gebildet werden. Beim Submit wird nun geprüft, ob das Element pRequest den String „SAVE“ beinhaltet. Wenn dies der Fall ist, so wird erneut eine Prüfsumme gebildet. Unterscheiden sich beide Summen, wurden Daten auf der Seite geändert und das Formular wird wie gewohnt abgesendet. Ist die Summe gleich kann beispielsweise der Submit abgebrochen und dem Benutzer eine Benachrichtigung angezeigt werden, dass es nichts zu speichern gibt. Alternativ kann auch der Request so manipuliert werden, dass es im Backend zu keiner Aktion mehr kommt außer dem gewünschten Sprung. Damit diese Variante funktioniert muss allerdings bei der Wahl der Requestwerte etwas Planungsaufwand investiert werden. Dafür kann aber mit generischem JavaScript viel einfacher als in PL/SQL die Prüfung auf geänderte Daten implementiert werden, da in man in PL/SQL notgedrungen in jeder Prozedur die Prüfung auf geänderte Daten, beispielsweise ebenfalls mit einer Prüfsumme, ausprogrammieren muss.

Navigation innerhalb einer Anwendung mit „Zurück“ Buttons

Entwickelt man mit ApEx eine Anwendung, so wird man im Regelfall den Benutzer durch Buttons innerhalb der Anwendung bestimmte Aktionen ausführten lassen. Dabei kann davon ausgegangen werden, dass in jeder Anwendung auf nahezu jeder Seite auch ein „Bring den Benutzer auf die vorherige Seite ohne irgendwelche Aktionen auf der aktuellen Seite zu manipulieren“ Button platziert werden muss. Wie kann dies effektiv umgesetzt werden?

Der erste Ansatz ist meistens der, bei den Back Buttons hart codiert zu hinterlegen, auf welche Seite zurück gesprungen werden soll. Dieser Ansatz kommt jedoch an seine Grenzen, wenn eine Seite von 2 verschiedenen Seiten aus erreicht werden kann. Für diesen Fall hat ApEx bereits eine Lösung vorgesehen: Man kann festlegen, dass der Inhalt eines Elements angibt, auf welche Seite verzweigt werden soll. Wird die Seite angesprungen muss beim Sprung dafür gesorgt werden, dass dieses Element entsprechend der Ursprungsseite gefüllt wird. Dieses Vorgehen hat zwei Nachteile: Zum einen muss immer wenn die Seite angesprungen wird auch die „Rücksprungadresse“ mit angegeben werden. Zum anderen ist man gezwungen, auf jeder Seite ein entsprechendes Element für die Rücksprungadresse vorzuhalten. Ein globales Element funktioniert nicht, da beim Sprung von Seite 2 auf Seite 3 die Information verloren ginge, dass vor Seite 2 die Seite 1 aufgerufen wurde.

Eine Lösung für das Problem ist die Implementierung eines Navigationsstacks. Dieser Stack muss die gesamte Seitenhistorie des Nutzers speichern. Um einen solchen Stack zu implementieren, eigenen sich Collections am besten, da diese bereits eine Sessionverwaltung mitbringen. Die Collection muss

```
DECLARE
   v_last_page   PLS_INTEGER;
BEGIN
   SELECT c001
     INTO v_last_page
     FROM (SELECT   seq_id, c001
                FROM apex_collections
               WHERE collection_name = 'NAVIGATION_COLLECTION'
            ORDER BY seq_id DESC)
    WHERE ROWNUM < 2;

   IF v_last_page != :app_page_id
   THEN
      apex_collection.add_member
       (p_collection_name      => 'NAVIGATION_COLLECTION',
        p_c001                 => :app_page_id
       );
   END IF;
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      apex_collection.add_member
       (p_collection_name      => 'NAVIGATION_COLLECTION',
        p_c001                 => :app_page_id
       );
END;
```

Listing 4: Seiten auf den Stack legen

dabei zunächst initialisiert werden, was am besten im POST Authentication Prozess erledigt wird. Um den Pfad der Benutzernavigation zu speichern wird am besten ein Anwendungsprozess definiert, der auf jeder Seite beim Laden ausgeführt wird. Der Prozess muss prüfen, ob die aktuelle Seite dem obersten Element auf dem Stack entspricht. Ist dies nicht der Fall, so muss die Seite auf den Stack gelegt werden (Listing 4). Damit der generische Effekt nicht dadurch zunichte gemacht wird, dass auf jeder Seite ein entsprechender Prozess angelegt werden muss der – im Falle eines Rücksprungs – den Stack dekrementiert, soll ein weitere Anwendungsprozess angelegt werden, der in der Phase des Seitenverarbeitung noch vor den Berechnungen läuft und der das zweite Element (das Erste ist die aktuelle Sei-

te) vom Stack holt, den Stack dekrementiert und auf die entsprechende Seite einen Redirect macht (Listing 5).

```
DECLARE
   v_last_page   PLS_INTEGER;
   v_seq_id      PLS_INTEGER;
BEGIN
   -- Vorherige Seite vom Stack ermitteln
   SELECT c001
     INTO v_last_page
     FROM (SELECT seq_id, c001,
                  ROW_NUMBER () OVER (ORDER BY seq_id DESC) seq
             FROM apex_collections
            WHERE collection_name = 'NAVIGATION_COLLECTION')
    WHERE seq = 2;
   -- Aktuelle Seite vom Stack ermitteln, damit sie gelöscht werden kann
   SELECT seq_id
     INTO v_seq_id
     FROM (SELECT seq_id, c001,
                  ROW_NUMBER () OVER (ORDER BY seq_id DESC) seq
             FROM apex_collections
            WHERE collection_name = 'NAVIGATION_COLLECTION')
    WHERE seq = 1;
   -- Aktuelle Seite löschen
   apex_collection.delete_member
                                (p_collection_name      =>
'NAVIGATION_COLLECTION',
                                 p_seq                  => v_seq_id
                                );
   -- Redirect
   OWA_UTIL.redirect_url (   'f?p='
                          || :app_id
                          || ':'
                          || v_last_page
                          || ':'
                          || :app_session
                         );
   -- Verarbeitung abbrechen
   apex_application.g_unrecoverable_error := TRUE;
EXCEPTION
   -- Keine Daten auf dem Stack, kein Redirect.
   WHEN NO_DATA_FOUND
   THEN
      NULL;
END;
```

Listing 5:Automatischer Rücksprung zur vorherigen Seite

Allerdings wird bei diesem Ansatz die Kette des Navigationspfades niemals unterbrochen, was nicht immer zum gewünschten Ergebnis führt. Beispielswiese ist es nach einem Speichern Vorgang nicht unbedingt sinnvoll, mit einem Zurück Button wieder auf die Seite zu navigieren, die soeben verlassen

wurde. Um diese die Kette des Navigationspfades zu unterbrechen, gibt es zwei unterschiedliche Ansätze:

- Der Prozess, der die Seiten auf den Stack legt, prüft, ob es auf der aktuellen Seite einen Zurück Button gibt. Dies kann mit Hilfe des ApEx Dictonary geschehen. Existiert kein „Zurück" Button wird der Stack geleert. Dies funktioniert natürlich nur, wenn die „Zurück" Buttons auf den Seiten einem bestimmten Muster folgen, z.B. immer „BACK" als Request eingetragen haben, damit sie zuverlässig über das Dictonary erkannt werden können. Dies ist jedoch auch Voraussetzung auch für einen generischen Redirect Prozess.
- Der Prozess der die Seiten auf den Stack legt prüft, ob die aktuelle Seite bereits an irgendeiner Position auf dem Stack liegt. Wenn ja, werden alle Elemente des Stacks, die nach dieser Seite kommen, gelöscht. Dadurch ist zwar keine zyklische Navigation innerhalb der Anwendung (Von Seite 3 nach Seite 4 von dort weiter zu Seite 3) möglich, die ist jedoch sowieso nur in wenigen Anwendungen notwendig.

Natürlich kann der Stack auch bewusst manipuliert werden, um die Navigation auf speziellen Seiten an besondere Bedürfnisse anzupassen.

Fazit

Die hier vorgestellten Lösungen ermöglichen es, einige typische Standardanforderungen einfach und schnell umzusetzen. Dadurch werden Entwickler entlastet und dem Anwender ein hoher Bedienkomfort beschert. Ein weiterer Vorteil der hier vorgestellten Lösungen ist, dass nur wenig Code an zentralen Stellen werden platziert werden muss. Durch die Vermeidung von redundantem Code wird die Wahrscheinlichkeit von Programmierfehlern minimiert. Dadurch werden ApEx Anwendung noch einfacher zu entwickeln, besser zu bedienen, durchgängiger und fehlerfreier – kurz: effektiver.

Kontaktadresse:

Stephan Engel
OPITZ CONSULTING Bad Homburg GmbH
Kaiser-Friedrich-Promenade 93-96
D-61348 Bad Homburg

Telefon: +49 (0) 6172 66260 1535
E-Mail stephan.engel@opitz-consulting.de
Internet: http://www.opitz-consulting.com