Websockets mit Grails

Ahmed Salame, Ida Vanessa Gouleu Mokam
Hochschule Mannheim
Fakultät für Informatik
Paul-Wittsack-Str. 10, 68163 Mannheim

***Zusammenfassung*—Dieses Paper soll aufzeigen, wie Websockets mit Grails realisiert werden kann. Zunächst wird auf einige Grundlagen näher eingegangen, um dem Leser ein Besseres Verständnis für das Thema zu vermitteln.**

–TODO–

Inhaltsverzeichnis

1. Einleitung

WebSockets 5.1 sind eine langlebige, interaktive, Zwei-Wege-Kanal Kommunikation zwischen einem Client-Browser und End-Server, die kontinuierliche Kommunikation ohne Abfragen ermöglichen. WebSockets sind schon seit einigen Jahren da, seit sie von RFC 6455 vorgeschlagen wurden und so ziemlich alle großen Browser haben seitdem Unterstützung gewonnen.

Spring 5.4, das Java-MVC-Framework, erhielt die integrierte WebSocket-Unterstützung in Version 4.0, und diese Unterstützung wurde seitdem in Spring Boot und Grails integriert (Grails-Unterstützung wird von diesem hilfreichen Plugin bereitgestellt). Dieser Artikel gibt einen Überblick über einige Beispiele, um sich mit WebSockets vertraut zu machen und gleichzeitig etwas über dessen Integration in Grails zu lernen, als Grails Version 3.0 erreichte.

2. Forschungsmethode

Beim Recherchieren wurde unter anderem Google Books[1] verwendet, um passende Literatur zu finden. Des weiteren wurde die Homepage The Groovy Project herangezogen, um die Sprache Groovy näher zu beleuchten. Zudem wurde nach passenden Quellen aus dem Internet gesucht, um bestimmte Themen näher zu erläutern. Zudem sind diese auch für jeden zugänglich.

1. https://books.google.de/

3. State of the art

Wenn Clients den Server regelmäßig in einem bestimmten Zeitfenster abfragen, nur um zu fragen, ob aktualisierte Informationen verfügbar sind, kann es zu Problemen kommen. Wenn sich viele Clients gleichzeitig mit dem Server verbinden, kann der Thread-Pool des Servers erschöpft sein, wodurch die Anwendung angehalten wird.

Es gibt zwar nicht blockierende Server auf dem Markt, aber oft können Sie Ihr Serverprodukt nicht einfach wechseln oder die gesamte Anwendungsarchitektur in eine Nachrichten gesteuerte Architektur ändern.

Eine mögliche architektonische Lösung war daher das Konzept von WebSockets. Die HTML5 Websocket-Spezifikation definiert eine bidirektionale Duplexkommunikation zwischen Client und Server über eine einzige Verbindung. Zwei-Wege-Kommunikation bedeutet, dass ein Server mit WebSockets eine Nachricht an seine Clients senden kann. Folglich kann ein gegebener Thread-Server mehr Clients bedienen oder mehr gleichzeitige Verbindungen unterstützen.

WebSocket ist keine Socket-Technik, wie man es von einem Unix-System kennt, WebSocket ist ein Protokoll. Der WebSocket startet als gewöhnliche HTTP-Verbindung (http: //), fragt dann aber den Server, ob er zum WebSocket-Protokoll (ws: //) wechseln möchte. Wenn diese Anforderung erteilt wird, wird die HTTP-Verbindung unterbrochen und durch eine WebSocket-Verbindung über dieselbe zugrunde liegende TCP / IP-Verbindung ersetzt. Der WebSocket verwendet Port 80 oder 443. Nun können die Kommunikationspakete zwischen Client und Server hin- und hergeschickt werden. Sie reisen sogar durch Proxies und Firewalls, was die Anwendungsentwicklung erheblich erleichtert.

4. Ein Beispiel von WebSockets mit Grails

Dieser Demonstrator sendet die CPU-Last des Servers an den Webbrowser eines Clients. Auf der Serverseite existiert ein Dienst SimpleMessagingService, der die CPU-Last alle 5 Sekunden misst. Das Ergebnis wird in eine Themenwarteschlange ”topic / cpuLoad”gestellt. Über das WebSocket-Protokoll wird der Inhalt der Warteschlange an die Browser aller Clients übermittelt, die diese Themenwarteschlange abonniert haben. Alle abonnierten Browser werden alle gleichzeitig aktualisiert. Abbildung 1 zeigt dieses Beispiel.

Ein sehr nettes WebSocket-Plugin existiert für Grails. Es erleichtert die Integration der Spring 4.0 WebSocket-Unterstützung. Fügen Sie einfach compile ”: spring-websocket: 1.3.0ßum Plugins-Bereich von Grails BuildConfig.groovy hinzu. Mit Spring 4.0 WebSocket gibt es verschiedene Möglichkeiten, Nachrichten zu senden oder zu empfangen. Hier werde ich das SimpMessagingTemplate und seine convertAndSend-Funktion verwenden.

```
package com.jolorenz

import org.springframework.messaging.simp.
   SimpMessagingTemplate
import java.lang.management.ManagementFactory;

import com.sun.management.OperatingSystemMXBean;
import grails.transaction.Transactional
import groovy.json.JsonBuilder

@Transactional
class SimpleMessageService {

  SimpMessagingTemplate brokerMessagingTemplate
  OperatingSystemMXBean bean = (com.sun.
     management.OperatingSystemMXBean)
  ManagementFactory.getOperatingSystemMXBean()

  void systemCpuLoad() {

    def cpuLoad = bean.getSystemCpuLoad()
    cpuLoad = (cpuLoad*100).round(2)

    def builder = new JsonBuilder()
    builder {
      message("CPU system load: " + cpuLoad + "
         %")
      timestamp(new Date())
    }
    def msg = builder.toString()
    brokerMessagingTemplate.convertAndSend "/
       topic/cpuLoad", msg
  }
}
```

Listing 1. SimpMessagingTemplate.java

Als nächstes benötigen wir eine HTML / Javascript-Seite mit der entsprechenden Themenwarteschlange, z. ”/ Topic / cpuLoad”.

2. https://jolorenz.wordpress.com/2015/07/21/websockets-with-grails/

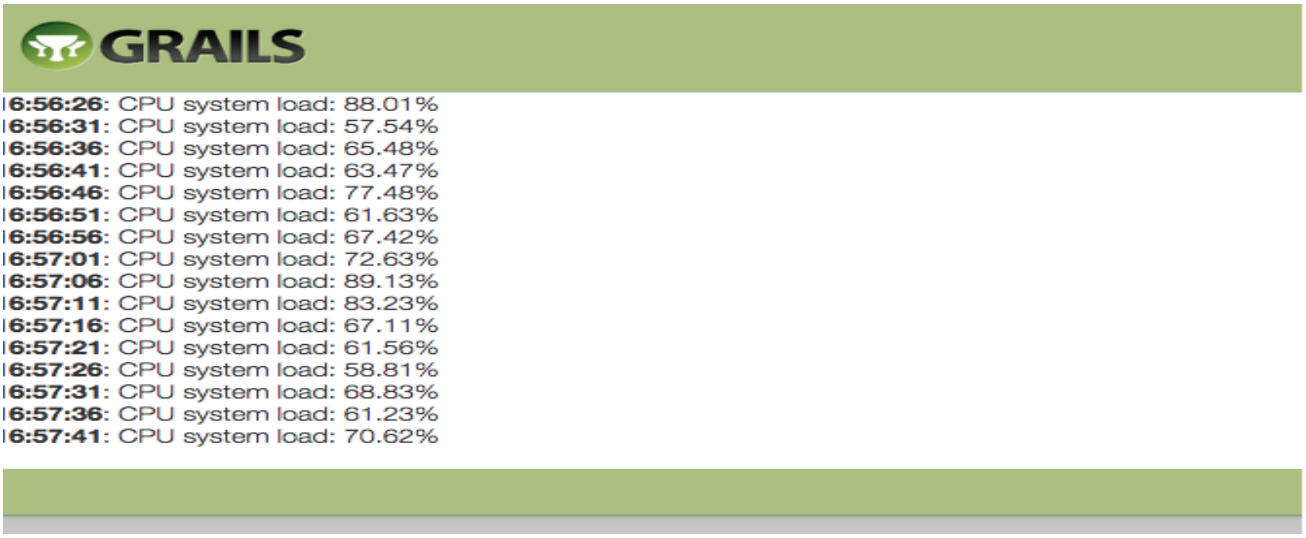

Abbildung 1. Ausgabe der CPU Auslastung mit Grails[2]

```
&lt;!DOCTYPE html&gt;;
&lt;html&gt;;
  &lt;head&gt;;
  &lt;meta name="layout" content="main" /&gt;
  &lt;asset:stylesheet src="application.css" /&
      gt;
  &lt;asset:javascript src="application.js" /&gt
      ;
  &lt;asset:javascript src="jquery" /&gt;
  &lt;asset:javascript src="spring-websocket" /&
      gt;

  &lt;script type="text/javascript"&gt;
    $(function() {
      //Create a new SockJS socket - this is
          what connects to the server
      //(preferably using WebSockets).
      var socket = new SockJS("${createLink(uri
          : '/stomp')}");

      //Build a Stomp client to send messages
          over the socket we built.
      var client = Stomp.over(socket);

      //Have SockJS connect to the server.
      client.connect({}, function() {
        client.subscribe("/topic/cpuLoad",
            function(message) {
          var msg = JSON.parse(message.body)
          var time = '&lt;strong&gt;:' + new
              Date(msg.timestamp).
              toLocaleTimeString() + '&lt;/
              strong&gt;'
          $("#cpuLoadDiv").append(time + ': ' +
               msg.message + "&lt;br/&gt;");
        });
      });
    });
  &lt;/script&gt;
&lt;/head&gt;
```

```
&lt;body&gt;
  &lt;div id="cpuLoadDiv"&gt;&lt;/div&gt;
&lt;/body&gt;
&lt;/html&gt;
```

Listing 2. Beispiel einer GSP Datei

Das ist alles was benötigt wird, um eine WebSocket-Kommunikation zwischen Server und Client einzurichten. Es ist ein einfaches Beispiel und es fehlen Funktionen wie Filter für Benutzergruppen oder Sicherheit, aber es gibt eine Idee, sich mit WebSockets vertraut zu machen. Was sind die Nachteile?

WebSocket ist eine Browserfunktion und nicht alle Browser unterstützen diese Funktion. Wahrscheinlich werden in naher Zukunft alle Browser und Server WebSockets unterstützen. CanIUse bietet einen guten Überblick über die unterstützten Webbrowser.

5. Grundlagen

In diesem Kapitel sollen einige Grundlagen erklärt werden. Zunächst wird erklärt, was genau Websockets sind. Dann wird auf die Technologie Groovy on Grails näher eingegangen. Anschließend wird das Spring Framework beleuchtet, bevor es dann eine Erläuterung bezüglich SockJS und Stomp gibt.

5.1. Websockets

Websockets erlauben eine Full-Duplex single-socket Verbindung. Zwischen dieser Verbindung können Nachrichten zwischen einem Server und einem Client ausgetauscht werden.

Ursprünglich musste z. B. ein Client bei einem Server polling betreiben, das heißt immer wieder bei einem Server nachfragen, ob relevante Informationen für den Client vorhanden sind. Ein Websocket stellt eine Bidirektionale Verbindung her, welches das polling nicht mehr notwendig macht [1, Seite 751f]. Daten können zwischen dem Client und dem Server gleichzeitig hin und her geschickt werden. Eine Websocket Verbindung zwischen Client und Server ist persistent [8].

5.2. Groovy

Groovy ist eine Programmiersprache unter der Apache-Lizenz [6]. Es wurde für die Java Plattform entwickelt, um die Entwicklung von Anwendungen zu beschleunigen. Zudem war eines der Ziele, dass die Syntax vertraut ist und einfach zu lernen. Zudem sit es mit anderen Java Programmen Kompatibel.

Grooy ist sowohl eine kompilierte, als auch eine Interpretierte Programmiersprache. Listing 3 zeigt ein einfaches Beispiel eines Groovy Programms. Es wird eine einfache Klasse angelegt mit verschiedene Attribute. Dabei ist zu sehen, dass nicht zwingend der Konkrete Datentyp angegeben werden muss. Es besteht die Möglichkeit den Datentyp mit *def* anzugeben, so dass der Datentyp des Attributes automatisch ermittelt wird, je nachdem was das Attribute für einen Wert zugewiesen bekommt.

In der Groovy Dokumentation [7] ist die Sprache Groovy beschrieben.

```
class Person{

  def vorname = 'Max'
  String name = 'Mustermann'
  int alter = 42

  def printName(){

    println name
  }
}

def person = new Person()

person.printName()
```

Listing 3. Eine einfache Groovy Klasse

5.3. Grails

Grails ist eine Groovy basierte Webtechnologie, welche auf die Java Virtual Machine (JVM) läuft. Grails setzt dabei auf Spring auf [2]. Grails erlaubt das erstellen von Web Anwendungen innerhalb kürzester Zeit. Zudem unterstützt Grails Scaffolding, was das Generieren von CRUD Pages ermöglicht. Grails basiert zudem auf das *convention over configuration* Prinzip.

Grails nutzt das JavaEE als Grundlage für die Architektur und Spring für die Strukturierung der Anwendung via *dependency injection*. Zudem nutzt Grails Gradle [10]. Abbildung 2 gibt einen Einblick in Grails Architektur.

5.4. Spring

Bei Spring handelt es sich um ein plattformunabhängiges Framework, welches unter der Apache-Lizenz von SpringSource[4] veröffentlicht wurde. Da Spring selber in Java geschrieben ist, ist es zu Groovy kompatible. Mithilfe von Spring soll das entwickeln von Java/J2EE Anwendungen vereinfacht werden, zudem soll die Entwicklung auch Produktiver vonstatten gehen [3, Seite 1].

Spring ermöglicht es, dass Anwendungen für viele verschiedene Zwecke geschrieben werden. So ist es möglich Anwendungen sowohl für Applets zu Entwickeln, als auch für standalone Clients. Das half Spring dabei sich gegenüber der Konkurrenz zu behaupten. Spring ist eines der meist genutzten leichtgewichtigen Frameworks. Einige Merkmale von Spring sind [3, Seite 5]:

- Inversion of Control:
  Der core Container von Spring bietet

3. https://www.dunebook.com/wp-content/uploads/2017/02/Grails-Architecture1.jpg
4. **https://spring.io/**.

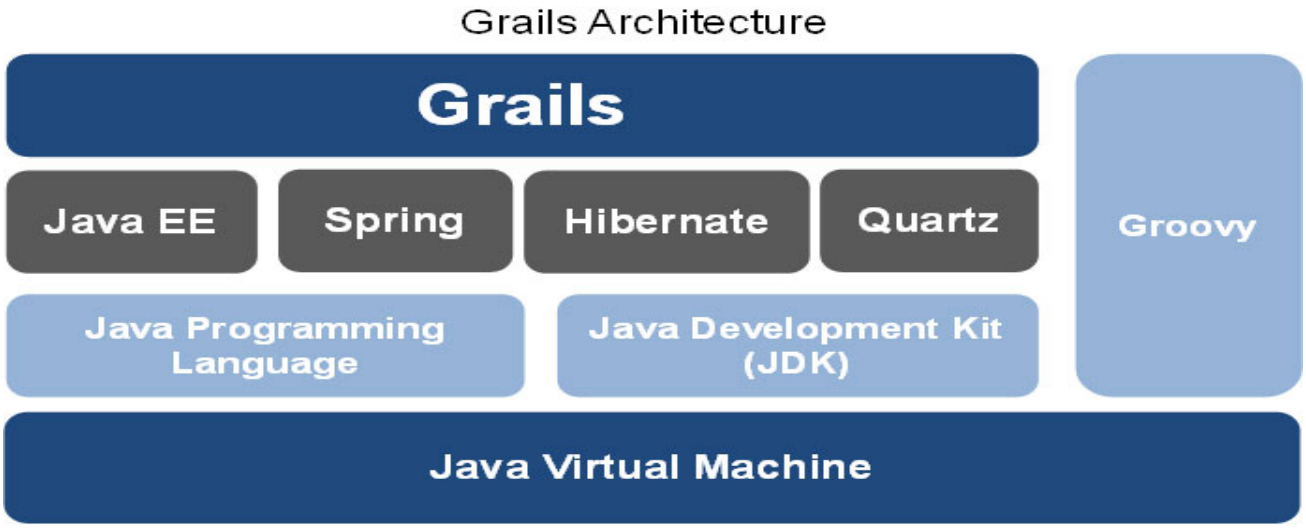

Abbildung 2. Grails Architektur [3]

Konfigurationsmanagement für Plain Old Java Object (POJO) Dateien. POJOs sind einfache Java Objekte, die möglichst wenig externe Abhängigkeiten haben.

- Aspektorientierte Programmierung:
  Spring bietet für die Aspektorientierte Programmierung wichtige out-of-the-box Service an, wie z. B. deklaratives Transaktionsmanagement.

- Transaktionsmanagement:
  Spring bietet eine Abstraktion von einer Transaktion, welches auf Java Transaction API (JTA) aufsetzt.

- MVC Web Framework:
  Spring bietet ein flexibles Request basiertes MVC Web Framework.

- Leichtgewichtiger Fernzugriff:
  Spring bietet Unterstützung für POJO basierte Fernzugriffe mithilfe von Protokollen, wie RMI, IIOP, Hessian oder anderen Webservice Protokolle.

Mit Version 4.0 unterstützt Spring auch die Anforderung JSR-356 [1, Seite 14]. JSR-356 ist die Java Spezifikation für Websockets[5]. Websockets werden in 5.1 näher erläutert.

5. https://www.jcp.org/en/jsr/all

5.5. SockJS

SockJS ist eine Browser JavaScript Bibliothek. Es bietet ein Websocketartiges Objekt an. Die JavaScript Bibliothek bietet eine Kommunikationschannel zwischen Browser und Webserver an, welches wenig Latenzzeit hat [5].

5.6. Stomp

Bei Stomp handelt es sich um ein einfaches Textorientiertes Nachrichten Protokoll. Jeder Stomp Client kann mit einem Stomp Nachrichten Broker Kommunizieren. So können unterschiedliche Systeme miteinander Kommunizieren [4].

6. Websocket mit Grails

In diesem Kapitel gibt es ein Überblick über die Implementierung eines Websockets mit Grails. Zunächst werden die relevanten Annotationen eingeführt. Anschließend wird erklärt, wie der HTML Code des Clients aussehen kann. Darauf aufbauend wird dann der Quellcode des Servers erklärt.

–TODO–

Um in Grails über Websockets zu kommunizieren, muss folgende Zeile in build.gradle unter den *Depenencies* eingefügt

werden[6].
compile 'org.grails.plugins:grails-spring-websocket:2.3.0'

6.1. Annotationen

Da bei der Websocket Programmierung in Grails, die Bibliothek Spring 5.4 genutzt wird, wird auch von dessen Annotationen starken Gebrauch gemacht. Folgend werden die wichtigsten Annotationen, die für die Websocket Programmierung mit Grails benötigt werden, aufgelistet und erklärt [9]

- @Controller:
  Diese Annotation zeichnet eine Klasse als Controller aus. In Grails ist diese Angabe Optional.
- @MessageMapping:
  Diese Annotation ruft die annotierte Methode auf, wenn eine Nachricht zur angegebenen Stelle versendet wird. In Listing 4 wird die Methode *sendMessage* dann aufgerufen, wenn eine Nachricht an */destination* gesendet wird.
- @SendTo:
  Diese Annotation sorgt dafür, dass alle Clients, welche die angegebene Stelle abonniert haben, den Wert, denn die Methode *getGreeting* zurück gibt, auch erhalten. In Listing 5 werden alle Clients die Nachricht *Hallo user* erhalten, wenn sie */topic/destination* abonniert haben.

```
import org.springframework.messaging.handler.
    annotation.MessageMapping

class MessageMappingExampleController {

    @MessageMapping("/destination")
    protected String sendMessage(String message){

      return message;
    }
}
```

Listing 4. Beispiel von @MessageMapping

6. https://plugins.grails.org/plugin/zyro/grails-spring-websocket

```
import org.springframework.messaging.handler.
    annotation.MessageMapping
import org.springframework.messaging.handler.
    annotation.SendTo

class SendToExampleController {

  @MessageMapping("/destination")
  @SendTo("/topic/destination")
   protected String getGreeting(String message){

     return "Hallo user";
    }
}
```

Listing 5. Beispiel von @SendTo

```
import org.springframework.stereotype.Controller

@Controller
class ExampleController { }
```

Listing 6. Beispiel von @Controller

6.2. Quellcode des Clients

Damit der Client über den Websocket mit dem Server Kommunizieren kann, benötigt dieser ein Objekt vom Typ SockJS, dass als Socket dient, über dass Kommuniziert wird. Zudem ist auch das Protokoll Stomp notwendig. In Listing 7 ist der Code eines Groovy Server Pages (GSP) Dokuments zu sehen. Hier wird mithilfe der Grails Methode createLink[7] eine Uniform Resource Identifier (URI) generiert, auf. Mithilfe des Sockets wird ein Stomp Objekt generiert. Anschließend wird mit der Methode connect[8] der Client einen Asynchronen Aufruf tätigen. In dieser wird festgelegt, was der Client abonniert, in diesem Fall */topic/hello*. Anschließend wird das verhalten definiert, was beim drücken eines Buttons passieren soll. Die dargestellte Seite stellt dem Nutzer nur einen Button zur Verfügung, welche beim drauf klicken eine Nachricht in eine Div-box einfügt.

```
<!DOCTYPE html>
<html>
<head>
    <meta name="layout" content="main"/>

    <asset:javascript src="application"/>
```

7. http://docs.grails.org/3.2.3/ref/Tags/createLink.html
8. https://stomp-js.github.io/stomp-websocket/codo/class/Client.html# connect-dynamic

```
    <asset:javascript src="spring-websocket"/>

    <script type="text/javascript">
        $(function () {

            var socket = new SockJS("${createLink(
                uri: '/stomp')}");
            var client = Stomp.over(socket);

            client.connect({}, function () {
                client.subscribe("/topic/hello",
                    function (message) {
                    $("#textbox").append(message.body
                        );
                });
            });

            $("#examplebutton").click(function () {
                client.send("/app/hello", {}, "
                    Beispiel Nachricht ");
            });
        });
    </script>
</head>

<body>
<button id="examplebutton">Button</button>

<div id="textbox"></div>
</body>
</html>
```

Listing 7. GSP Code des Clients

6.3. Quellcode des Servers

Der Quellcode auf der Server Seite ist sehr simple. Für ein einfach Beispiel reichen die Annotationen aus dem Kapitel 6.1. In diesem Beispiel Quellcode wird die Methode *Message* dann aufgerufen, wenn eine Nachricht an */hello* erfolgt. Anschließend wird die Methode ausgeführt, bis sie dann alle Abonnenten den Wert übermittelt, denn diese Methode liefert. In diesem Quellode wird mit dem drücken des Buttons aus 6.2 dafür gesorgt, dass eine Nachricht sowohl versendet, als auch empfangen wird.

```
import org.springframework.messaging.handler.
    annotation.MessageMapping
import org.springframework.messaging.handler.
    annotation.SendTo

class ExampleController {

  @MessageMapping("/hello")
  @SendTo("/topic/hello")
  protected String Message(String message) {
    return "${message}"
  }
}
```

Listing 8. Beispiel der Controller Klasse

7. Fazit

Listings

Abbildungsverzeichnis

Abkürzungen

POJO Plain Old Java Object
GSP Groovy Server Pages
JTA Java Transaction API
JVM Java Virtual Machine
URI Uniform Resource Identifier

Literatur

[1] Iuliana Cosmina u.a. *Pro Spring 5*. Apress, 2017, S. 849. ISBN: 978-1-4842-2808-1.

[2] *Grails*. URL: https://grails.org/ (besucht am 29. 12. 2017).

[3] Rod Johnson u.a. *Professional Java Development with the Spring Framework*. Birmingham, UK, UK: Wrox Press Ltd., 2005. ISBN: 0764574833, 9780764574832.

[4] Jeff Mesnil. *STOMP Over WebSocket*. 2012. URL: http : / / jmesnil . net / stomp - websocket / doc/ (besucht am 29. 12. 2017).

[5] SockJS. *SockJS*. 2017. URL: https://github.com/sockjs (besucht am 29. 12. 2017).

[6] The Groovy Project, Apache Software Foundation. *A multi-faceted language for the Java platform*. 2017. URL: http://groovy-lang.org/ (besucht am 29. 12. 2017).

[7] The Groovy Project, Apache Software Foundation. *Documentation*. 2017. URL: http://groovy-lang.org/documentation.html (besucht am 29. 12. 2017).

[8] Malte Ubl und Eiji Kitamura. *Einführung zu WebSockets: Sockets im Web*. Oktober 2010. URL: https://www.html5rocks.com/de/tutorials/websockets/basics/ (besucht am 29. 12. 2017).

[9] *Using WebSocket to build an interactive web application*. 2017. URL: https://spring.io/guides/gs/messaging-stomp-websocket/#initial.

[10] Lars Vogel. *Grails Development - Tutorial*. 2016. URL: http://www.vogella.com/tutorials/Grails/article.html (besucht am 29. 12. 2017).