Neue Kolumne: Docker rockt Java ▸ 85

4.2015

JAVA Mag

Javamagazin™

Java • Architekturen • Web • Agile

www.javamagazin.de

Brownies Collections
High-Performance Lists für Java ▸ 30

Selenium
Funktionale Oberflächentests ▸ 90

jax 2015
Programminfos ab Seite 35!

CDI

exensio
intelligente informationssysteme

Sonderdruck für

...ntige Tests von CDI-Komponenten ▸ 46

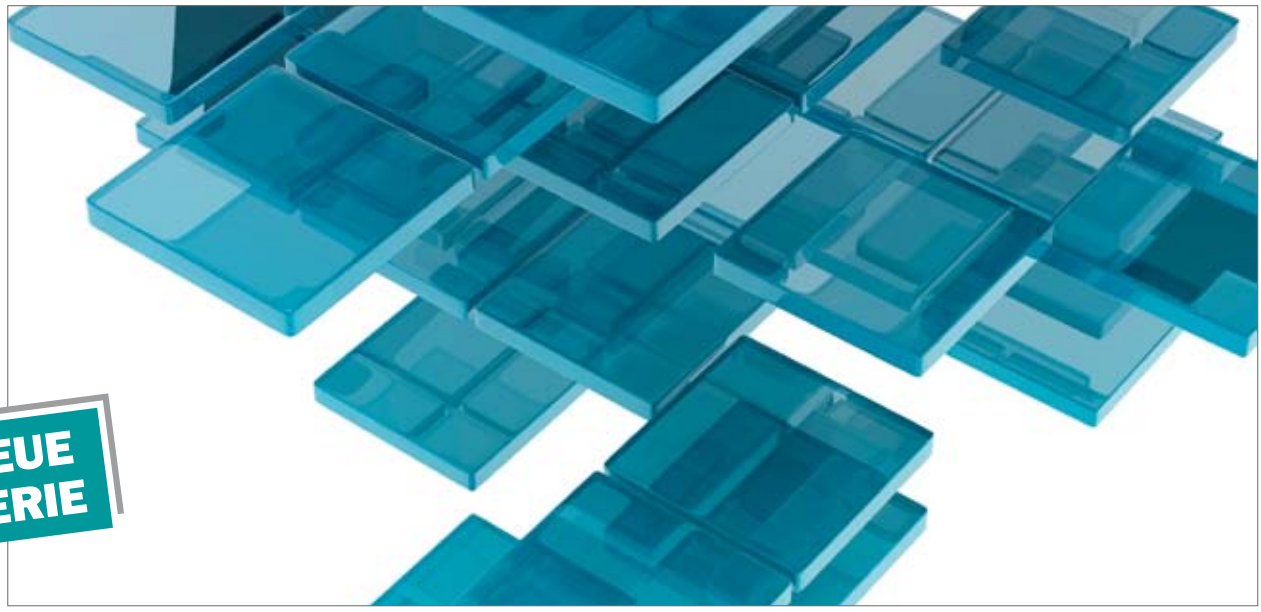
Kontext ist alles: Warum das C den Unterschied macht ▸ 54

©iStockphoto.com/Physicx

Java 8 CompletableFuture: Asynchrone Ereignisse behandeln ▸ 12

Geodaten à la Carte: OpenStreetMap-Daten aufbereiten ▸ 102

Amazon Fire Phone: Fire OS aus Entwicklersicht ▸ 105



NEUE SERIE

© iStockphoto.com/Henvry

Teil 1: Funktionale Tests mit Selenium Builder

Testen mit Selenium

Das Erstellen funktionaler Oberflächentests gehört in Entwicklerteams erfahrungsgemäß zu den weniger beliebten Aufgaben. Mit Selenium Builder existiert ein Werkzeug, mit dem sich diese Aufgaben einfach und ohne tiefere Kenntnisse einer Programmiersprache durchführen lassen.

von Roland Rickborn und Tobias Kraft

Selenium hat sich beim Testen webbasierter Oberflächen bewährt und ist bereits seit über zehn Jahren auf dem Markt. Insbesondere das WebDriver API wird von einigen Testframeworks eingesetzt. Selenium Builder [1] ist ein Instrument zum Aufzeichnen und Abspielen funktionaler Tests auf Basis von Selenium. Es wird in diesem Teil der zweiteiligen Artikelserie näher betrachtet. Im zweiten Teil liegt der Fokus auf der automatisierten und parallelen Ausführung der erstellten Tests mit Selenium Grid [2].

Artikelserie

Teil 1: Funktionale Tests mit Selenium Builder
Teil 2: Selenium Grid

Grundlagen des Selenium Builder

Selenium Builder wurde ursprünglich zur Ablösung der schon länger verfügbaren Selenium-IDE entwickelt, die nicht mehr weiterentwickelt werden sollte. Aktuell wird die Selenium-IDE noch gewartet, wobei sich mit Selenium Builder das Testing nach unserer Erfahrung aber einfacher gestaltet.

Zur Nutzung von Selenium Builder ist keine eigenständige Softwareinstallation notwendig, sondern es muss lediglich ein Firefox-Add-on installiert werden. Das Add-on öffnet sich in Firefox als zusätzliches Fenster, in dem, wie in **Abbildung 1** rechts oben zu sehen, der jeweils aufgezeichnete Testfallschritt gezeigt wird. Aktionen im Hauptfenster des Browsers, wie das Navigieren auf eine andere Seite, spiegeln sich direkt im Add-on-Fenster wider.

Vor dem Erstellen eines Tests muss die gewünschte Selenium-Version 1 oder Version 2 selektiert werden. In

© Software & Support Media GmbH

den folgenden Abschnitten wird die auf WebDriver basierende aktuellere Version 2 betrachtet.

Erstellte Tests können in JSON abgespeichert und somit entsprechend auf Dateiebene strukturiert werden. Neben dem Standardformat wird der Export der Aufzeichnungen in eine Vielzahl von Programmiersprachen wie Java oder Python angeboten. Dies ermöglicht beispielsweise das Einbinden in eine Suite wie JUnit, damit sowohl Unit als auch funktionale Tests in einer einzigen Testsuite ausgeführt werden können. Daneben bietet sich mit Programmiersprachen eine detailliertere Validierung der Testergebnisse an, da bspw. im dortigen Code mit Werten aus einer Datenbank validiert werden kann. Allerdings ist anzumerken, dass dieser Weg nur in eine Richtung funktioniert und exportierte Tests innerhalb von Selenium Builder nicht mehr bearbeitet werden können. Insofern sollte immer das JSON-Format zusätzlich gespeichert und versioniert werden, um zukünftig auf dieser Basis weiterarbeiten zu können.

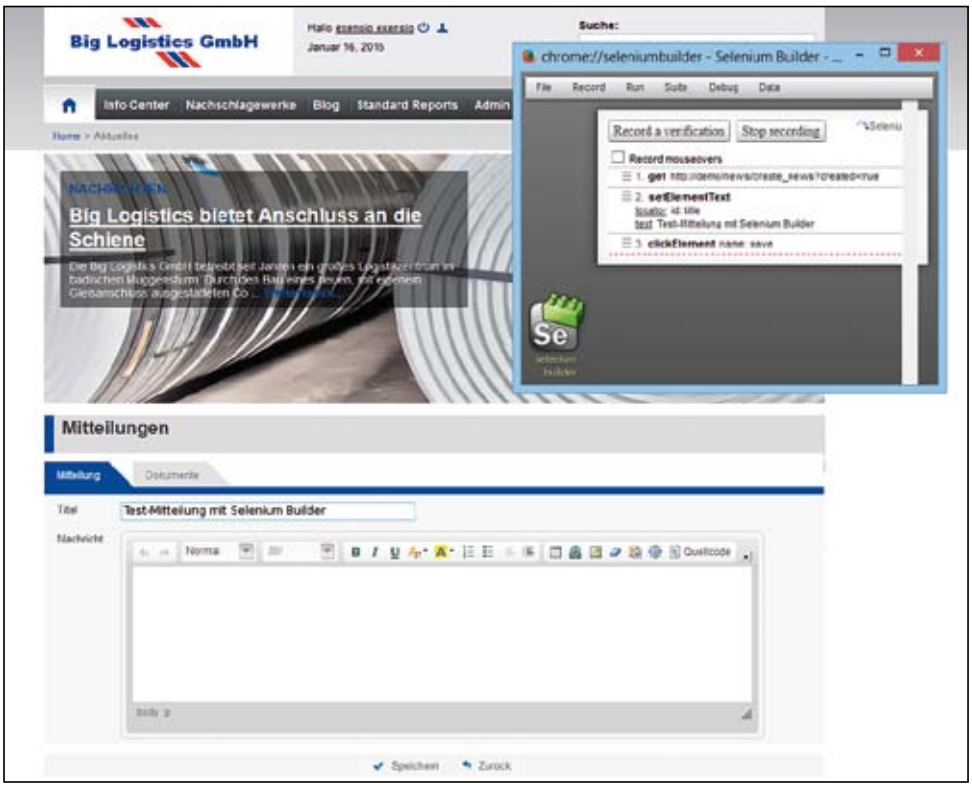


Abb. 1: Browserfenster und separates Selenium-Builder-Add-on

Das Abspielen aufgezeichneter Tests erfolgt direkt lokal über das Add-on-Fenster, das auch immer den aktuellen Testfortschritt anzeigt. Auf die Ausführung über Remote Controller wird im zweiten Teil des Artikels eingegangen.

Aufzeichnen und abspielen

Der Aufnahmemodus speichert eine im Browser geklickte Abfolge und legt sie als JSON-Skript an. Dabei wählt das Tool selbstständig die zu verwendenden Kommandos, wie *clickElement* oder *setElementText*. Als Einstiegsbeispiel wird das Aufrufen eines Formulars und das Ausfüllen eines Textfelds betrachtet. Aufgerufen wird ein Dialog, um neue Mitteilungen in einem Portal anzulegen. Das in **Abbildung 1** dargestellte Formular wird über einen Link aufgerufen und besitzt u. a. ein Textfeld für den Titel der Mitteilung. Nach dem Start des Aufnahmemodus über das Menü RECORD klickt man den Link zum Formular und schließlich im Formular ins entsprechende Textfeld, wo die Eingabe des gewünschten Titels erfolgt.

Listing 1

```
{
  "type": "script",
  "seleniumVersion": "2",
  "formatVersion": 2,
  "steps": [
    {
      "type": "get",
      "url": "http://demo/news/create_news?created=true"
    },
    {
      "type": "setElementText",
      "locator": {
        "type": "id",
        "value": "title"
      },
      "text": "Test-Mitteilung mit Selenium Builder"
    },
    {
      "type": "clickElement",
      "locator": {
        "type": "name",
        "value": "save"
      }
    }
  ],
  "data": {
    "configs": {},
    "source": "none"
  },
  "inputs": [],
  "timeoutSeconds": 60
}
```


© Software & Support Media GmbH

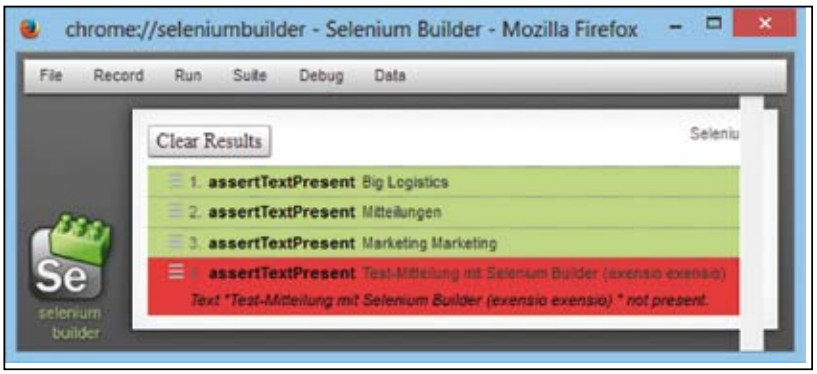


Abb. 2: Anzeige eines fehlgeschlagenen Testschritts

Danach sind im Builder-Fenster die gerade eingegebenen Aktionen zu sehen. Beim ersten Schritt wurde das Kommando *get* verwendet. Der zweite Schritt nutzt das Kommando *setElementText*. Als Selektor in Schritt 2 wurde automatisch die ID des Textelements erkannt. Für das Attribut *text* wurde der Beispieltitel übernommen. Damit ist das erste Beispiel schon fertig (Listing 1), und der Aufnahmemodus wird über den Knopf STOP RECORDING beendet. Das Skript kann über den Menüeintrag RUN TEST LOCALLY getestet werden.

Für die Navigation innerhalb des Browsers stehen die Kommandos *get*, *refresh*, *goBack*, *goForward* und *close* zur Verfügung. Darüber hinaus gibt es das Kommando *clickElement*, mit dem alle Elemente einer Webseite geklickt werden, die auch im Browser angeklickt werden können. Selenium Builder bietet eine komfortable Möglichkeit, Selektoren zu definieren. Dazu definiert man z. B. einen Schritt mit dem *clickElement*-Kommando. Anschließend erfolgt der Klick auf ID, um in den Auswahlmodus für Selektoren zu gelangen. Die zur Verfügung stehenden Selektortypen sind *id*, *name*, *link text*, *CSS-Selector* und *Xpath*. Für die grafische Auswahl eines Selektors wird der Link „Find a different target" gewählt. In diesem Modus lässt sich das gewünschte Element im Browserfenster sehr einfach selektieren. Optional bietet sich der Einsatz des Firefox-Add-ons FirePath [3] an, mit dem sich bequem Selektoren des Typs *XPath* und *CSS* finden lassen.

Selenium Builder kennt zwei Kommandogruppen zur Validierung von Inhalten. Die Kommandos der Gruppe *Verify* werden „weich" überprüft, d. h. im Fehlerfall wird ein Problem dargestellt, aber die Ausführung wird nicht abgebrochen. Ein Fehlerfall von Kommandos der *Assert*-Gruppe führt bei Verwendung dagegen sofort zur Beendigung des Testfalls bzw. zum Abbruch der Schleife.

Listing 2

```
{
  "type": "suite",
  "seleniumVersion": "2",
  "formatVersion": 1,
  "scripts": [
    {
      "where": "local",
      "path": "allgemein/login.json"
    },
    {
      "where": "local",
      "path": "createNews.json"
    },
    {
      "where": "local",
      "path": "allgemein/logout.json"
    }
  ],
  "shareState": true
}
```

Listing 3

```
{
  "type": "script",
  "seleniumVersion": "2",
  "formatVersion": 2,
  "steps": [
    ...
    {
      "type": "setElementText",
      "locator": {
        "type": "id",
        "value": "title"
      },
      "text": "${Titel}"
    }
    ...
  ],
  "data": {
    "configs": {
      "csv": {
        "path": "createNews_parametrisiert.csv"
      }
    },
    "source": "csv"
  },
  "inputs": [
    [
      "Titel",
      "string"
    ],
    ...
  ],
  "timeoutSeconds": 60
}
```

Arbeiten mit Testsuites

Eine hilfreiche Funktion von Selenium Builder ist das Zusammenstellen und Abspielen von Skriptfolgen, so genannten *Suiten*. In unserem zweiten Beispiel soll sich der Anwender am Portal anmelden, eine Mitteilung anlegen und schließlich vom Portal abmelden. Hierfür werden drei Testskripte, wie in Listing 2 dargestellt, angelegt:

- *login.json*
- *createNews.json*
- *logout.json*

Es wäre umständlich, jedes Skript separat zu laden und auszuführen. Daher werden die drei Dateien in einer Suite zusammengefasst. Dazu wird das erste Skript geladen und im Builder das Menü SUITE geöffnet. Über den Eintrag ADD SCRIPT FROM FILE können weitere Skripte zur Suite hinzugefügt werden. Da alle drei Skripte zu einem Testfall gehören, ist es ratsam, die Option „Share

© Software & Support Media GmbH

Zur Nutzung von Selenium Builder ist keine eigenständige Softwareinstallation notwendig, sondern es muss lediglich ein Firefox-Add-on installiert werden.

state across suite“ im Menü Run zu aktivieren. Damit stehen verwendete Variablen und Parameter in nachfolgenden Skripten derselben Suite zur Verfügung.

Suiten werden ebenfalls im JSON-Format gespeichert und können in beliebigen Texteditoren bearbeitet werden. Eine Datei enthält neben wenigen grundlegenden Parametern für die Suite alle Pfade zu den zusammengefassten Skripten. Die Pfade werden immer relativ angegeben, was die Verwendung von UNC-Pfaden bzw. Netzlaufwerken erschwert. Trotzdem ist es empfehlenswert, allgemeine Skripte wie z. B. *login.json* und *logout.json* an einer zentralen Stelle abzulegen. Auf diese Weise können diese Skripte in anderen Suiten wiederverwendet werden, und der Aufwand für die Wartung ist minimal.

Reporting

Beim Abspielen eines Skripts wird die Hintergrundfarbe der einzelnen Schritte gewechselt. Der gerade im Prozess befindliche Schritt wird gelb dargestellt. Erfolgreich durchlaufene Schritte werden grün angezeigt. Eine rosa Hintergrundfarbe bekommen Schritte, bei denen das Verify-Kommando nicht erfolgreich war. Das Skript wird in solchen Fällen weiter durchlaufen. Einen roten Hintergrund haben dagegen Schritte mit Assert-Kommandos, deren Bedingungen nicht erfüllt wurden (**Abb. 2**). Selenium Builder beendet in diesen Fällen die Bearbeitung des Skripts bzw. bricht den aktuellen Durchlauf einer Schleife ab und beginnt mit dem nächsten. In allen Fällen wird die Fehlermeldung im betroffenen Schritt eingeblendet. Außerdem besteht die Möglichkeit, Screenshots der geöffneten Webseite zu speichern, die beim Reporting ebenfalls hilfreich sind. Beim Kommando *saveScreenshot* kann ein beliebiger Pfad zum Speichern des Bilds angegeben werden, auch UNC-Pfade sind möglich.

Selenium Builder ist als Add-on von Firefox konzipiert. Das bedeutet, es wird davon ausgegangen, dass der Anwender das Skript oder die Suite über das Add-on startet und direkt visuelles Feedback erhält. Ein umfangreiches Reporting ist deswegen nicht implementiert. Im zweiten Teil dieses Artikels, in dem es unter anderem um Automatisierung gehen wird, werden auch die Reportingmöglichkeiten ausführlicher beschrieben.

Testen komplexer Weboberflächen

In Selenium Builder sind die Kommandos in Gruppen untergliedert. In der Gruppe „Misc“ gibt es Kommandos, die für komplexe Weboberflächen nützlich sind. Das Kommando *deleteCookie* wurde beispielsweise im Skript *logout.json* verwendet, um nach dem Abmelden aus dem Portal den Session-Cookie zu entfernen. Ein anderes wertvolles Kommando ist *switchToFrame*, mit dem man zu einem iFrame wechseln kann. Mit dem Kommando *switchToDefaultContent* wechselt man zurück zum ursprünglichen Inhalt.

Bei komplexen Weboberflächen ist es hilfreich, den Funktionsumfang des Add-ons mittels JavaScript erweitern zu können. Dafür bietet sich das Kommando *storeEval* an. Es besteht aus den zwei Attributen *script* und *variable*. Im ersten Attribut gibt man den JavaScript-Code an, der einen Rückgabewert hat. Wichtig dabei ist das Schlüsselwort *return*, das den Rückgabewert von *script* an *variable* definiert:

```
{
  "type": "storeEval",
  "script": "var now = new Date();var year = now.getFullYear();return year;",
  "variable": "jahr"
}
```

In den nachfolgenden Schritten des JSON-Skripts oder auch der Suite steht der Rückgabewert von *year* als Variable *jahr* zur Verfügung, wobei die Selenium-Variable in geschweiften Klammern und mit führendem *$*-Zeichen angegeben wird. Diese Variablen können auch innerhalb von JavaScript-Code verwendet werden:

```
{
  "type": "setElementText",
  "locator": {
    "type": "id",
    "value": "jahr"
  },
  "text": "${jahr}"
}
```

Ein weiteres Element komplexer Weboberflächen sind WYSIWYG-HTML-Editoren wie der CKEditor. Um mit Selenium Builder einen Text in den Editor zu schreiben, bietet sich wiederum das *storeEval*-Kommando an. Da kein Rückgabewert erwartet wird, kann das Feld für *variable* leer bleiben:

```
{
  "type": "storeEval",
  "script": "CKEDITOR.instances['text'].setData( '<p>Written by Selenium
                                                    Builder.</p>' );",
  "variable": ""
}
```


© Software & Support Media GmbH

	A	B	C
1	Titel	Nachricht	Bereich
2	Tourenausgaben Vorlage	Lorem ipsum dolor sit amet cc	Marketing
3	Mit neuen Lösungen für Spedition und Logistik auf der transfairlog	Lorem ipsum dolor sit amet cc	Marketing
4	Deutscher Logistik-Kongress: Systemhäuser zeigen Logistiksoftware der Zukunft	Lorem ipsum dolor sit amet cc	Vertrieb
5	Best4log: Integrierte XYZ-Lösung für Logistikdienstleister	Lorem ipsum dolor sit amet cc	Aussendienst
6	Popel setzt auf XYZ	Lorem ipsum dolor sit amet cc	Marketing

Abb. 3: CSV-Datei für Iteration zum Anlegen mehrerer Nachrichten

Anlegen von Testdaten

Die Möglichkeit der schnellen Aufnahme von Skripten ist auch für das schlichte Anlegen von Testdaten bzw. die unkomplizierte Migration von Daten nützlich. Selenium Builder bietet hierfür das Menü Data, in dem sowohl die Eingaben (Input) als auch die Quellen (JSON, XML, CSV) definiert werden können. In der Beispielapplikation soll die Blätterung zwischen 5 und 50 Mitteilungen je Seite getestet werden. Zu diesem Zweck wird das schon bestehende JSON-Skript zur Erzeugung neuer Mitteilungen um die Eingabefelder *Titel*, *Nachricht* und *Bereich* erweitert. Listing 3 zeigt die relevanten Stellen des erweiterten Skripts.

An den entsprechenden Stellen des Skripts wird danach einfach der statische Text durch die dazugehörigen Variablen ersetzt, die als „Input“ definiert wurden. Als „Data“ gibt man z. B. CSV an und verweist auf eine CSV-Datei mit den gleichnamigen Spalten (**Abb. 3**). Das Skript wird anschließend so oft durchlaufen, wie es Zeilen in der CSV-Datei gibt. Mit jeder Zeile der CSV-Datei wird eine neue Nachricht angelegt. Das Ergebnis ist in der Übersicht der Mitteilungen von **Abbildung 4** zu sehen.

Ein weiteres Szenario für Iterationen ist das Testen verschiedener Benutzersichten. Dabei werden die Log-in-Daten der einzelnen Benutzer und die zu prüfenden Kriterien, wie beispielsweise die Anzahl der zu sehenden Mitteilungen, in einer CSV-Datei angegeben. Diesen Anwendungsfall zeigt ebenfalls **Abbildung 4** mit den beiden Browserfenstern.

Beim Arbeiten mit Iterationen ist zu beachten, dass sich Schleifen immer auf einen kompletten Testfall beziehen. Es ist weder möglich, nur in einem Abschnitt eines Testfalls zu iterieren, noch eine vollständige Suite als Schleife zu durchlaufen.

Welche alternativen Ansätze gibt es?

Das Framework Geb [4] sorgt in Kombination mit Spock [5] aktuell für Aufsehen in der Java-Welt. Die beiden auf Groovy basierenden Frameworks verfolgen ein völlig anderes Vorgehen. Geb setzt ebenfalls auf das WebDriver-API, womit einfaches Testing mit unterschiedlichen Browsern ermöglicht wird. Allerdings werden die Tests nicht mit dem Browser aufgezeichnet, sondern programmatisch über Domain-specific Languages (DSL). Die DSLs sind leicht verständlich, jedoch ist Programmiererfahrung unumgänglich. Tabelle 1 stellt die Funktionsweise der beiden Werkzeuge gegenüber.

Beschreibung	Selenium Builder	Geb mit Spock
Aufzeichnungen und Selektionen	Aufzeichnung erfolgt direkt über den Browser	Aufzeichnen erfolgt programmatisch mithilfe leicht verständlicher DSLs und einer jQuery-ähnlichen Syntax
Verifikation	Auswahl der zu validierenden Elemente über den Browser und bei Bedarf anschließend die Bedingung im Testschritt entsprechend anpassen	Programmtisch über Groovy, bei zusätzlicher Verwendung von Spock
Navigation	Durch direktes Klicken im Browser wird das aufgetretene Event gespeichert	Programmatisch über Groovy DSL von Geb
Reporting	Direktes Feedback im Steuerungsfenster durch farbliche Hervorhebung des fehlgeschlagenen Testschritts	Detaillierte Fehlermeldungen mit Anzeige von Differenzen sowie Screenshots bei Fehlern
Wartung und Erweiterung von Testfällen	Tests in JSON-Formaten können erneut geladen und weiterverarbeitet werden; bei der Verwendung von Suites ist jedoch das neue Aufzeichnen der geänderten Teilbereiche oftmals schneller wie die Bearbeitung eines bestehenden Tests	Erweiterungen erfolgen innerhalb des bestehenden Codes und sind bei sauberer Codestrukturierung problemlos möglich
Integration mit anderen Testtypen	Export in verschiedene Formate und damit Integration in eine „Gesamt“-Testsuite möglich	Direkte Integration mit Unit und Integrationstests
Automatisierung	Plug-ins für die Integration mit Jenkins/Hudson sind vorhanden	Integration in CI-Umgebung ist mithilfe von Build-Tools wie Gradle oder durch direkte Integration in das verwendete Framework wie bspw. Grails sehr einfach

Tabelle 1: Gegenüberstellung der Funktionsweise von Selenium Builder und Geb

© Software & Support Media GmbH

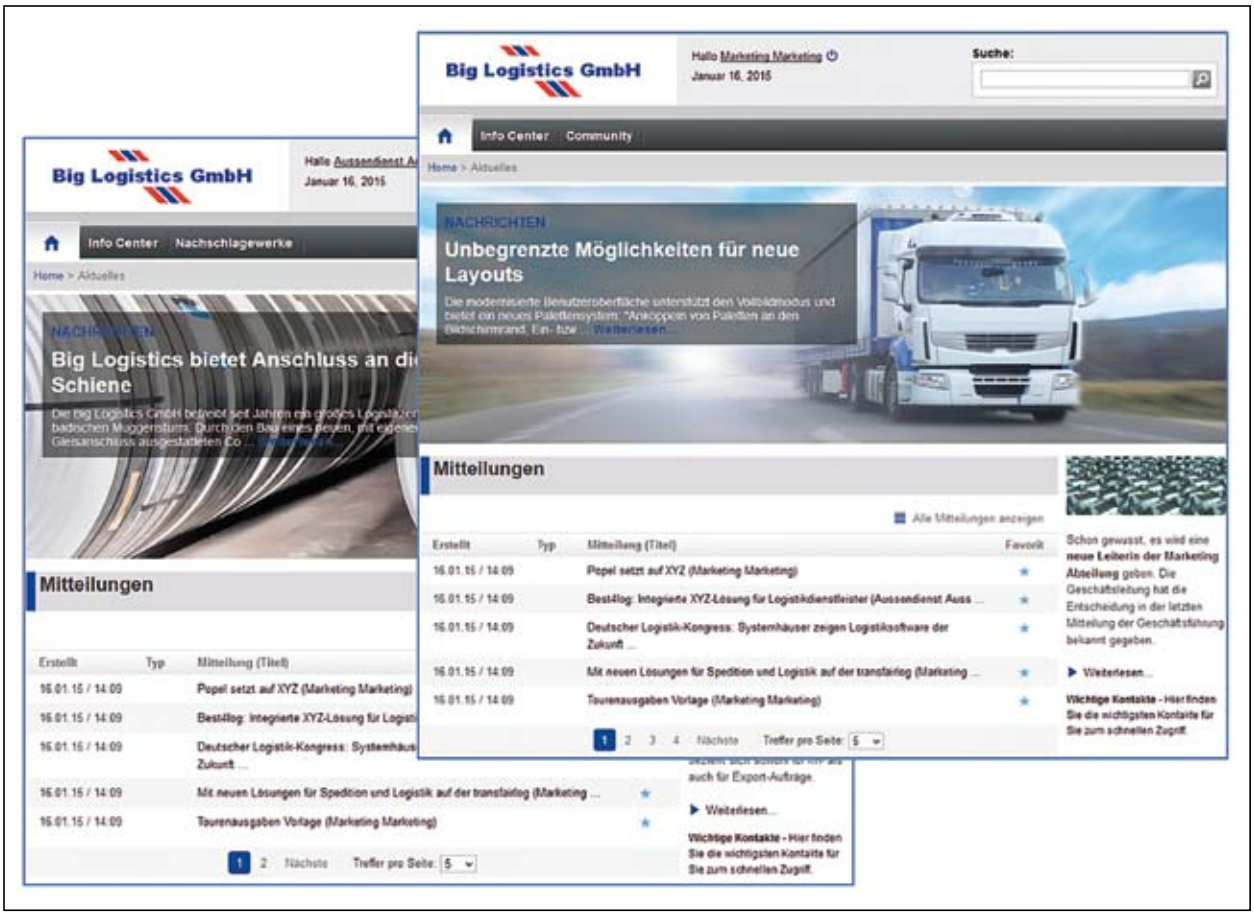


Abb. 4: Verschiedene Sichten im Portal, abhängig vom angemeldeten Benutzer

Geb steht wie Selenium Builder und auch Selenium als Open-Source-Projekt unter der Apache-2.0-Lizenz frei zur Verfügung. Welche der beiden Vorgehensarten für Tests geeigneter ist, hängt von Kriterien wie der Komplexität der Anwendung, Änderungshäufigkeiten des Systems oder den Erfahrungen innerhalb des Teams ab.

Fazit

Durch das direkte Aufzeichnen über Mausklicks und Eingaben innerhalb eines Browsers lassen sich mit Selenium Builder sehr schnell funktionale Tests erstellen. Beim Aufzeichnen sollte man trotzdem Disziplin walten lassen und die Testfälle bei Bedarf nachbearbeiten. Insbesondere die Nutzung von Testsuites ist empfehlenswert, um eine Gliederung in separate Module zu ermöglichen. Die Auswirkungen von Änderungen innerhalb der Applikation können dadurch einfacher auf bestehende Tests übertragen werden, was zu einer besseren Wartbarkeit führt.

Bisher wurde davon ausgegangen, dass die Tests manuell vom Anwender angestoßen und ausgewertet werden. Für den Test mit verschiedenen Browsern sowie einer vollständigen Automatisierung bedarf es jedoch mehr. Diese Aspekte werden im nächsten Teil der Serie beleuchtet.

Roland Rickborn ist IT-Berater bei der exensio GmbH in Karlsruhe. Als Consultant arbeitet er in Kundenprojekten im Bereich fachliche Konzeption, Projektleitung und automatisierte Tests. Aktuell betreut er die Weiterentwicklung und Optimierung von Intranetportalen bei einem großen Kunden aus der Pharmabranche.

roland.rickborn@exensio.de **@RolandRickborn**

Tobias Kraft beschäftigt sich bei der exensio GmbH mit der Architektur und Umsetzung von Enterprise-Portalen, Webapplikationen und Suchtechnologien basierend auf dem Java-Stack sowie dem Grails-Framework. Des Weiteren ist er Mitorganisator des Search Meetup Karlsruhe.

tobias.kraft@exensio.de **@tokraft**

Links & Literatur

[1] Selenium Builder: http://sebuilder.github.io/se-builder

[2] Selenium Grid: https://code.google.com/p/selenium/wiki/Grid2

[3] FirePath: https://addons.mozilla.org/de/firefox/addon/firepath

[4] „Mr. Spock ruft Geb“, in Java Magazin 1.2015

[5] „Mit Mr. Spock beim Testeinsatz“, in Java Magazin 12.2014

© Software & Support Media GmbH