# JSXP, Just Simple eXtensible

**David Tanzer / Oliver Szymanski**
**Freiberufler**
**Linz / Erlangen**

**Schlüsselworte:**

Rapid Web-Application Framework

## Einleitung

JSXP steht für „Just Simple eXtensible Pages“ und ist ein Open Source Web-Anwendungsframework, für das seit kurzem ein Release Candidate existiert. Wie der Name des Frameworks schon sagt, wurde bei der Entwicklung auf Einfachheit aber flexibler Erweiterbarkeit großer Wert gelegt. Außerdem bietet das Framework eine enge Zusammenarbeit zwischen Designer und Entwickler.

JSXP wurde mit der Zielsetzung entwickelt, Rapid Application Development und eine reibungslose Zusammenarbeit von Designern und Entwicklern zu ermöglichen. Dazu baut das Framework auf folgenden Kern-Prinzipien auf:

### Compiler-Prüfung

Verbindung zwischen Design und Code: Durch aus Designvorlagen generierten Code wird sichergestellt, dass im Code nur Elemente referenziert werden können, die im Design auch tatsächlich existieren. Deshalb werden viele Fehler zur Übersetzungszeit erkannt und müssen nicht durch Unit - Tests gefunden werden.

### Agile Entwicklung

JSXP erlaubt das Design während des Projekts anzupassen und dabei Zugriff von Designern und Entwicklern auf dieselben Dateien. Insbesondere durch die bei JSXP weiterhin gewährleistete Originalstruktur und Ansicht der Designdateien wird dieser iterative und agile Prozess möglich. Desweiteren kann man nach und nach reine statische Designvorlagen durch die Implementierung mit dynamische Ansichten ersetzen. JSXP erlaubt von Beginn an eine Navigation über die gesamte Webanwendung in Form der Designvorlagen, so dass stets eine testbare und im Laufe des Projekte um Funktionalität erweiterbare Anwendung vorliegt.

### Trennung von Code und Design

Es ist nicht möglich, Code in die Design-Dateien zu mischen, wie das zum Beispiel bei JSF (durch die Expression Language) der Fall ist. Implementierung erfolgt ausschließlich in reinem Java – der Sprache die wir alle beherrschen, an Stelle einer weiteren.

### Einfaches (X)HTML – Design

Der Web-Designer arbeitet mit normalen, gültigen HTML-Dateien und kann somit jedes beliebige Werkzeug zum Bearbeiten von HTML verwenden. Die einzige Einschränkung, die der Standard – Parser von JSXP bietet, ist dass diese Dateien gültiges XML enthalten müssen (d.h. dass zum Beispiel alle Tags geschlossen werden müssen).

### IDE Unterstützung

Da JSXP auf reinem HTML und Java basiert, sind keine besonderen Tools notwendig, sondern JSXP ist in jede IDE einfach integrierbar. Durch den generierten Code gibt es trotzdem eine Syntaxprüfung und eine Konsistenzprüfung der Designvorlagen und mit der Implementierung. Somit ist stets sichergestellt, dass man mit den bereits eingesetzten Tools ohne weiteres JSXP-Projekte durchführen kann. Man ist nicht von speziellen Plugins abhängig und sowohl der Designer kann die Vorlagen als auch der Entwickler die Implementierung auf die Weise erstellen, die er gewöhnt ist.

### Lesbare URLs

In vielen Fällen wird durch das Framework selbst sichergestellt, dass die URLs von Seiten „bookmarkable“ und lesbar sind. Für die Fälle, wo das nicht automatisch passieren kann, wird dem Programmierer eine Möglichkeit gegeben selbst solche URLs auf einfache Weise zu realisieren.

### Weitere Features

Außerdem unterstützt das Framework Komponentenorientierung, Element Templating, View Templating, Internationalisierung, einfache Verwaltung von Server-seitigem Status, View Flows und AJAX. Über das Resource Handling System kann nicht nur eine der Arten von Internationalisierung realisiert werden, es können auch Resourcen von eingebetteten Komponenten (wie z.B. CSS oder Java Script Dateien) verwaltet werden.

### Entwicklung einer ersten JSXP–Seite

In JSXP geht man davon aus, dass ein Design vorliegt, um eine Seite zu entwickeln. JSXP ist somit Design-Driven. Dabei kann das Design beliebig oft innerhalb des Projektes geändert werden. Man kann also mit einem Prototyp bis hin zur späteren richtigen Anwendung das Projekt entwickeln. Das Design kann eine beliebige XML-Datei sein. Wir starten hier mit einer HTML-Seite, die XML-konform ist:

index.html:

```
<html>
    <body/>
</html>
```

Aus diesem Design View erzeugt der JSXP Generator die Basisklasse für unseren View Controller: „IndexHtmlControllerGenerated.java“. Der Generator wird über ein Shell-Skript in den Build-Vorgang eingebaut. Beispielsweise kann über die „Builder“ - Einstellungen eines Eclipse-Projekts konfiguriert werden, dass diese Basisklassen immer dann generiert werden, wenn sich eine (x)html Datei ändert. Somit hat man immer direkt die Syntax und Konsistenzprüfungen.
Nachdem die Basisklasse generiert wurde, kann die Webseite schon in einem Web-Container „deployed“ werden. Wenn man nun „index.html“ aufruft, sieht man, dass nicht nur eine leere Seite kommt, sondern eine ganze Menge Debug-Meldungen des JSXP Frameworks angehängt sind. Das JSXP-Framework löst zum Namen „index.html“ die dazugehörige View-Controller Klasse auf und führt dort den Lebenszyklus der Seite aus, was unter anderem zu den Debug-Meldungen führt.
Streng genommen sind wir also schon fertig mit der Entwicklung unserer ersten JSXP-View. Wir wollen der Seite aber noch etwas Funktionalität hinzufügen. Dazu müssen wir einerseits auf Komponenten des Design-Views aus dem Code zugreifen können und andererseits auf Ereignisse im Lebenszyklus reagieren.
Für unser Beispiel nehmen wir an, dass wir in einem iterativen Prozess eng mit einem Designer zusammenarbeiten. Der Designer hat uns also inzwischen eine neue Version der Seite geliefert, die jetzt einen Logout-Link und einen Administrations-Link enthält. Um diese beiden Links aus dem Java – Code ansprechen zu können, vergeben wir erst mal IDs:

index.html:

```
<html>
    <body>
        <a href="admin.html" jsxp:id="adminLink">Admin</a>
        <a href="index.html jsxp:id="logoutLink">Logout</a>
    </body>
</html>
```

Die IDs stören das Design nicht weiter, so dass die Vorlage immer noch mit jedem HTML-Tool angezeigt werden kann. Wir wollen jetzt den Logout-Link verschwinden lassen, wenn der Benutzer nicht eingeloggt ist. Dazu leitet man von der generierten Controller-Klasse ab:

IndexHtmlController.java:

```
[...]
public class IndexHtmlController extends IndexHtmlControllerGenerated<Object> {
    @Override
    protected void init(Importer importer) throws Exception {
        if(!((UserContext)Context.getContext().getUserContext()).isLoggedIn()) {
            getElementLogoutLink().remove();
        }
    }
}
```

Init ist eine der Methoden, die im Lebenszyklus der Komponente aufgerufen wird. Hier kann man eine allgemeine Initialisierung der Komponente durchführen, während man Geschäftslogik in „execute" packt. Diese wird nur aufgerufen, wenn „validate" (ein vorheriger Schritt im Lebenszyklus zur Validierung von Eingaben) keine Fehler geliefert hat.
Um die Methode „isLoggedIn" am UserContext aufrufen zu können muss noch ein eigener User-Context implementiert und dieser an der Web-Application registriert werden.
Das eigentliche Entfernen des Logout-Links ist nur noch ein Methodenaufruf. Die get-Methode, die das benötigte Element liefert, wurde vom Generator anhand der jsxp:id generiert und steht in der Basisklasse zur Verfügung. Durch die Verwendung einer speziellen Methode „getElementLogoutLink" passend zur ID in der Vorlage ist gewährleistet, dass man Fehler wie versehentliche Umbenennung oder Löschung des Links aus der Vorlage bereits in der Kompilierungsphase und beim Build bemerkt. Bei den meisten anderen Frameworks treffen einen solche schnell ausgelösten Fehler zur Laufzeit – und was ist schlimmer, als in einem laufenden System einen Fehler zu haben und diesen langwierig analysieren zu müssen?
Auf die Seite „admin.html" soll man aber natürlich nur Zugriff haben, wenn man eingeloggt ist. Also muss als nächstes der Admin – Bereich abgesichert und der Benutzer gegebenenfalls auf eine Login-Seite weitergeleitet werden.

**Absichern des Admin – Bereichs**

Um festzulegen, welche Seiten ein Benutzer sehen darf, gibt es 2 Methoden im UserContext, die man als Entwickler überschreiben kann: „public boolean isAllowedToAccessUri (String uri)" und „public boolean isAllowedToViewPage (ViewController<?> view, String uri)". Die erste der beiden Methoden wird aufgerufen, bevor das Framework versucht, den View-Controller zu der URI zu finden, und kann verwendet werden um URIs generell zu sperren. Die zweite Methode wird

aufgerufen, nachdem der View-Controller gefunden wurde und wird verwendet, um den Zugriff auf bestimmte View-Controller einzuschränken. Deshalb verwenden wir jetzt diese Methode:
So sieht ein Listing aus:

```
public class UserContext extends HttpUserContext {
      @Override
      public boolean isAllowedToViewPage
         (ViewController<?> view, String uri) {
            if((view instanceof AdminHtmlController.class ||
               uri.indexOf("admin") > 0) && !isLoggedIn()) {
                  return false;
            }
            return true;
      }
}
```

Versucht man nun, die URL „admin.html“ auzurufen, so bekommt man einen HTTP-Fehler 403 (Zugriff verboten). Als nächstes soll der Benutzer in diesem Fall auf eine Login-Seite weitergeleitet werden. Dazu legen wir eine leere HTML-Seite an (nur HTML und Body tags), damit eine View-Controller Basisklasse generiert wird. Der Designer wird die Login-Seite befüllen, während wir die Weiterleitung implementieren.
Um den Benutzer weiterzuleiten, verwenden wir einen sogenannten View Alias:

```
public class Application extends WebApplication{
    @Override
    public ViewAlias getAliasFor(
      String sourcePath, String targetPath, Map<String, Object[]> parameter) {
        if(!Context.getContext().getUserContext().isAllowedToViewPage(null, targetPath)) {
            return new ViewAlias(new LoginHtmlController(), parameter);
        }
        return null;
    }
}
```

Hierbei handelt es sich um eine einfache Möglichkeit, für eine bestimmte URI einen anderen View auszuliefern als standardmäßig vorgesehen wäre. Damit kann man nicht nur Weiterleitungen realisieren, sondern auch URLs lesbarer machen: Über die URL „some-page-that-includes-a-query.html“ kann man z.B. die Seite „page.html?id=17“ abrufbar machen, wie man dies von gängigen Blogs und Content Management Systemen kennt.
Nun wird der Benutzer zum Login-Formular weitergeleitet, das als nächstes ausgewertet werden muss. Hierzu müssen keine jsxp:ids vergeben werden, aber als „action“ eines Formulars setzen wir hier „login.command“, um die Eingaben später komfortabel auswerten zu können.

**Schlusswort**
Dies waren einige der Features, die JSXP bietet. Weitere Features wie Element-Tempating, Formulareingabenauswertung und Beispiele und Erläuterung der Funktionalitäten findet man unter www.jsxp.org sowie ein Tutorial-Video, in dem eine vollständige Webanwendung in einer Stunde live erzeugt wird auf www.source-knights.com. Gern stehen die Autoren natürlich auch für Antworten auf Fragen und Hilfen/Unterstützung zur Verfügung.

**Kontaktadresse:**

**Name**
David Tanzer
E-Mail david@davidtanzer.net
Internet: www.davidtanzer.net

Oliver Szymanski
E-Mail oliver.szymanski@source-knights.com
Internet: www.source-knights.com