02-2015 | Sommer | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

JAVA IST SUPER STARK

Programmierung
JavaScript für Java-Entwickler

Cloud Computing
Software-Architekturen in wolkigen Zeiten

Applikationsserver
JBoss vs. WebLogic Server

JavaServer Faces
Interview mit Spec Lead Ed Burns

4 191978 304903 02

iJUG Verbund

In Xtend geschriebener Code ist meist kompakter und eleganter als sein Java-Äquivalent

Der Autor ist ein sehr großer Fan von Java, aber auch von C#. Er stellt einige interessante Seiten von C# vor

Die Korrektheit erzeugter PDF-Dokumente überprüfen – nicht manuell, sondern automatisiert

Einstieg in die Liferay-Portal-Entwicklung unter Verwendung von JSF

Frank Schlinkheider und Wilhelm Dück, ITSD Consulting GmbH

Liferay ist der am meisten verbreitete Portal-Server und JSF ist der offizielle Java-Standard für die Web-Entwicklung. Dieser Artikel zeigt, wie beide Technologien zusammenspielen und auf welche Art und Weise mit JSF einfache und effektive Liferay-Portlets entwickelt werden. Dabei lassen sich vorhandene Funktionalitäten wie der Team-Kalender, das Berechtigungssytem und der WYSIWG-Editor einfach nutzen. Zudem werden Gründe für den Einsatz geliefert, aber auch die Nachteile nicht verschwiegen.

Für Liferay sprechen viele Gründe; hier die wichtigsten:

- Unterstützung aller gängigen Industrie-Standards
- Erfüllung aller aktuellen Anforderungen eines Portal-Servers (Navigation, Themes, umfangreiches Rechte/Rollen-Konzept, SSO, Dokumenten-Management etc.)
- Es ist das einzige Produkt aus dem Open-Source-Umfeld, das laut Gartner mit Produkten von Microsoft, IBM und SAP mithalten kann
- Großer Funktionsumfang (Wiki, Foren, Benachrichtigungen, Sozialnetzwerk
- Komponenten etc.)
- Große aktive Community
- Workflow-Unterstützung (Geschäftsprozesse)
- Offene, flexible SOA-Strategie
- Enterprise-Support

Noch ein Schwergewicht

Man hört hin und wieder, dass Java Server Faces (JSF) nicht mehr zeitgemäß sei. Mit neueren Web-Technologien wie GWT, Wicket, Vaadin oder auch jQuery seien schneller und einfacher komfortable Web-Oberflächen zu entwickeln. Nichtsdestotrotz spricht vieles für den Einsatz des Java-Standard-Frameworks. Es gibt viele Entwickler, die über entsprechendes JSF-Know-how verfügen, und es existieren entsprechende Migrationswege für die Portierung von vorhandenen JSF-Web-Anwendungen zu neuen Portal-Anwendungen. Zudem wird standardmäßig JSF in den unterschiedlichen Portal-Servern mittels der später noch vorgestellten Faces Bridge unterstützt. Dank Komponenten-Bibliotheken wie PrimeFaces etc. lassen sich auch mit dem Standard-Web-Framework (JSF) anspruchsvolle Web-Anwendungen für den Unternehmenseinsatz entwickeln.

Allerdings gibt es auch Nachteile für den Einsatz von JSF in Liferay. Die mit Liferay ausgelieferten Standard-Portlets sind meistens nicht mit JSF entwickelt. Hier findet man Standard-JSP, Scriptlets, Struts, etc. Das bedeutet, für die Anpassung dieser Portlets finden, nach unserer Meinung, teilweise veraltete oder zumindest uneinheitliche Web-Technologien Verwendung. Ein weiterer Nachteil ist, dass JSF Komponentenbibliotheken wie z.B. PrimeFaces über grafische Komponenten verfügen, die sich bzgl. „Look and Feel" etwas anders verhalten als die Standard-Elemente von Liferay. Um ein einheitliches Erscheinungsbild im Portal zu erreichen, sollten die verwendeten Komponenten entsprechend ausgewählt werden. Liferay bietet inzwischen eine Reihe JSF Komponenten, die hier entgegenwirken und dem Layout und der Funktionalität der sonst in Liferay eingesetzten grafischen Bibliothek entsprechen.

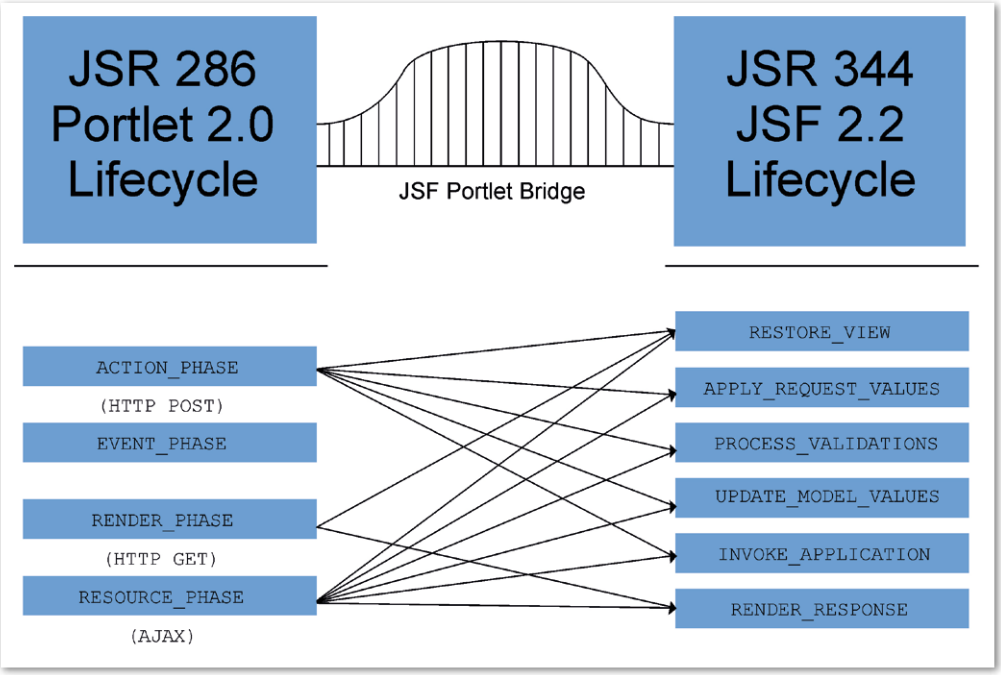


Abbildung 1: Portlet LifeCycle auf Faces LifeCycle mittels Bridge

Wie JSF ins Portal kommt

Nachdem die beiden JSRs für JSF und Portlets bei deren Spezifikation wenig Rücksicht

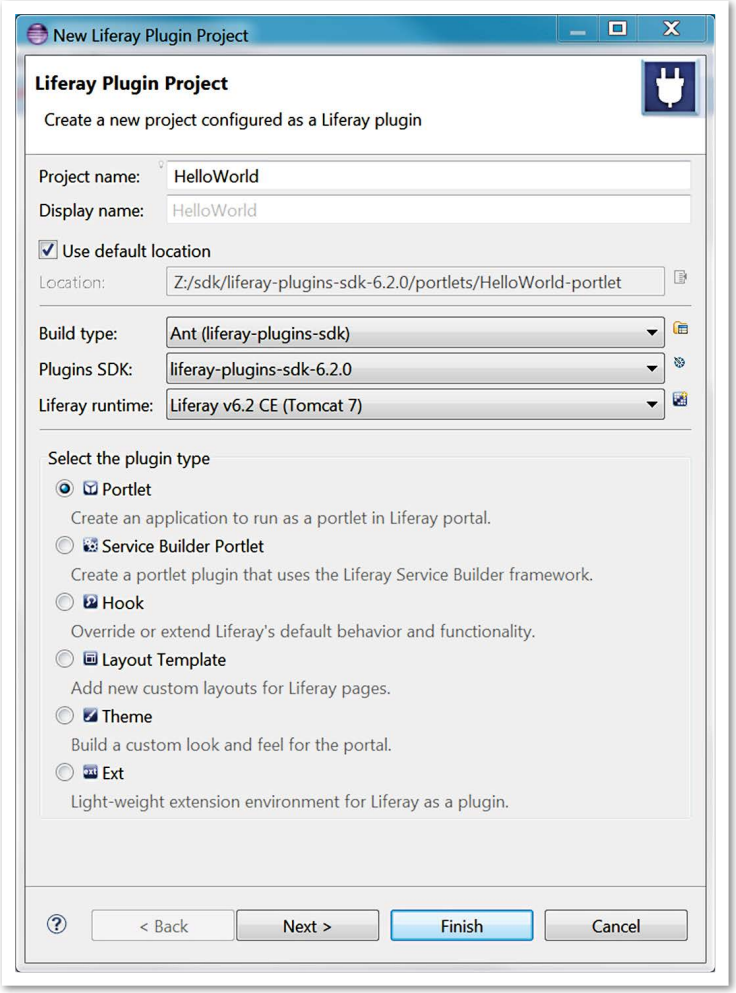


Abbildung 2: Wizard zur Erstellung eines Liferay-Plug-in-Projekts

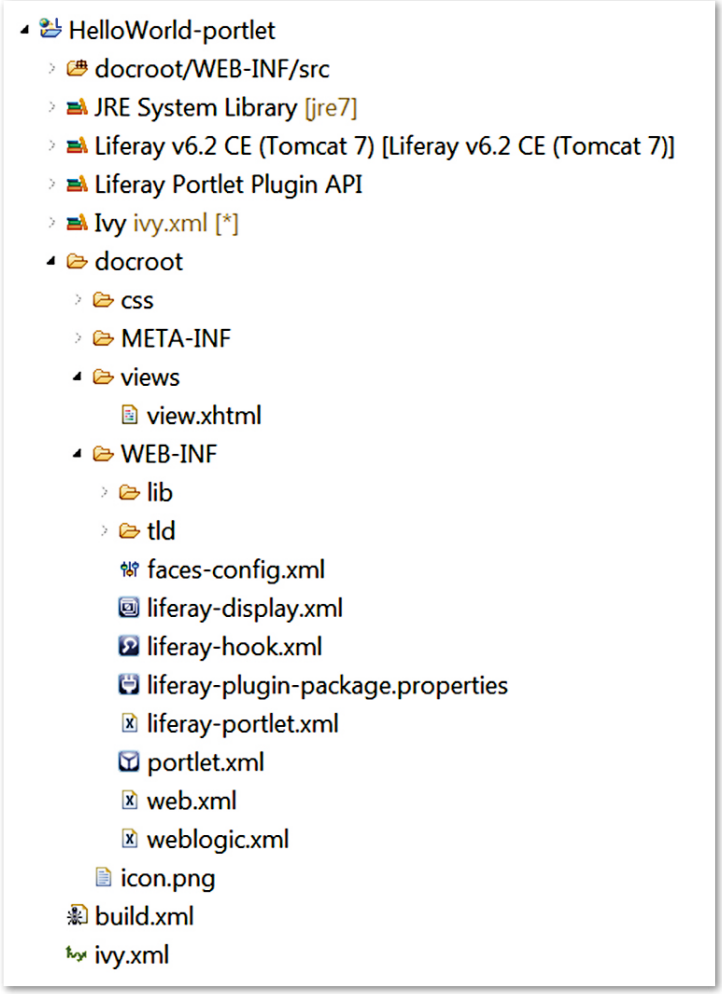


Abbildung 3: Projekt-Struktur eines Liferay-Plug-in-Projekts

aufeinander genommen haben, muss zwischen beiden eine Brücke geschlagen werden. Mit der JSR 301 wurde für Portlet 1.0 und JSF 1.2 und später mit der JSR 329 für die Portlet 2.0 Spezifikation die Faces Bridge ins Leben gerufen. Sie soll folgende Aufgaben erfüllen:

- JSF dem Portlet-Container zur Verfügung stellen
- JSF LifeCycle und Portlet LifeCycle aufeinander abstimmen
- Portal-Funktionalität der JSF-Anwendung verfügbar machen (etwa die Inter-Portlet-Kommunikation)

Abbildung 1 gibt einen einfachen Überblick, wie die Phasen des Portlet LifeCycle auf die des Faces LifeCycle mittels der Bridge abgebildet werden.

Aufmerksamen Lesern wird auffallen, dass hier kein JSR für den Einsatz von Portlet 2.x/3.0 in Kombination mit JSF 2.x genannt wird. Tatsache ist, dass es hier bisher noch keinen Standard gibt. Aktuelle Bridge-Implementierungen unterstützen aber zumindest Portlet 2.0 in Kombination mit JSF 2.x.

JBoss hat eine der ersten Bridge-Implementierungen geliefert. Der Einsatz im Liferay-Portal war aus unterschiedlichen Gründen oft mühsam und schwierig. Bei Problemen saß der Entwickler oft zwischen den Stühlen. Das eine Open-Source-Projekt verwies auf die nicht korrekte Umsetzung der fremden Komponente und umgekehrt. Man kann sagen, dass die beiden sich nicht immer grün waren; trotzdem wurde die Kombination „Liferay und JBoss Faces Bridge" produktiv in Projekten eingesetzt.

Im Jahr 2011 entschied sich Liferay, eine eigene Faces Bridge einschließlich Liferay-Erweiterungen zu entwickeln beziehungsweise weiterzuentwickeln. Folgende Vorteile ergaben sich dadurch:

- Alles aus einer Hand: Keine Schnittstellen-Probleme durch unterschiedliche Hersteller-Implementierungen
- Gute Unterstützung in der Liferay-Entwicklungsumgebung
- Keine Interoperabilitätsprobleme: Jede Liferay-Version besitzt immer eine passende Faces-Bridge-Version
- Einfacher Zugriff auf den Liferay Context mittels EL-Expressions (aktuell angemeldeter User, Permissions etc.)
- Eigene JSF-Tag-Lib, die die Verwendung von Liferay-Controls innerhalb von JSF ermöglicht, etwa für die Verwendung des Liferay-HTML-Editors, der auch für das Wiki, Forum etc. Verwendung findet

„Hello World"-Beispiel mit Wizard

Nach der Theorie ein kleines Praxis-Beispiel: Die eingesetzte Liferay-IDE 2.2.x basiert auf Eclipse Luna und wurde um wichtige Liferay-Plug-ins und Wizards erweitert. Als Erstes entsteht mit dem Wizard ein neues Liferay-Projekt *(siehe Abbildung 2)*.

Für das Beispiel dient der Plug-in-Type „Portlet" und dessen Name soll „HelloWorld" lauten. An den Projekt-Namen wird damit au-

Abbildung 4: Ausgabe des HelloWorld-Portlets (englisch)

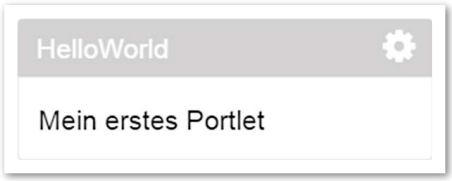


Abbildung 5: Ausgabe des HelloWorld-Portlets (deutsch)

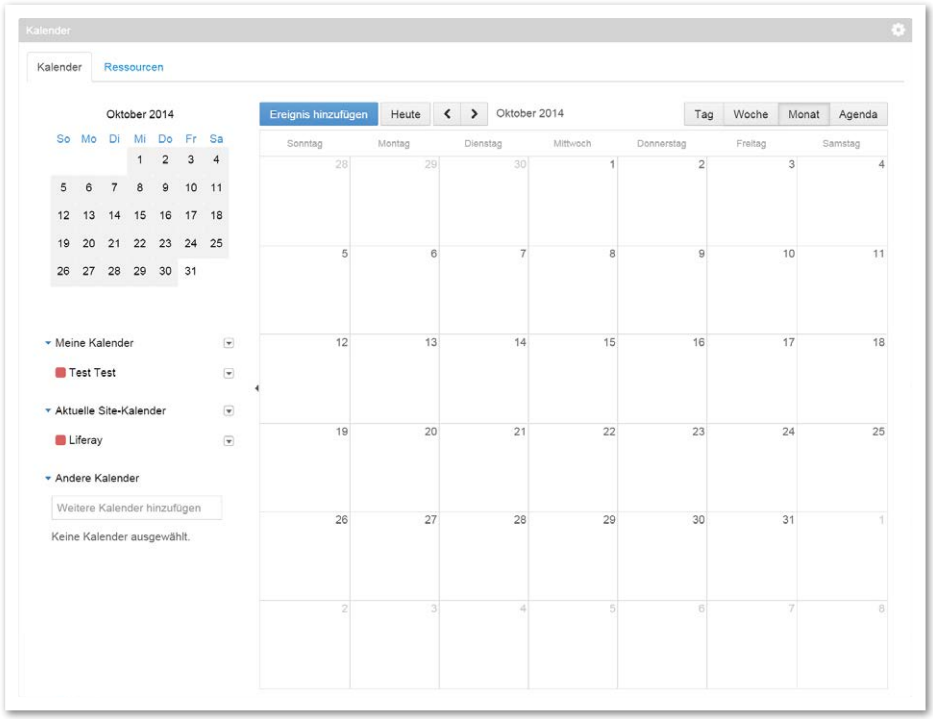


Abbildung 6: Liferay-Kalender-Portlet

tomatisch „-portlet" angehängt. Im nächsten Schritt legt man fest, dass es sich um ein JSF 2.x-Portlet mit der PrimeFaces-Bibliothek handeln soll. Auf diese Art entsteht auf einfache Weise eine JSF-2.x-Portlet-Projekt-Struktur. Die benötigten Komponenten und Bibliotheken (JSF, PrimeFaces etc.) werden dabei automatisch in das Projekt geladen. *Abbildung 3* zeigt die Projekt-Struktur.

Im Verzeichnis „docroot\WEB-INF" sind die Konfigurationsdateien zu finden, insbesondere „portlet.xml", in der die meisten Portlet-spezifischen Konfigurationen vorgenommen werden. Die „xhtml"-Seiten sind im Verzeichnis „docroot\views" und der Classpath für die Java-Dateien lautet „docroot\WEB-INF\src". Das Projekt kann nun direkt auf den zuvor eingerichteten Liferay-Server eingerichtet und das Portlet auf einer Seite hinzugefügt werden. Innerhalb des Portlets wird der Text „Hello HelloWorld!" ausgegeben *(siehe Abbildung 4)*.

Beim genauen Betrachten der Datei „view.xhtml" unter „docroot\view" stellt man fest, dass der Ausgabetext nicht vorkommt. Stattdessen kommt wie üblich Expression Language (EL) zum Einsatz: „<h:outputText value="#{i18n['HelloWorld-hello-world']}" />".

„i18n" steht bekanntlich für „internationalization" und es wird in entsprechenden Sprachdateien für die aktuell eingestellte Sprache nach dem in Hochkommata gesetzten Schlüssel (in unserem Fall „HelloWorld-hello-world") gesucht und der zugeordnete Wert ausgegeben. Der Wizard hatte die „Language_en_US.properties" generiert.

Um sicherzustellen, dass für alle Sprachen wenigstens immer eine englische Begrüßung erscheint, muss die Sprachdatei

```
<hook>
     <language-properties>
        Language.properties
     </language-properties>
  <language-properties>
        Language_de_DE.properties
     </language-properties>
</hook>
```

Listing 1

```
<h:form>
  <h:panelGrid columns="2">
  <!-- Titel -->
  <h:outputText value="Titel:" />
  <h:inputText value="#{calendarBean.title}" />
  </h:panelGrid>

  <h:outputText value="Beschreibung:" />
  <br />
  <!-- Beschreibung über ein Liferay-JSF-Tag -->
  <liferay-ui:input-editor value="#{calendarBean.description}" />

  <h:panelGrid columns="2">
  <h:outputText value="Kalender:" />
  <!-- Auswahl des Kalenders -->
  <h:selectOneMenu value="#{calendarBean.calendarName}">
     <f:selectItems value="#{calendarBean.calendars}" />
  </h:selectOneMenu>

  <h:outputText value="Datum:" />
  <!-- Datum (mit dem Datepicker von PrimeFaces) -->
  <p:calendar value="#{calendarBean.startDate}" pattern="dd/MM/yyyy" />

  <h:outputText />
  <!-- Button zum Speichern (Methode „save" aus CalendarBean) -->
  <h:commandButton action="#{calendarBean.save}" value="Speichern" />
  </h:panelGrid>
</h:form>
```

Listing 2

„Language_en_US.properties" in „Language.properties" im Portlet-Projekt umbenannt werden.

Für die deutsche Übersetzung legt man eine zweite Sprachdatei mit dem Namen „Language_de_DE.properties" an, kopiert den Inhalt aus der Standard-Sprachdatei „Language.properties", fügt ihn in die deutsche Sprachdatei ein und trägt beim entsprechenden Key den Text „Mein erstes Portlet" ein: „HelloWorld-hello-world=Mein erstes Portlet". Damit beide Dateien aktiviert werden, müssen sie als Sprachdateien registriert sein. Dies geschieht in der Datei „liferay-hook.xml" *(siehe Listing 1)*. *Abbildung 5* zeigt dann das Ergebnis.

```
public void save() throws SystemException, PortalException {

  ExternalContext ec = FacesContext.getCurrentInstance()
           .getExternalContext();

  PortletRequest request = (PortletRequest) ec.getRequest();

  // liefert die ID des in der Liste ausgewählten Kalenders
  long calendarId = getCalendarId();

  // aus Titel und Beschreibung werden zwei Maps inkl.
  // Übersetzungen erstellt
  Map<Locale, String> titleMap = new HashMap<>();
  titleMap.put(LocaleUtil.getDefault(), title);
  Map<Locale, String> descriptionMap = new HashMap<>();
  descriptionMap.put(LocaleUtil.getDefault(), description);

  // holen der ServiceContext-Informationen
  ServiceContext serviceContext = ServiceContextFactory
           .getInstance(CalendarBooking.class.getName(), request);

  // Erzeugen eines Kalendereintrages in Liferay
  CalendarBookingLocalServiceUtil.addCalendarBooking(
     Long.parseLong(ec.getUserPrincipal().getName()), calendarId,
     new long[0], 0, titleMap, descriptionMap, "", startDate(),
     endDate(), false, "", 900000, "email", 300000, "email",
     serviceContext);
 }
```

Listing 3

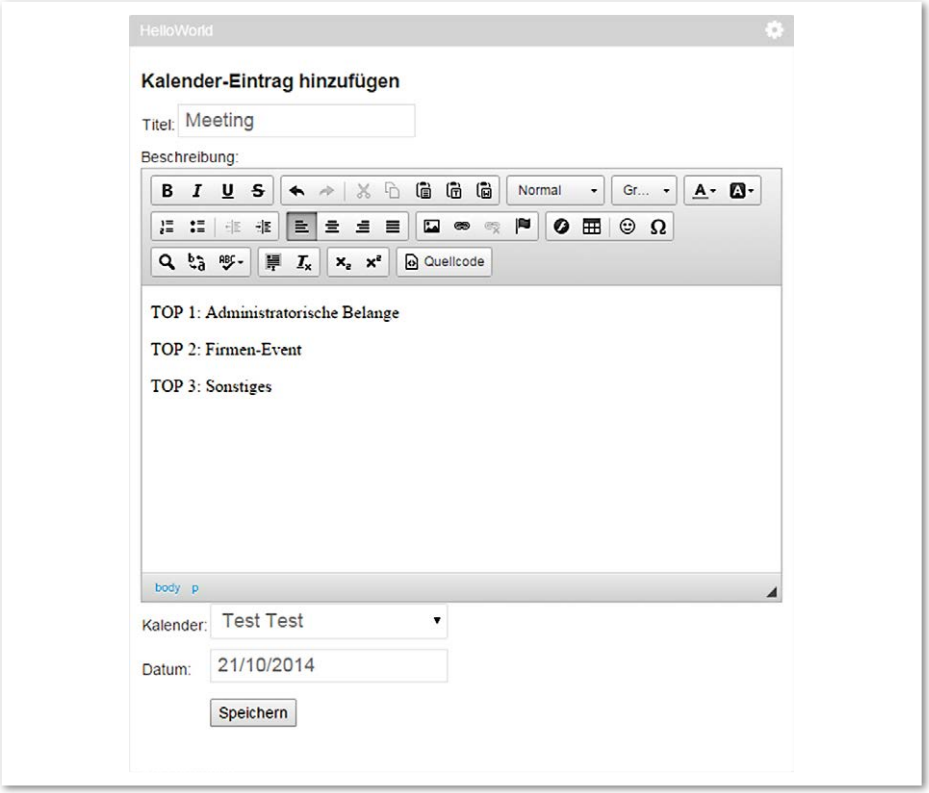


Abbildung 7: „Hello World"-Portlet für die Anlage eines Termins

Anlegen eines Kalender-Events

Eines der von Liferay mitgelieferten Portlets ist „calendar portlet". Dieses wurde für die Version 6.2 von Liferay komplett überarbeitet und verfügt nun über echte Team- und Einladungsfunktionalitäten. Ziel ist es nun, mittels des Portlets einen Team-Kalender-Eintrag zu erzeugen. *Abbildung 6* zeigt den zu Beginn leeren Kalender.

Um das Ziel zu erreichen, werden in der Datei „view.xhtml" die entsprechenden Felder für die Eingabe angelegt *(siehe Listing 2)*. Zusätzlich entsteht die bereits erwähnte Klasse „CalendarBean", die per Annotation als „ManagedBean" gekennzeichnet ist. Sie verwaltet die Attribute „title", „description", „calendarName" sowie „startDate" und bietet eine save-Methode zum Speichern der Eingaben *(siehe Listing 3)*.

Um einen Kalender-Eintrag erstellen zu können, kommt die Service-Klasse „CalendarBookingLocalServiceUtil" des Kalender-Portlets von Liferay zur Anwendung. Sie bietet die Funktion „addCalendarBooking()", die mit den übergebenen Parametern einen neuen Kalender-Eintrag erstellt. Nach dem Deployment des Portlets lassen sich anschließend die Kalender-Informationen eintragen und ein Termin in Liferay erzeugen *(siehe Abbildung 7)*.

Berechtigungen anlegen

Diese Erweiterung zeigt, wie einfach JSF auf das Liferay-Berechtigungssystem zugreifen kann. Um das Anlegen eines Kalender-Eintrages mit einer entsprechenden Berechtigung versehen zu können, ist in der Datei „resource-actions/default.xml" ein Eintrag einzufügen *(siehe Listing 4)*.

Innerhalb des Tags „supports" werden die Berechtigungen definiert, die das Portlet unterstützt. Im Tag „guest-defaults"

```
<permissions>
  <supports>
     <action-key>ADD_ENTRY</action-key>
  </supports>
  <guest-defaults>
  </guest-defaults>
  <guest-unsupported>
     <action-key>ADD_ENTRY</action-key>
  </guest-unsupported>
</permissions>
```

Listing 4

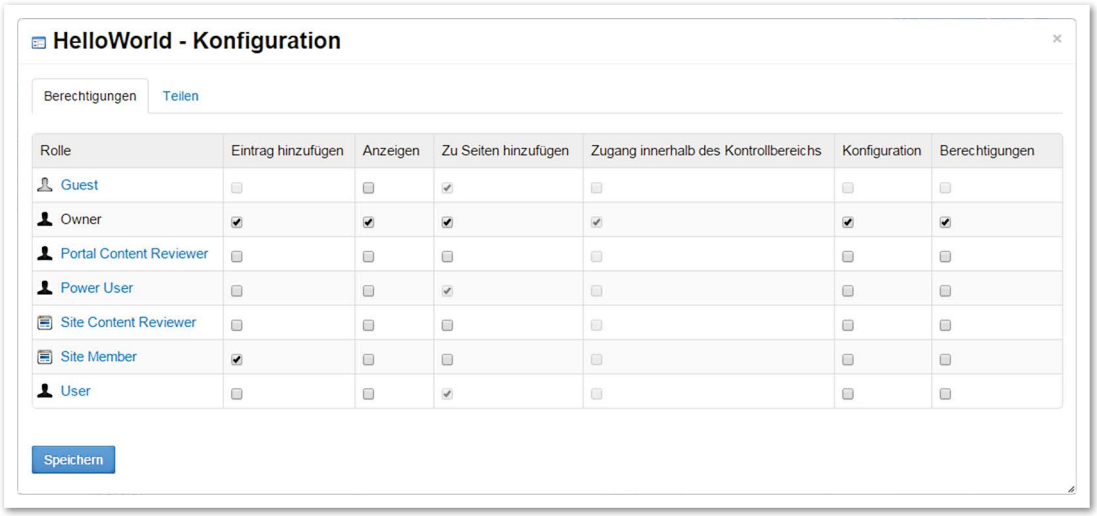


Abbildung 8: Berechtigungen setzen

werden die Standard-Berechtigungen eines nicht angemeldeten Benutzers festgelegt und unter „guest-unsupported" die Berechtigungen festgelegt, die einem unangemeldeten Nutzer niemals zugewiesen werden sollten.

Um die Berechtigungsdefintionen zu registrieren, wird in „portlet.properties" (Datei muss im Classpath liegen) „resource.actions.configs=resource-actions/default.xml" eingetragen. Die Berechtigungen lassen sich nach dem Bereitstellen des Portlets von einem angemeldeten Benutzer mit entsprechender Berechtigung einfach ändern. Dazu muss er im Portal in der oberen rechten Ecke des angezeigten Portlets auf das kleine Zahnrad klicken. Unter „Konfiguration" lassen sich dann die Berechtigungen anpassen *(siehe Abbildung 8)*.

Damit die Berechtigungen auch in der Anwendung Berücksichtigung finden, ist noch die „view.xhtml" anzupassen. Alle Eingabefelder in der „xhtml"-Datei sind innerhalb eines Form-Tag. Dessen „rendered"-Attributs bestimmt, ob die darzustellende HTML-Form inklusive Inhalt angezeigt werden soll oder nicht *(siehe Listing 5)*.

Als „Guest", also als unangemeldeter User, wird nichts angezeigt. Sobald jemand in einer Rolle mit entsprechenden Berechtigungen angemeldet ist, kann er Kalender-Einträge anlegen. Nach der Anlage über das Portlet ist der Eintrag im Liferay-Kalender zu sehen *(siehe Abbildung 9)*.

```
<h:form
rendered="#{liferay.userHasPortletPermission['ADD_ENTRY']}">
  …
</h:form>
```

Listing 5

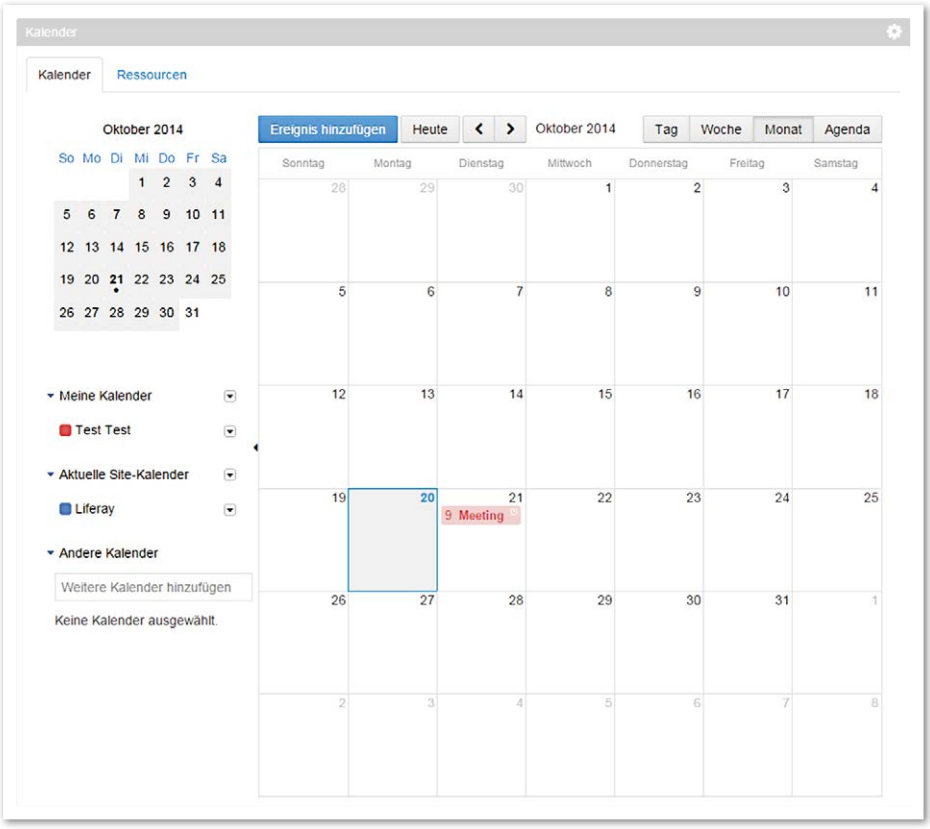


Abbildung 9: Kalender-Portlet mit dem neuen Termin

Fazit

JSF kann auf einfache Weise Portlets für Liferay entwickeln. Die Liferay Faces Bridge liefert alle wichtigen Funktionen, um das Beste aus beiden Welten – Portal-Technologie und Web-Entwicklung – mit JSF zu kombinieren. Seitdem Liferay eine eigene Faces Bridge liefert, stehen dem JSF-Entwickler alle Möglichkeiten der Portlet-Entwicklung einschließlich einfachen Zugriffs auf Liferay-

Core-Funktionalitäten zur Verfügung. Wer allerdings eine Plattform-unabhängige Entwicklung von Portlets anstrebt, sollte bei der Portlet-Entwicklung die zusätzlichen LR-Bridge-Funktionen, die über den Bridge-Standard hinausgehen, mit Bedacht einsetzen. Man kann nur hoffen, dass auch Liferay selbst in den neuen Versionen ihre eigene JSF-Bridge für die Core-Portlets-Entwicklung verwenden wird, auch wenn dieses Vorgehen Veränderungen an den bisherigen Anpassungskonzepten wie Hook oder Ext-Plug-in bedeuten würde.

Weitere Informationen

1. Liferay: http://www.liferay.com/de
2. JavaServer Faces: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

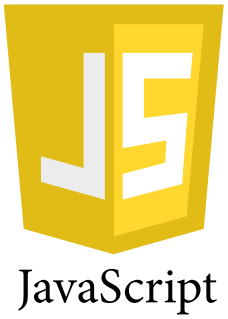
Frank Schlinkheider
fs@itsd-consulting.de

Frank Schlinkheider ist Senior Consultant bei der ITSD Consulting GmbH und dort unter anderem für die Entwicklung von Portal-Lösungen verantwortlich. Neben normalen Liferay-Entwicklungsthemen beschäftigt er sich mit Migrationsmöglichkeiten von vorhandenen IT-Systemen hin zu Enterprise-Portal-Systemen. Darüber hinaus gehört die Umsetzung von BPM-Lösungen (jbpm, Activiti oder Camunda BPM) und die Einführung von Enterprise-2.0-Systemen in Unternehmen und Verwaltungen (etwa mit Social Office) zu seinen Schwerpunkten.

Wilhelm Dück
wilhelm.dueck@itsd-consulting.de

Wilhelm Dück ist bei der ITSD Consulting GmbH als Java-Entwickler tätig und arbeitet seit einem Jahr intensiv mit Liferay in Kombination mit JSF. Neben seiner Tätigkeit als Dozent in Liferay-Schulungen beschäftigt er sich schwerpunktmäßig mit Anpassungen von Liferay-Enterprise-2.0-Systemen.

http://ja.ijug.eu/15/2/5



JavaScript für Java-Entwickler

Niko Köbler, www.javascript-training.net

Ein Workshop im JavaLand 2015 sollte sich kein Java-Entwickler entgehen lassen, wenn er zukunftsfähige (Unternehmens-)Web-Anwendungen entwickeln möchte.

In den letzten zwei Jahren hat sich JavaScript wieder mehr und mehr als ernst zu nehmende Programmiersprache auch in Unternehmensanwendungen durchgesetzt. Sei es im Frontend (Browser), wo sich dank der JavaScript-Frameworks AngularJS, Backbone, Ember etc. die Single-Page-Web-Applications mit HTML5, CSS3 und JavaScript mehr und mehr durchsetzen, oder im Backend auf dem Server, wo sich auch hierzulande „Node.js" mit seinem Ansatz der asynchronen und nicht-blockierenden Request-Verarbeitung („Evented I/O") langsam immer breiter macht. Zusätzlich wird die serverseitige JavaScript-Verwendung dank Nashorn – der JavaScript Engine in der JVM seit Java 8 – immer populärer. Es wird also höchste Zeit für den versierten Java-Entwickler, sich auch mit JavaScript als Programmiersprache auseinanderzusetzen.

Auf den ersten Blick meinen viele Entwickler immer noch, dass JavaScript schon allein aufgrund der Ähnlichkeit im Namen irgendwas mit Java zu tun haben muss. Auch die Sprach-Syntax ist ja sehr ähnlich. Also versuchen sich diese Entwickler mit ersten JavaScript-Programmen und -Anwendungen, ohne aber die Sprache und ihre Eigenschaften genauer kennengelernt und sich mit ihnen beschäftigt zu haben. Dabei ist JavaScript nicht nur aufgrund der Tatsache, dass es eine Script-Sprache ist, der kleine Bruder von Java und nur ein Subset des Sprachumfangs. Die Namens- und Syntax-Ähnlichkeit sind neben dem Grundsatz „Write once, run everywhere" auch die einzigen Gemeinsamkeiten, die beide Sprachen miteinander haben. Doch man kann sie sehr gut zusammen anwenden.

In seinem Workshop im JavaLand 2015 beschäftigt sich der Autor nicht mit der unübersichtlichen Fülle von JavaScript-Frameworks, sondern führt Java-Entwickler in die Sprachgrundlagen von JavaScript ein. Dies kann sehr effizient geschehen, da die meisten Java-Entwickler über ausreichend Vor-