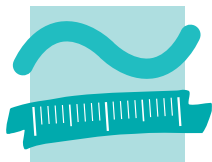


Implementierung eines JavaScript WebSocket Streamingclients



BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Name: Eugen Pirogoff

Matrikel: 772814

Bearbeitungszeitraum: 1. Juli 2013 – 1. Oktober 2013

Studiengang : Medieninformatik

Fachbereich : VI

Betreuer und Gutachter 1 : Prof. Dr. Targo Pavlista

Gutachter 2 : Prof. Dr. Felix Alexander Gers

Inhaltsverzeichnis

1	Einleitung	3
2	Fachliches Umfeld.....	5
2.1	Node.js.....	5
2.1.1	Node.js als HTTP-Server.....	7
2.1.2	NPM.....	8
2.1.3	Socket.io und WebSocket	10
2.1.4	Express.....	12
2.2	HexGL.....	13
2.3	VirtualJoystick.js	14
3	Aufgabenstellung.....	15
3.1	Steuerungssapplikation	15
3.2	Spieleapplikation HexGL.....	16
3.3	HTTP- und Backend-Server.....	16
4	Anforderungsdefinition	17
4.1	Zielbestimmungen	17
4.2	Produkteinsatz	17
4.3	Produktfunktionen	18
4.4	Produktdaten	18
4.5	Produktleistungen	19
4.6	Betriebsbedingungen	19
4.7	Benutzungsschnittstelle	19
4.8	Technische Produktumgebung.....	20
4.8.1	Software.....	20
4.8.2	Hardware	21
4.8.3	Orgware.....	22
5	Systementwurf.....	23
5.1	Steuerungssapplikation	23
5.2	Spieleapplikation	25
5.3	Serverapplikation	27
6	Realisierung	30
6.1	Entwicklungsprozess	30
6.2	HTTP-WebSocket-Server und Bibliotheken.....	32
6.3	Steuerungssapplikation.....	33
6.4	Spieleapplikation	34
7	Verbesserung der Implementierung.....	35
7.1	PeerJS.....	35
7.2	Installation	36
7.3	Implementierung.....	36
8	Installationsanleitung und Deploy	38
8.1	Virtuelle Maschine.....	38
8.2	Native Installation	41
8.3	Deploy auf Nodejitsu	42
9	Fazit.....	44
	Abbildungsverzeichnis	46
	Appendix	47
	Quellenverzeichnis	49

1 Einleitung

Durch die immer größere Verbreitung von internetfähigen Mobilgeräten wie Smartphones und Tablets ergeben sich neuartige Interaktions- und Kommunikationsmöglichkeiten zwischen Benutzern und Serviceanbietern. Der Akzeptanz und Spontanität einiger dieser Möglichkeiten sind jedoch Grenzen gesetzt, da durch die Notwendigkeit, vorab eine Applikation installieren zu müssen, eine gewisse Hemmschwelle auf Seiten der Benutzer entsteht. Die Nutzung einer nativen Applikation ist dabei nicht in allen Fällen notwendig, da die gleiche Funktionalität mit einer in einem Webbrowser lauffähigen Applikation erreicht werden kann. Durch die Bereitstellung einer solchen Applikation in Form einer in jedem modernen HTML5-Webbrowser lauffähigen Webanwendung würde die Notwendigkeit einer Installation entfallen, und somit jene Hemmschwelle gesenkt werden.

In den meisten Fällen wird ein Datenaustausch zwischen Anwendungen zur Interaktion angestrebt, der je nach Anwendungsfall, unterschiedliche Kriterien erfüllen soll. Ein solcher Datenaustausch könnte exemplarisch zur Steuerung einer Web-Anwendung, bei entsprechender Anpassung und latenzfreier Verarbeitung, verwendet werden. Durch die weiter voranschreitende Implementierung¹ von Webstandards² in modernen Webbrowsern könnte eine solche Web-Applikation zur Steuerung einer weiteren Web-Applikation, unter Nutzung von WebSockets³ realisiert werden, um den klassischen Spiele-Controller zu ersetzen.

Ziel dieser Arbeit ist die Implementierung einer Gamecontroller-Web-Applikation für Smartphones und Tablets am Beispiel der Bedienung eines webbasierten Rennspiels. Zu Demonstrationszwecken wird hier das Spiel HexGL⁴ eingesetzt, da es quelloffen auf GitHub verfügbar ist. Besonderes Augenmerk soll dabei auf eine möglichst einfache Handhabung und verzögerungsfreie Bedienung gelegt werden.

¹ Implementierungsstatus von Standards in Webbrowsern: <http://caniuse.com>

² W3C Gremium zur Standardisierung von Web-Technologien: <http://www.w3.org>

³ Weiterführende Informationen zu WebSockets: <http://www.w3.org/TR/websockets>

⁴ Weiterführende Informationen und Quellcode des Rennspiels HexGL: <http://github.com/BKcore/HexGL>

Bei der Implementierung dieses Softwaresystems steht der Verzicht auf native und damit plattformspezifische Lösungen zu Gunsten des Einsatzes von webbasierten und plattformunabhängigen Technologien im Vordergrund. Dabei zeigt diese Arbeit die Entwicklung sowie die Vor- und Nachteile einer solchen Implementierung.

2 Fachliches Umfeld

Die folgenden Abschnitte stellen eine Einleitung in fachlich relevante Technologien dar, die zur Realisierung dieser Abschlussarbeit verwendet werden.

2.1 Node.js

Die Plattform Node.js wurde im Jahre 2009 von Ryan Dahl veröffentlicht. Sie basiert auf der V8-JavaScript-Engine⁵, die von der Firma Google für den Webbrowser Chrome entwickelt wird. Node.js selbst steht unter der MIT-Lizenz⁶, ist quelloffen und wurde in den Programmiersprachen C und JavaScript geschrieben. Bei ihrer Entwicklung liegt der Fokus darauf, viele parallele Netzwerkverbindungen gleichzeitig auf eine sparsame Weise offen zu halten, um bei Bedarf proaktiv Daten an diese ausliefern zu können. Die V8-JavaScript-Engine wurde zur Realisierung von Node.js mit vielen bereits vorhandenen Bibliotheken erweitert. Besonders hervorgehoben werden sollte die Bibliothek libev⁷, die das grundlegende Architekturkonzept des Eventloop⁸ implementiert, welche den Kontrollfluss der laufenden Applikationen regelt und Aufgaben an entsprechende systemnahe Funktionen verteilt. Zur Kapselung von systemnahen Funktionen in asynchrone und nicht blockierende Funktionen wird libuv⁹ verwendet, um auf unterschiedlichen Betriebssystemen ein einheitliches Verhalten zu erzielen. Das Verwenden von externen Komponenten für Node.js hat den Vorteil, dass Leistungssteigerungen und Problembehebungen, die von Google an der V8-JavaScript-Engine vorgenommen werden, sowie Verbesserungen aus den zusätzlich in Node.js verwendeten Bibliotheken, auch in Node.js einfließen. Damit müssen eventuell vorhandene Fehler nur an einer zentralen Stelle behoben werden, um alle bibliotheksnutzenden Projekte davon profitieren zu lassen.

⁵ Projektseite der V8-JavaScript-Engine von Google: <http://code.google.com/p/v8/>

⁶ Weitere Informationen zur MIT-Softwarelizenz: <http://opensource.org/licenses/mit-license.php>

⁷ Projektseite von libev: <http://software.schmorp.de/pkg/libev.html>

⁸ Weiterführende Informationen zu Eventloop:
<http://www.fh-wedel.de/~si/seminare/ss11/Ausarbeitung/08.nodejs/konzepte.html>

⁹ Dokumentation und Quellcode von libuv: <https://github.com/joyent/libuv>

Da Node.js eine ereignisgesteuerte Architektur¹⁰ besitzt, wird die Applikation nicht durch direkten Aufruf von Methoden, sondern durch Events gesteuert. Der Eventloop nimmt Anfragen entgegen, delegiert diese an die entsprechende Funktion und steht anschließend wieder für die nächste Anfrage bereit. Sobald diese systemnahe Funktion die Bearbeitung der Anfrage abgeschlossen hat, gibt sie das Ergebnis dem Eventloop zurück und der Client erhält die entsprechende Antwort mit der anschließenden Ausführung der Callback-Funktion. Da Node.js im Kern nur einen einzigen Thread ausführt, würden CPU-intensive Aufgaben, wie das Lesen einer Datei von der Festplatte oder das Verbinden zu einer Datenbank, ohne die Verwendung des Eventloop die gesamte Applikation verlangsamen und für eine parallele Benutzung ungeeignet machen.

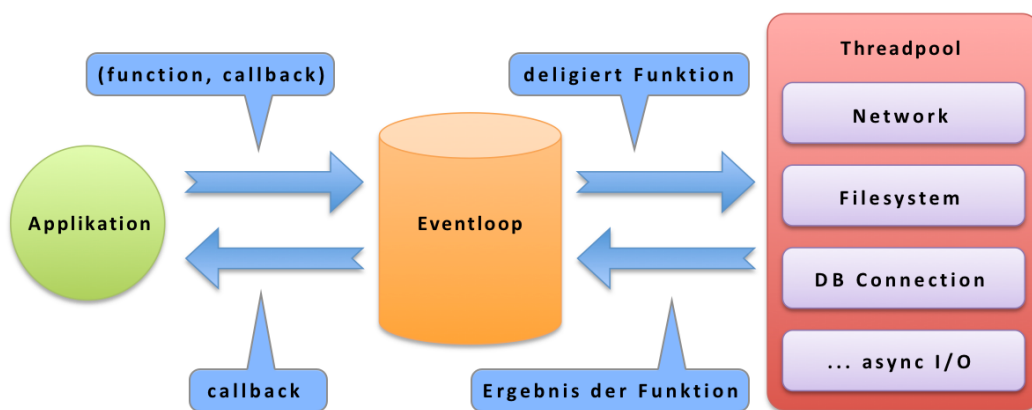


Abbildung 1: Aufruf und Kapselung von Funktionen in Node.js durch das Delegieren an systemnahe Funktionen. Quelle: Angelehnt an die Grafik von Aaron Stannard.

<http://www.aaronstannard.com/post/2011/12/14/Intro-to-NodeJS-for-NET->

Letzter Zugriff: 17. Juli 2013

Durch den Einsatz von Callback-Funktionen und der Delegation an entsprechende Funktionen wird ein asynchrones Verhalten erzielt, welches Node.js viele CPU-Zyklen spart. Eine rein prozedurale Programmierweise dagegen würde mit dem Warten auf eine Antwort der angesprochenen Ressource die Applikation blockieren, welches sich gemeinsam mit dem benötigten Rechenaufwand, und der CPU-Geschwindigkeit auf die Latenzzeiten auswirkt.

¹⁰ Informationen zu ereignisgesteuerter Architektur: http://de.wikipedia.org/wiki/Ereignisgesteuerte_Architektur

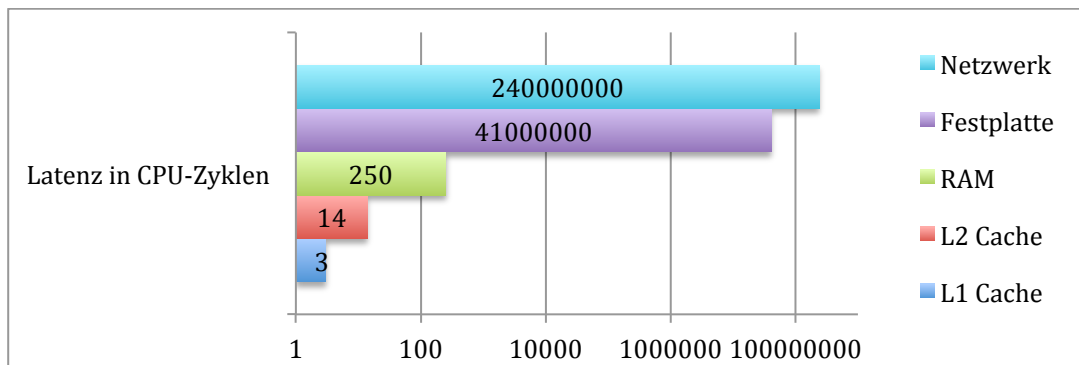


Abbildung 2: Latenzzeiten in Abhängigkeit der angesprochenen Ressource.

Quelle: Ryan Dahl „Node.js Evented I/O for V8 JavaScript“, Präsentation auf JSConfEU

8. November 2009. Aufzeichnung, , ab Minute 2: <http://html5tv.rot13.org/JSConf.eu-Node.js.html>
 letzter Zugriff: 21. Juli 2013

Eine asynchrone, non-blocking Programmierweise hat im Vergleich zu der üblichen prozeduralen Programmierweise bei Netzwerkanwendungen deutliche Vorteile und sollte besonders bei Applikationen mit vielen parallelen Anfragen genutzt werden.

2.1.1 Node.js als HTTP-Server

Um die Programmierweise von Node.js zu demonstrieren, folgt hier die Implementierung eines begrenzt nutzbaren HTTP-Servers. Dafür ist zunächst das Nachladen von zwei Kernmodulen von Node.js notwendig, um HTTP und das Filesystem verwenden zu können. Ist dies geschehen, kann nun ein HTTP-Server erstellt werden, der eine anonyme Funktion bei einer Anfrage auf den Port 8888 ausführt. In dieser Funktion findet die Verarbeitung der Anfrage statt. Da es sich um einen begrenzt nutzbaren HTTP-Server handelt, wird keine Unterscheidung zwischen den HTTP-Anfragen getroffen und immer die Datei „server.js“ zurückgegeben.

```

1 var http = require('http');
2 var fs = require('fs');
3
4 http.createServer(function (req, res) {
5   res.writeHead(200, {'Content-Type': 'text/plain'});
6   res.end(fs.readFileSync('server.js'));
7 }).listen(8888);

```

Abbildung 3: Node.js HTTP-Server, der bei allen Anfragen nur eine Datei zurückliefert.

Quelle: SublimeText2, eigenes Beispiel.

Die in Abbildung 3 dargestellten Request- und Response-Argumente in der HTTP-Server-Funktion verdeutlichen die eventbasierte Datenflusssteuerung einer Node.js-Applikation. Hierbei wird innerhalb der Callback-Funktion des HTTP-Servers eine zusätzliche Funktion gestartet, welche das Dateisystem anspricht, um die für die Antwort benötigte „server.js“ auszulesen. Die gesamte HTTP-Anfrage wird erst dann beantwortet, wenn das Auslesen der Datei vom Filesystem abgeschlossen ist. Dies unterscheidet sich stark von einer prozeduralen Vorgehensweise, in der die Datei schon vorab in einer Variablen gespeichert wird.

2.1.2 NPM

Um die Vielzahl an vorhandenen und stetig wachsenden Paketen für Node.js verwalten zu können, wird NPM¹¹ als Paketmanager genutzt. Dieser verwendet die Datei `package.json`¹² als zentrale Konfigurationsdatei. Sie besteht aus Schlüssel-Wert-Paaren in einem JSON-Datenformat¹³. Neben Projektnamen, Versionsnummern, Autoren, etc., besteht die Möglichkeit Anforderungen an die Laufzeitumgebung in Form von externen Abhängigkeiten zu definieren. Um solche Abhängigkeiten aufzulösen, genügt das Ausführen des Befehls „npm install“ im zentralen Wurzelverzeichnis, in dem auch die Konfigurationsdatei `package.json` liegt. Es hat zur Folge, dass NPM die aufgeführten Abhängigkeiten

¹¹ Projektseite vom NPM-Paketmanager: <https://github.com/isaacs/npm>

¹² Interaktive Anleitung für das Erstellen einer `package.json` mit zugehörigen Erklärungen: <http://package.json.nodejitsu.com>

¹³ Dokumentation zum JSON-Datenformat: <http://www.json.org>

im zentralen Repository¹⁴ nachschlägt, herunterlädt und im Unterverzeichnis „node_modules“ des Wurzelverzeichnis der Applikation ablegt. Jedes installierte Paket beinhaltet wiederum die eigenen Abhängigkeiten in Form eines weiteren Unterverzeichnisses mit dem Namen „node_modules“ und ist damit autark. Das gewünschte Paket kann nun innerhalb der Applikation, unter der Verwendung der Befehlszeile `var express = require('express');` (Bei Installation des Express-Frameworks), geladen und verwendet werden. Eine automatische Deduplizierung¹⁵ der eventuell mehrfach vorhandenen Module, die sich in den Abhängigkeiten der Pakete befinden könnten, findet nicht statt und muss manuell unter Verwendung der Befehlszeile `npm dedupe` angestoßen werden. Dadurch werden mehrfach vorkommende Abhängigkeiten um eine Hierarchieebene nach oben verschoben. Die Abhängigkeiten werden damit transformiert und die Hierarchie somit flacher.

Eine Versionsnummerierung innerhalb der „package.json“ findet bei allen Modulen auf eine semantische¹⁶ Weise statt. Es ist immer ein „Major.Minor.Bugfix“ oder eine Versionsrelation in Form von „>=Major.Minor.Bugfix“ von der gewünschten Bibliothek anzugeben, um eine korrekte Installation mit NPM zu erzielen.

Der Aufruf des Befehls `npm install`, unter Verwendung der in Abbildung 4 dargestellten package.json, installiert die neuesten Revisionsnummern von Redis 0.3 und Socket.io 0.1 samt ihren Abhängigkeiten mit einer Node.js Version 0.10 oder neuer. Außerdem führt dies zu einer Duplikation von Redis, da Socket.io diese als interne Abhängigkeit bereits besitzt.

¹⁴ NPM und das zentrale Verzeichnis von registrierten Modulen: <https://npmjs.org>

¹⁵ Deduplizierung von Paketen mit NPM: <https://npmjs.org/doc/cli/npm-dedupe.html>

¹⁶ Dokumentation der semantischen Versionsnummerierung: <http://semver.org>

```

1 {
2   "name": "myapplication",
3   "version": "0.1.0",
4   "author": "Eugen Pirogoff <eugenpirogoff@me.com>",
5   "description": "does awesome stuff",
6   "contributors": [
7     {
8       "name": "Eugen Pirogoff",
9       "email": "eugenpirogoff@me.com"
10    }
11  ],
12  "scripts": {
13    "start": "node server.js",
14    "predeploy": "do this stuff before deploy",
15    "postdeploy": "do that stuff after deploy"
16  },
17  "main": "server.js",
18  "repository": {
19    "type": "git",
20    "url": "https://github.com/eugenpirogoff/myapplication.git"
21  },
22  "dependencies": {
23    "socket.io": "0.1.x",
24    "redis": "0.3.x"
25  },
26  "license": "MIT",
27  "engines": {
28    "node": ">=0.10"
29  }
30 }

```

Abbildung 4: Darstellung einer `package.json` zur Verdeutlichung der Abhängigkeiten.

2.1.3 Socket.io und WebSocket

Socket.io ist eine aus server- und clientseitigen Teilen bestehende JavaScript-Bibliothek¹⁷. Sie stellt eine Abstraktion für WebSockets¹⁸ dar und bietet sowohl client- als auch serverseitig ein beinahe identisches Interface. Der Vorteil dieser Bibliothek liegt in ihrer Fähigkeit, in vielen unterschiedlichen Browsern das gleiche Interface bereitzustellen, ohne dass es dabei zu Problemen bei der Interoperabilität mit verschiedenen Webbrowsern kommt. Um dies zu erreichen, verwendet Socket.io mehrere Fallback-Möglichkeiten wie z.B. „Adobe Flash Socket“¹⁹ und „JSONP polling“, die je nach technischem Stand des Webrowsers, automatisch bei Nichtverfügbarkeit von WebSockets zur Anwendung kommen.

¹⁷ Dokumentation und Quellcode der Bibliothek : <http://socket.io>

¹⁸ WebSocket-Standard des W3C : <http://www.w3.org/TR/websockets/>

¹⁹ Informationen zu Adobe Flash Socket:
http://livedocs.adobe.com/flash/9.0_de/ActionScriptLangRefV3/flash/net/Socket.html

Der Vorteil in der Benutzung von WebSockets liegt in der Möglichkeit, nach einer vom Client anfänglich hergestellten Verbindung, der bidirektionalen und nahezu latenzfreien Kommunikation und dem äußerst geringem Overhead des Protokolls im Vergleich zu HTTP. Um eine WebSocket-Verbindung zu etablieren, muss der Client unter Verwendung des HTTP-GET-Befehls den Wechsel auf das WebSocket-Protokoll beim Server beantragen. Eine solche Anfrage des Clients an den Server zum Wechsel auf das WebSocket-Protokoll könnte wie folgt aussehen:

```
1 GET ws://echo.websocket.org/?encoding=text HTTP/1.1
2 Pragma: no-cache
3 Origin: http://www.websocket.org
4 Host: echo.websocket.org
5 Sec-WebSocket-Key: awQB089RkIC5hAAi1npV8A==
6 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_5)
7 Upgrade: websocket
8 Sec-WebSocket-Extensions: x-webkit-deflate-frame
9 Cache-Control: no-cache
10 Connection: Upgrade
11 Sec-WebSocket-Version: 13
```

Abbildung 5: Clientseitige Anfrage an den Server bei einem WebSocket-Handshake.

Quelle: Handshake mit <http://www.websocket.org/echo.html> in einem Chrome Browser.

Letzter Zugriff: 27. September 2013

Falls der Server einem Wechsel auf das WebSocket-Protokoll zustimmt, würde die Antwort des Servers an den Client der folgenden Abbildung entsprechen:

```
1 HTTP/1.1 101 Web Socket Protocol Handshake
2 Date: Fri, 27 Sep 2013 22:40:41 GMT
3 Server: Kaazing Gateway
4 Upgrade: WebSocket
5 Access-Control-Allow-Origin: http://www.websocket.org
6 Access-Control-Allow-Credentials: true
7 Sec-WebSocket-Accept: /fR+hvEPAWg/Xc3823b4+TqjlKE=
8 Connection: Upgrade
9 Access-Control-Allow-Headers: content-type
```

Abbildung 6 : Serverseitige Antwort an einen Client bei einem WebSocket-Handshake.

Quelle: Handshake mit <http://www.websocket.org/echo.html> in einem Chrome Browser.

Letzter Zugriff: 27. September 2013

Eine solche, als Handshake bezeichnete, Kommunikation (Abbildung 5 und 6) für eine WebSocket-Verbindung findet nur einmalig beim Verbindungsaufbau über das HTTP-Protokoll statt. Wohingegen HTTP bei jeder neuen Anfrage eine größere Menge an Verwaltungsdaten, wie den Header-Informationen, sendet.

Die eigentliche Kommunikation über einen WebSocket enthält im Vergleich zu HTTP eine wesentlich geringere Menge an Verwaltungsdaten²⁰, was mit Hilfe der Entwicklerwerkzeuge eines Browsers eingesehen werden kann.

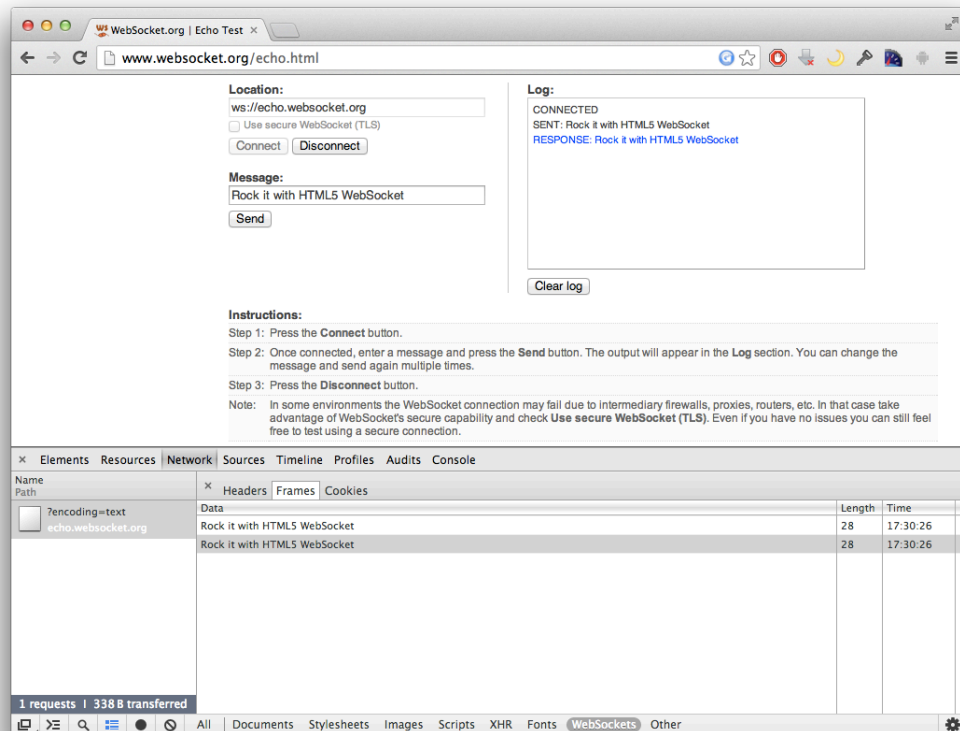


Abbildung 7: Nachrichtenübertragung mittels einer WebSocket Verbindung im Webbrowser Google Chrome auf der Echo-Testseite.

Quelle: Kommunikation mit <http://www.websocket.org/echo.html> in einem Chrome Browser.

Letzter Zugriff: 27. September 2013

2.1.4 Express

Das serverseitige Webapplikations-Framework Express dient der Vereinfachung der Konstruktion von Applikationen im REST-Architekturstil²¹ auf Basis von Node.js und kann parallel dafür benutzt werden statische Inhalte auszuliefern.

²⁰ Weiterführende Informationen zum WebSocket-Header findet sich auf der Seite der Internet Engineering Task Force: <http://tools.ietf.org/html/rfc6455>

²¹ Vorlesung zu REST-Architektur von Prof. Dr. S. Alda :
Quelle: http://www.sascha-alda.de/lehre/soa/vorlesung/Kapitel_8_-_REST.pdf
Letzter Zugriff: 2. September 2013

Dabei ist es möglich die gängigen HTTP-Verben und die empfangenen Parameter auf eine kompakte Weise miteinander zu verknüpfen (siehe Abbildung 8).

```
1 var express = require('express');
2 var ap = express();
3
4 app.get('/apiv1/user/:user_id', function(req, res){
5   res.send(req.params.user_id);
6   // do some awesome stuff
7 });
8 app.listen(80);
```

Abbildung 8: Konstruktion einer REST-API mit dem Express Framework.

Quelle: SublimeText2, eigenes Beispiel.

Die oben dargestellte Applikation liefert auf die HTTP-GET-Anfrage „http://localhost/apiv1/user/12345“ ausschließlich die aus der Anfrage extrahierte user_id „12345“ zurück. Die Bearbeitung von weiteren HTTP-Verben ist mit der gleichen Vorgehensweise möglich. So kann eine, je nach Applikation, unterschiedliche Sammlung an Aktionsmöglichkeiten geschaffen werden, die alle Interaktionsstränge einer Anwendung abbildet.

2.2 HexGL

HexGL ist ein in JavaScript implementiertes Rennspiel vom Entwickler Thibaut Despoulain²², welches stark an den Spieleklassiker WipEout²³ angelehnt ist. Das Spiel setzt eine leistungsstarke Grafikkarte und einen modernen WebGL-fähigen²⁴ Webbrowser für ein flüssiges (30 fps) Spielerlebnis voraus. Da das Spiel²⁵ quelloffen auf Github verfügbar und unter der Creative-Commons-Lizenz veröffentlicht worden ist, kann es für eine quelloffene Implementierung verwendet werden.

²² Weitere Informationen zu Thibaut Despoulain: <http://bkcore.com>

²³ Spieleklassiker WipEout: <http://de.wikipedia.org/wiki/WipEout>

²⁴ WebGL Spezifikationen der Khronos Group: <http://www.khronos.org/registry/webgl/specs/latest/>

²⁵ Projektseite von HexGL: <https://github.com/BKcore/HexGL>

2.3 VirtualJoystick.js

Die in JavaScript implementierte Joystick-Bibliothek²⁶ vom Entwickler Jerome Etienne²⁷ bietet verschiedene Funktionen zur Emulation von virtuellen Joysticks. Dies ermöglicht dem Entwickler einen Joystick an jener Stelle zu positionieren, an der ein Touch-Event im Webbrowser ausgelöst worden ist. Dabei kann die erfolgte Bewegungsrichtung beim Touch-Event erfasst werden.

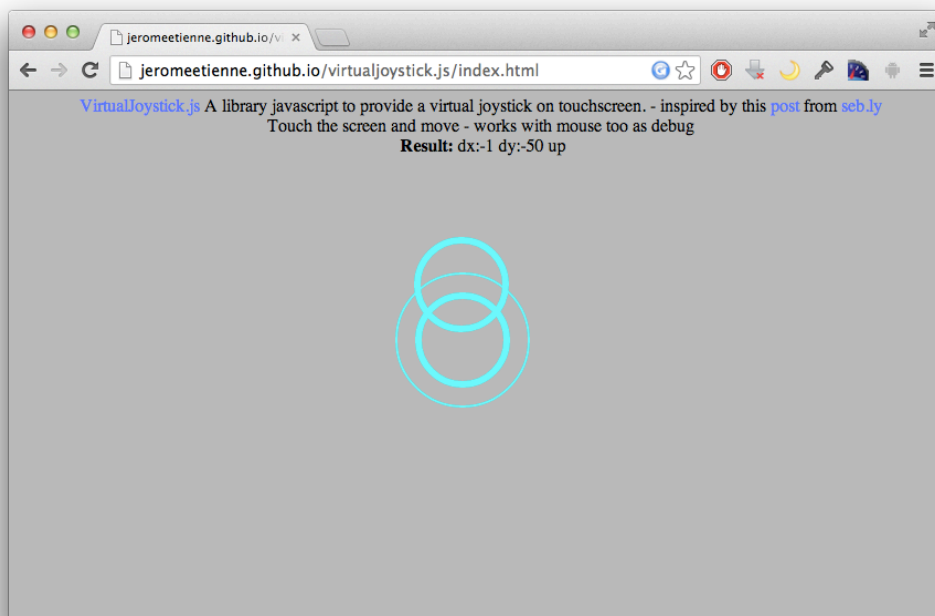


Abbildung 9 : VirtualJoystick.js mit Messung eines Touch-Events. Dabei repräsentiert der zentrale Kreis den Startpunkt, sowie der obere Kreis den Endpunkt der Touch-Events des Joysticks.

Quelle: <http://jeromeetienne.github.io/virtualjoystick.js/index.html> in einem Chrome Browser.

Letzter Zugriff: 1. September 2013

Da die Positionierung des Joysticks dynamisch erfolgt, ist es möglich den gleichen Quellcode auf Geräten unterschiedlicher Größe ohne eine Anpassung auszuführen und dennoch ein ergonomisches Verhalten zu erzielen.

²⁶ Dokumentation der Joystick Bibliothek: <https://github.com/jeromeetienne/virtualjoystick.js>

²⁷ Informationen zu Jerome Etienne: <http://jetienne.com>

3 Aufgabenstellung

Das Ziel dieser Abschlussarbeit ist die Erstellung eines webbasierten Softwaresystems, welches eine Controller-Applikation und einen HTTP- und WebSockets-Backend-Server bereitstellt, sowie die Anpassung der zu steuernden Web-Applikation(Rennspiel HexGL) an die verwendete Kommunikationsmethode. Die Aufgabe lässt sich in drei Teilaspekte gliedern, welche in den folgenden Abschnitten beschrieben werden.

Die gesamten Anforderungen sollen als eine lauffähige Anwendung im Internet auf eine geeignete Weise bereitgestellt werden.

3.1 Steuerungsapplikation

Für die Implementierung der Steuerungsapplikation bestehen unterschiedliche Anforderungen. Sie sollte eine grafische Benutzeroberfläche implementieren, die möglichst unabhängig von dem eigentlichen Einsatzgebiet des Gerätes ist, da eine zu große Fragmentierung bei Größe, Auflösung und Form innerhalb der mobilen Geräteklassen herrscht. Dafür muss unter Berücksichtigung der unterschiedlichen Gerätegrößen und -typen ein geeignetes Layout gefunden werden.

Eine weitere Anforderung an die Applikation ist die Herstellung einer möglichst latenzfreien Kommunikationsverbindung zwischen der Controller-Applikation und dem zu steuernden Rennspiel HexGL, unter Verwendung des WebSocket-Standards. Hierfür ist eine Weiterleitung der Steuerungsbefehle über den HTTP- und Backend-Server notwendig, da für eine WebSocket-Verbindung ein Client und ein Server benötigt werden. Eine direkte WebSocket-Verbindung zwischen zwei oder mehr Clients ist unter Verwendung vom reinen WebSocket-Protokoll nicht möglich.

3.2 Spieleapplikation HexGL

Das Rennspiel HexGL soll die an der Steuerungsapplikation ausgeführten Steuerkommandos empfangen und möglichst verzögerungsfrei umsetzen. Dafür muss das Rennspiel HexGL um eine passende WebSocket-Verbindung zum HTTP- und Backend-Server erweitert werden, und die spieleeigenen Mechanismen müssen zur Steuerung des Spiels angepasst werden.

3.3 HTTP- und Backend-Server

Der Backendserver stellt die Controller-Applikation, das Rennspiel HexGL sowie eine zentrale Startseite via HTTP bereit. Der Server soll in der Lage sein, WebSocket-Verbindungen zu akzeptieren und eine Datenverbindung zwischen der Controller-Applikation und dem Rennspiel HexGL via WebSockets herzustellen. Diese dient der Übertragung von Steuerkommandos. Auf dem Server sollen mehrere Spiele gleichzeitig laufen können, ohne sich gegenseitig zu behindern. Daher muss eine geeignete Methode implementiert werden, um die einzeln laufenden Spiele von einander zu trennen.

4 Anforderungsdefinition

Im Folgenden werden Anforderungen für die Implementierung dieser Abschlussarbeit spezifiziert, die zu großen Teilen in ein Lastenheft überführt werden können.

4.1 Zielbestimmungen

Im Rahmen dieser Abschlussarbeit wird ein Softwaresystem erstellt, welches dem Benutzer ermöglicht, ein mobiles Gerät (z.B. Smartphone oder Tablet-Computer) mit einem modernen Webbrowser für die Steuerung einer webbasierten Spieleanwendung zu nutzen.

4.2 Produkteinsatz

Das Softwaresystem dient der Steuerung eines webbasierten Computerspiels durch einen menschlichen Spieler unter Einsatz eines mobilen Gerätes zur Steuerung des Spiels. Der Spieler soll durch das Aufrufen einer Webseite auf beiden Geräten in die Lage versetzt werden, ein Spiel zu starten und die Kontrolle über das Spielgeschehen, welches auf dem Desktop-Computer stattfindet, auf dem mobilen Gerät zu erhalten. Dabei sollen die auf dem mobilen Gerät ausgeführten Steuerungsbefehle vom Spiel empfangen und umgesetzt werden. Dies ist besonders in Umgebungen interessant, in denen das Bereitstellen eines physikalischen Gamecontroller nicht möglich, unverhältnismäßig oder zu riskant ist. Ein weiterer Einsatzort für ein solches Softwaresystem könnten Abstimmungen bei Events oder andere Großveranstaltungen sein, bei denen eine direkte Interaktion mit einem Publikum von Vorteil wäre.

4.3 Produktfunktionen

/F10/	<p>Prozess: Spielen</p> <p>Akteur: Spieler</p> <p>Beschreibung: Der Spieler öffnet die Webseite des Softwaresystems sowohl auf seinem Desktop-Computer als auch auf seinem mobilen Gerät. Eine dadurch entstandene Verbindung ermöglicht es dem Spieler die Kontrolle über das Spielgeschehen auf seinem Desktop-Computer zu übernehmen.</p>
/F20/	<p>Prozess: Systemcheck</p> <p>Akteur: Administrator</p> <p>Beschreibung: Durch das Aufrufen des vom Softwaresystem geführten Ereignisprotokolls ist es dem Administrator möglich, Einsicht in die Menge der Spielesitzungen zu bekommen.</p>

4.4 Produktdaten

/D10/	<p>Das Softwaresystem speichert keine der anfallenden Daten und verwirft das Ereignisprotokoll bei jedem Neustart des Softwaresystems.²⁸</p>
-------	---

²⁸ Auf den ersten Blick scheinen die anfallenden Verbindungsdaten und Zusatzinformationen der Webbrowser ohne große Sicherheitsbedenken gespeichert werden zu können. Neuere Tests der Electronic Frontier Foundation²⁸ stehen der Identifizierbarkeit eines Webbrowsers jedoch entgegen. Weitere Informationen zum Test der Einzigartigkeit des benutzen Webbrowsers finden sie in dem folgenden Link: <http://panoptickick.eff.org>

4.5 Produktleistungen

- /L10/ Das Softwaresystem soll bis zu fünf parallel laufende Spielesitzungen eines Spiels von unterschiedlichen Spielern verwalten können.
- /L20/ Das Softwaresystem soll eine für mobile Geräte geeignete Methode zur Steuerung des Spiels bereitstellen.
- /L30/ Dem Benutzer soll es möglich sein, eine eigene Spiel-ID zum verbinden der Geräte für eine zu startende Spielesitzung zu nutzen. Dies erfordert die Eingabe der Spiele-ID auf beiden, bei einem Spiel verwendeten Geräte.

4.6 Betriebsbedingungen

Für eine Nutzung der Beispielapplikation muss zwingend eine Verbindung zum Internet mit einer Mindestgeschwindigkeit von 1 MBit/s auf allen Geräten vorhanden sein. Alternativ kann eine lokal vorliegende Version in Form einer virtuellen Maschine oder eine Installationsinstanz der Softwareanwendung auf einem Desktop-Computer gestartet werden. Auch hierbei ist eine Netzwerkanbindung aller drei Geräte untereinander erforderlich.

4.7 Benutzungsschnittstelle

- /B10/ Die Benutzungsschnittstelle der webbasierten Steuerungsapplikation ist auf ein berührungsempfindliches Display auszulegen.

- /B20/ Die Benutzungsschnittstelle des Administrators ist auf eine reine Ausgabe des Ereignisprotokolls in der Konsolenapplikation zu beschränken.
- /B30/ Die Benutzungsschnittstelle des webbasierten Computerspiels ist nicht zu verändern.

4.8 Technische Produktumgebung

4.8.1 Software

- /US10/ Die webbasierte Spieleapplikation benötigt einen HTML5 fähigen Webbrowser. Dessen Spezifikationen sollen Tests der HTML5-Test-Webseite²⁹, in den für die Softwareplattform notwendigen Punkten von WebSockets und WebGL-3D-Kontext, vollständig erfüllen.
- /US20/ Die webbasierte Steuerungsapplikation benötigt einen HTML5 fähigen Webbrowser, dessen Spezifikationen die Tests der HTML5-Test-Webseite³⁰, in denen für die Softwareplattform relevanten Punkten von WebSockets und SVG, erfüllen.
- /US30/ Die Serverapplikation benötigt eine in der package.json genau spezifizierte Node.js-Laufzeitumgebung.

²⁹ Diese Webseite prüft die HTML5-Fähigkeiten des Webbrowsers speziell in Hinsicht auf die Verwendung auf dem Desktop. Die Punkte WebSockets und WebGL sind dabei für diese Softwareplattform besonders relevant und sollten vom benutzten Browser auf einem Desktop-Computer erfüllt werden.

<http://html5test.com/compare/browser/chrome29/ff24/mybrowser.html>

³⁰ Diese Webseite prüft die HTML5-Fähigkeiten des Webbrowsers speziell in Hinsicht auf die Verwendung auf einem mobilen Endgerät. Die Punkte WebSockets und SVG sind dabei für die Softwareplattform besonders relevant und sollen vom benutzten Browser auf einem mobilen Gerät erfüllt werden.

<http://html5test.com/compare/browser/chromemobilebeta/ios70/mybrowser.html>

/US40/ Alle aufgeführten Teile der Softwareapplikation benötigen einen funktionierenden Softwareprotokollstapel, um untereinander eine Netzwerkverbindung via WebSockets und HTTP herstellen zu können.

4.8.2 Hardware

/UH10/ Verpflichtend für den Betrieb der Spieleanwendung ist eine leistungsstarke Grafikkarte mit WebGL-Unterstützung³¹, sowie die sich daraus ergebenden Randbedingungen.

/UH20/ Die minimale Prozessorgeschwindigkeit des mobilen Gerätes beträgt 800 MHz. Dabei sollte das Gerät mindestens 256 MB Arbeitsspeicher aufweisen.

/UH30/ Die Serverapplikation benötigt mindestens einen 1GHz Prozessor und 256 MB Arbeitsspeicher. Alternativ genügt auch eine ausführende Einheit eines Anbieters³² im Internet.

/UH40/ Alle beschriebenen Geräte benötigen eine physikalische Netzwerkschnittstelle über die der Netzwerkprotokollstapel eine Verbindung etablieren kann.

³¹ Eine Liste der WebGL-fähigen Grafikkarten unter Einsatz auf unterschiedlichen Betriebssystemen kann dem Wiki der Khronos Group Inc. entnommen werden. Link : <http://www.khronos.org/webgl/wiki/BlacklistsAndWhitelists>

³² Um die zur Verfügung stehende Rechenleistung eines Platform as a Service Anbieters beziffern zu können, gibt es keine einheitliche Maßeinheit. Jeder Anbieter hat eigene Beschreibungen zu diesem Zweck.

Anbieter Nodejitsu bezeichnet es als Drone: <https://www.nodejitsu.com/paas/faq/#what-is-a-drone>

Anbieter Heroku bezeichnet es als ein Dyno: <https://devcenter.heroku.com/articles/dynos>

4.8.3 Orgware

/UO10/ Eine Netzwerkverbindung zwischen den Systemen ist zwingend erforderlich. Die Verbindungsgeschwindigkeit ist maßgeblich für die Latenz der Applikation verantwortlich. Optimal ist eine Paketumlaufzeit von unter 33ms zwischen den Geräten.

5 Systementwurf

In den nächsten Abschnitten werden einzelne Teile des Softwaresystems in ihrer Entwurfsphase genauer dargestellt und wichtige Funktionalitäten hervorgehoben. Die Aufteilung des Softwaresystems ergibt sich durch die drei unterschiedlichen Geräte für die Ausführung der Quellcodeteile.

5.1 Steuerungssapplikation

Die Steuerungssapplikation besteht aus den folgenden Teilen, die ineinandergreifen und die eingegebenen Steuerungssignale an den Server weiterleiten.

Zunächst wird eine JSON-Datenstruktur für alle Datenfelder des zu sendenden Zustands der Steuerungssapplikation erzeugt. Dabei wird die Datenstruktur schon so ausgelegt, dass zukünftig mehr als eine Steuerungssapplikation zu einer Spielesitzung gehören kann und diese anhand ihrer `controller_id` in der jeweiligen `controller_session` von einander unterschieden werden können.

```
1 // pluto_key_data as JSON
2 var pluto_key_data = {
3   "controller_session" : pluto_id,
4   "controller_id" : 1,
5   "connection_type": pluto_connection_type,
6   "up" : false,
7   "down" : false,
8   "left" : false,
9   "right": false,
10  "a" : false,
11  "b" : false
12 }
```

Abbildung 10: Datenstruktur, die bei jedem Event übermittelt wird und den Status der Steuerungsanwendung enthält.

Quelle: „./public/controller/js/script.js“

Die `joystick_right` und `joystick_left` Objektinstanzen sind nur für jeweils 50% des Bildschirms zuständig und verfügen über Funktionen für die Abfrage des jeweiligen Zustandes.

```

1 // joystick_left init
2 var joystick_left = new VirtualJoystick({
3     container : document.body,
4     strokeStyle : 'orange'
5 });
6
7 // joystick_left adding a EventListener
8 joystick_left.addEventListener('touchStartValidation', function(event){
9     var touch = event.changedTouches[0];
10    if( touch.pageX >= window.innerWidth/2 )    return false;
11    return true
12 });
13
14 // joystick_left usage
15 pluto_key_data["up"] = joystick_left.up();
16 pluto_key_data["down"] = joystick_left.down();

```

Abbildung 11: Erzeugen der Objektinstanzen, Abfrage der Joystickzustände und Zuweisung in einem JSON für die spätere Versendung der Daten.

Quelle: „./public/controller/js/script.js“

Via Socket.io wird eine Verbindung zum Server hergestellt. Anschließend wird durch die Interval-Funktion alle 16ms das gesamte JSON als ein pluto_data-Event an den Server übermittelt.

```

1 // connecting to actual domain with websocket
2 io = io.connect(document.domain)
3
4 // adding pluto_key_data to the emitter
5 var pluto_emit_data = function(){
6     io.emit('pluto_data', pluto_key_data);
7 }
8
9 // timer for execution ob pluto_emit_data every 16 ms
10 setInterval(function(){pluto_emit_data()},16);

```

Abbildung 12: Instanz von Socket.io mit zugehörigem Event zum Senden von Daten.

Quelle: „./public/controller/js/script.js“

Zusätzlich werden noch einmalig beim ersten Aufruf zwei Piktogramme für das bessere Verständnis der Steuerungsapplikation eingeblendet und der Verbindungsstatus wird in der linken oberen Ecke eingeblendet und aktualisiert. Die genaue Konfiguration kann dem Quellcode entnommen werden.

5.2 Spielapplikation

Die Spielapplikation baut zum Start ebenfalls eine Verbindung via Socket.io zum Server auf und wartet auf den Eingang von `pluto_relay`-Events mit den richtigen Parametern.

```
1 // socket.io connect to current domain via websockets
2 io = io.connect(document.domain);
3
4 // join to current pluto_session
5 io.emit('pluto_join', pluto_id);
6
7 // building route for the pluto_relay event
8 io.on('pluto_relay', function(data){
9   pluto_event_emitter(data);
10 });
```

Abbildung 13: Die Spieleapplikation baut eine Socket.io-Verbindung auf und sendet ein `pluto_join`-Event an den Server als Anmeldung. Ein serverseitiges `pluto_relay`-Event kann danach nur noch mit der richtigen `pluto_id` an die Spieleapplikation gesendet werden. Quellcode: „./public/game/index.html“

Ein für das Spiel erzeugtes Event wird im DOM eingehängt und anschließend bei jedem eintreffenden Event über Socket.io ausgelöst. Die Steuerungsdaten werden so an das laufende Spiel weitergereicht.

```
1 // creating custom event
2 plutoEvent = document.createEvent("Event");
3 plutoEvent.initEvent("pluto_controller_event", true, true);
4 plutoconnected = false;
```

Abbildung 14: Erzeugen eines Events im eigenen Namensraum, um nicht mit eventuell vorhandenen Namen zu interferieren.

Quellcode: „./public/game/index.html“

```
1 // taking the values from JSON
2 function onPlutoEvent(event)
3 {
4   self.key.forward = event.data.up;
5   self.key.backward = event.data.down;
6   self.key.left = event.data.left;
7   self.key.right = event.data.right;
8   self.key.ltrigger = event.data.a;
9   self.key.rtrigger = event.data.b;
10 };
11
12 // adding event to game
13 domElement.addEventListener('pluto_controller_event', onPlutoEvent, false)
```

Abbildung 15: Übergabe der Steuerungskommandos an das Spiel.

Quellcode: „./public/game/bkcore/hexgl/ShipControls.js“

Außerdem wird ein geladenes Spiel erst beim ersten erfolgreich übermittelten Event gestartet. Bei einem reinen Aufruf ohne eingehende Verbindung wird ein Piktogramm als Platzhalter angezeigt, dass ein Warten auf eine eingehende Verbindung symbolisiert. Der Verbindungsstatus wird während des Spiels in der linken oberen Ecke kontinuierlich angezeigt und aktualisiert.

5.3 Serverapplikation

Die Node.js-Serverapplikation bearbeitet alle eintreffenden HTTP- und WebSocket-Anfragen. Dabei hält die Anwendung alle laufenden Sitzungsnummern der WebSocket-Verbindungen in einer Datenstruktur bereit, um die Gültigkeit einer Verbindung verifizieren zu können. Erst bei einer Anmeldung durch die Spieleapplikation mit einer Sitzungsnummer kann die Steuerungsapplikation einmalig ausgeliefert werden. Nur durch eine hergestellte Verbindung der Steuerungsapplikation zur Spieleapplikation, unter Nutzung des Servers, wird das Spiel gestartet.

Die Sitzungsnummern werden dabei erst durch das nahezu zeitgleiche Öffnen der Startseite vergeben und leiten sich von der EPOCH-Time Schreibweise der Uhrzeit ab. Hierfür werden die letzten beiden Stellen verworfen. Dadurch ergibt sich ein Zeitfenster von weniger als 100 Sekunden, um die gleiche Sitzungsnummer zu erhalten. Es kann jedoch auch eine vom Benutzer eingetragene und auf beiden Geräten gleiche Sitzungsnummer verwendet werden.

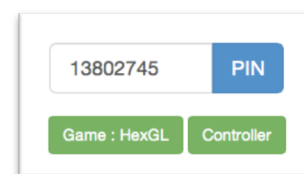


Abbildung 16 : Sitzungsnummer auf der Startseite des Software-systems.

```
1 $(document).ready(function() {
2
3     // Setting up Session-ID from EPOCH-Time
4     var pluto_id = Math.round(new Date().getTime()/100000.0);
5
6     // Adding Session-ID to Buttons
7     $("#pluto_id").val(pluto_id);
8     $("#btn_controller").attr('href', '/controller/'+pluto_id);
9     $("#btn_game").attr('href', '/game/'+ pluto_id);
10
11     // Adding a newly entered Session-ID to Buttons
12     $("#pluto_id").focusout(function(){
13         $("#btn_controller").attr('href', '/controller/'+$("#pluto_id").val());
14         $("#btn_game").attr('href', '/game/'+ $("#pluto_id").val())
15     });
16 });
```

Abbildung 17: Quellcode zur Vergabe von Sitzungsnummern auf der Startseite und das Einhängen von neuen Sitzungsnummern an die Knöpfe der Applikation bei einer Eingabe durch den Benutzer.
Quellcode: „./public/js/script.js“

Um eine Verbindung zwischen Steuerungsapplikation und Spieleapplikation zu etablieren, müssen die folgenden Schritte auf dem Server durchlaufen werden:

- Anmeldung einer Spielesitzung mit einer Sitzungsnummer von der Spieleapplikation beim Server
- Eintragen der Sitzungsnummer als laufende Sitzung
- Auslieferung der Steuerungsapplikation bei Anfrage mit einer gültigen Sitzungsnummer
- Start des Event-Stream durch die Steuerungsapplikation und Weiterleitung an die Spieleapplikation
- Start des Spiels durch das Eintreffen des Event-Stream durch die Spieleapplikation

```
1 // sessions JSON
2 var sessions = {}
3
4 // route with parameters passing
5 app.get('/game/:pluto_pin', function(req, res){
6   sessions[req.params.pluto_pin] = false;
7   res.sendFile(__dirname + '/public/game/index.html')
8 });
9
10 // route for pluto controller and the assigned pin
11 app.get('/controller/:pluto_pin', function(req, res){
12   if (sessions[req.params.pluto_pin] == false){
13     res.sendFile(__dirname + '/public/controller/index.html')
14     sessions[req.params.pluto_pin] = true;
15   }
16   else {
17     // send error
18   }
19 });
```

Abbildung 18: Die REST-Routen des hier genutzten Express-Frameworks gekoppelt an die Datenstruktur zur Speicherung der Sitzungsnummern.

Quellcode: „./server.js“

Der Server des Softwaresystems kann auf folgende Events reagieren:

- HTTP-Get auf „/“
Dies liefert die „/public/index.html“ als Startseite des Softwaresystems zurück und vergibt Clientseitig die Sitzungsnummern.
- HTTP-Get auf : „/game/session_id“
Dies liefert dem Anfragenden die Spieleanwendung zurück und trägt die als „session_id“ übertragenen Parameter als Sitzungsnummer ein.
- HTTP-Get auf „/controller/session_id“

Dies hat eine Prüfung der übergebenen „session_id“ zur Folge. Bei Gültigkeit wird die Steuerungsapplikation ausgeliefert.

- WebSocket-Event: „pluto_data“

Der Server prüft die Sitzungsnummer und leitet die Anfrage bei Gültigkeit an die Spieleapplikation weiter.

- WebSocket-Event: „pluto_join“

Diesem Event wird die Sitzungsnummer zugeteilt, die es sendet. Dadurch ist die angebundene Sitzung der Spieleapplikation eindeutig zu erreichen.

- WebSocket-Event: „pluto_leave“

Dieses Event entfernt die bestehende Verbindung zur Spieleapplikation.

Ein Überblick der Abläufe der Serverapplikation befindet sich als Flussdiagramm im Anhang dieses Dokuments. (siehe Abbildungsverzeichnis: Flussdiagramm)

6 Realisierung

Im Folgenden werden die Details der Implementierung des Softwaresystems dargestellt. Da direkt beim Anlegen der Applikation die Notwendigkeit bestand einen Namen zu vergeben, wurde hierbei „Pluto“ gewählt. Dieser Name hat keine tiefere Bedeutung, ist jedoch bei Vergabe von Variablennamen und als Präfix bei Events zum Einsatz gekommen, um sich von eventuell parallel laufender Software abzugrenzen.

Bei der Implementierung der Applikation tritt sehr häufig die Verwendung des Designpattern „Publish-Subscribe“ (aka. Observer) auf. Diese Tatsache ergibt sich durch die Verwendung von Socket.io und dem ereignisgesteuerten Applikationsfluss der Softwareplattform.

6.1 Entwicklungsprozess

Der Entwicklungsprozess fand unter Einsatz von Node.js 0.10.17 auf einem Mac OS X 10.8 statt. Das dabei genutzte Gerät verfügt über einen 1,8 GHz Intel Core i5 Prozessor, 4 GB Arbeitsspeicher, sowie eine Intel HD4000 Grafikkarte.

Zum Testen der Steuerungsapplikation auf mobilen Geräten wurden die im Folgenden aufgelisteten Geräte und Konfigurationen verwendet:

- GeeksPhone Keon
 - Firefox OS
 - 1GHz Prozessor
 - 512 MB Arbeitsspeicher
 - OS Version 1.1.1.0-pre-release
- Samsung Galaxy Player
 - Android 2.3
 - 1 GHz Prozessor
 - 512 MB Arbeitsspeicher

- Firefox 24.0
- Chrome 29.0
- Apple iPhone 5
 - iOS 7
 - 1.3 GHz Dual Core Prozessor
 - 1 GB Arbeitsspeicher
 - Safari
- IconBIT Pocket
 - Android 4.0
 - 1.3 GHz Prozessor
 - 512 MB RAM
 - Chrome 29.0
 - Firefox 24.0

Alle diese Geräte verfügen über vollständige WebSocket-Unterstützung in den jeweiligen Webbrowsern.

Als Versionsverwaltung wurde Git³³ in Form von GitHub³⁴ verwendet. Das Repositorium befindet sich unter: „<https://github.com/eugenpirogoff/pluto>“ und ist öffentlich zugänglich.

Für die einheitliche Verwaltung von Frontend-Komponenten wurde Bower³⁵ zum Installieren von jQuery, Bootstrap und Schriftarten verwendet. Bower benutzt dafür eine „`bower.json`“-Konfigurationsdatei, die der von NPM sehr ähnlich ist und geht beim Speichern der Bibliotheken auf die gleiche Weise vor. Da es sich hier um FrontEnd-Bibliotheken handelt, wurden diese unter „`./public/components`“ abgelegt.

Neben einer lokalen Instanz von Node.js wurde für Testzwecke auch eine Nodejitsu-Instanz³⁶ des Softwaresystems betrieben. Dadurch ergibt sich folgender Auslieferungsprozess:

³³ Git ist ein verteiltes Versionskontrollsystem und de facto Standard in der JavaScript-Nutzergruppe.

Projektseite: <http://git-scm.com>

³⁴ GitHub ist ein Git-Hosting-Dienst für Softwareentwicklungsprojekte. Projektseite: <https://github.com>

³⁵ Bower ist ein Paketmanager für Frontend-JavaScript Komponenten. Projektseite: <http://bower.io>

³⁶ Nodejitsu ist ein Platform as a Service Anbieter für Node.js Applikationen. Projektseite: <http://nodejitsu.com> .

Git-Repository auf Entwicklungsmaschine

- Der Quellcode wird lokal auf dem Computer editiert und zu Testzwecken ausgeführt.
- Eine Ausführung mit nodemon ist ideal, da diese bei Änderungen und Fehlern die Instanz von Node.js neu startet.

Nodejitsu in der Cloud

- Das Repository kann mit einem `jitsu deploy` auf eine Instanz hochgeladen werden.
- Nodejitsu installiert alle Abhängigkeiten aus der `package.json` nach und startet die Applikation.

Ausführen in der Cloud

- Die Applikation wird von Nodejitsu geladen und mit den Umgebungsvariablen wie Portnummer und DNS-Namen konfiguriert.
- Applikation steht für Anfragen unter einer Subdomain bereit und kann getestet werden.

Abbildung 19: Diese Abbildung zeigt den verwendeten Workflow des Entwicklungsprozesses.

Quelle: eigene Grafik

Die zugehörige Adresse der Applikation lautet: „<http://pluto.jit.su>“. Weitere Informationen zum Auslieferungsprozess auf Nodejitsu kann man dem Kapitel Installationsanleitung entnehmen.

6.2 HTTP-WebSocket-Server und Bibliotheken

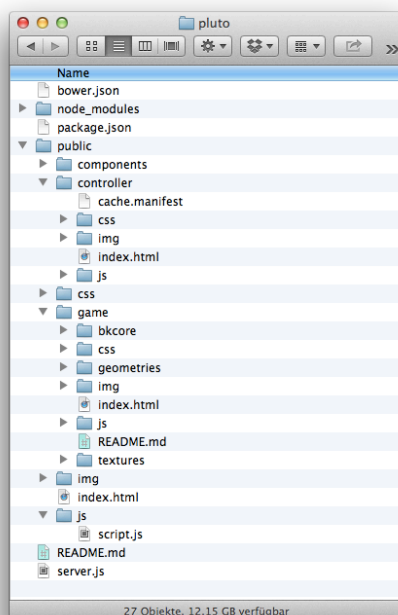


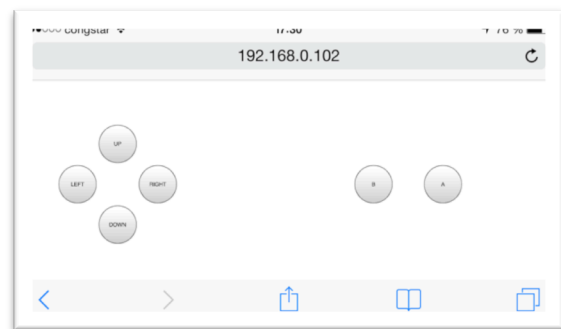
Abbildung 20: Ordnerstruktur von Pluto

Das gesamte Softwaresystem ist serverseitig in der in Abbildung 20 sichtbaren Ordnerstruktur untergebracht. Hierbei wird ersichtlich, dass die Frontend- und Backend-Komponenten von einander getrennt worden sind. Jede Komponente hat einen eigenen Ordner und jeweils eigene JavaScript- und CSS-Ordner für den verwendeten Quellcode des jeweiligen Programmteils. Die genutzten Bibliotheken

wurden von NPM bei der Installation in „./node_modules“ und von Bower in „./public/components“ gespeichert. Die genaue Auflistung aller benötigten Bibliotheken findet sich in der „package.json“ und in der „bower.json“ im Wurzelverzeichnis des Softwaresystems.

6.3 Steuerungssapplikation

Bei der Entwicklung der Steuerungssapplikation ist das Design der anfänglich angestrebten Version mehreren Iterationen unterworfen worden. Dabei zeigte sich, dass eine auf reinen Knöpfen



basierende Version einer *Abbildung 21: Erste Implementierung der Steuerungssapplikation.* Steuerungssapplikation für ein

Rennspiel nicht praktikabel ist. Sowohl die Erreichbarkeit der Knöpfe auf unterschiedlich großen Geräten, als auch das unpräzise Handling der Knöpfe und das Auslösen der Events im Browser, erwiesen sich dabei als besonders nachteilig. Ein auf Knöpfen basierendes Layout wurde zu Gunsten der VirtualJoystick.js-Bibliothek verworfen.

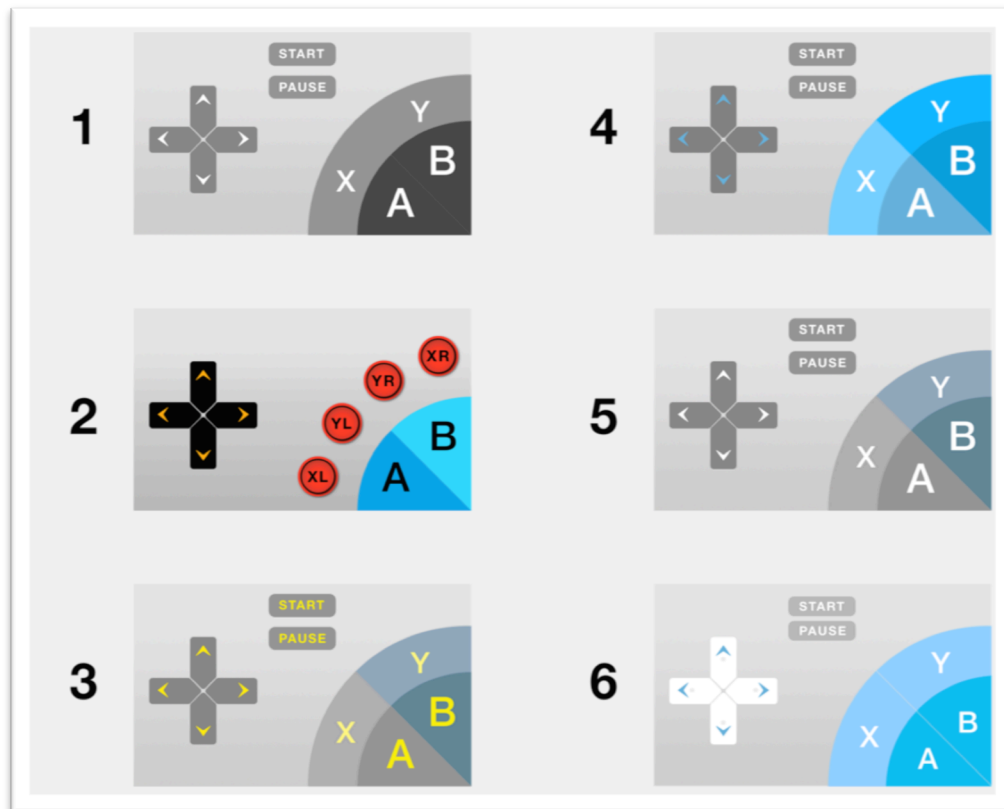


Abbildung 22 : Ein auf Knöpfen basiertes Layout für eine Steuerungsapplikation. Keiner der hier abgebildeten Entwürfe war komplett zufriedenstellend in der Handhabung, Geschwindigkeit oder Design. Aus diesem Grund wurden alle Entwürfe verworfen.

6.4 Spieleapplikation

Die Spieleapplikation wurde bei der Implementierung um einen Event-Emitter für eine Steuerung des Spiels durch Daten erweitert. Dabei werden die durch Socket.io ankommenden Daten weitergeleitet und vom Spiel als Steuerungskommando interpretiert. Im Hinblick auf die Benutzerfreundlichkeit wurde das Spiel um einen Wartebildschirm erweitert. Dabei startet das Spiel selbst erst nach einer erfolgreichen Datenverbindung durch die Steuerungsapplikation.

7 Verbesserung der Implementierung

Bei der erfolgreichen Implementierung mit WebSockets hat sich gezeigt, dass durch die Verwendung eines Servers zur Kommunikation erhebliche Latenzzeiten entstehen. Um dies zu verhindern kann WebRTC eingesetzt werden um den Nachteil eines Servers in der Kommunikationskette zu umgehen. Die Latenzzeiten bei einer solchen Implementierung sind je nach Geschwindigkeit der Verbindung und Entfernung zum Server unterschiedlich und oft zu lang. Eine unerwünschte Latenzzeit könnte mit dem Einsatz von WebRTC³⁷ und dem Verzicht auf einen Server in der Kommunikationskette eliminiert werden. Leider ist die Implementierung des WebRTC-Standards noch nicht so weit fortgeschritten, so dass diese Technik bisher nur ab Chrome Mobile 29 und Firefox Mobile 24 Beta auf mobilen Geräten verwendet werden kann.

7.1 PeerJS

Die PeerJS-Bibliothek³⁸ stellt eine Abstraktion für WebRTC³⁹ dar, die direkte Peer-to-Peer-Kommunikation zwischen den Webbrowsern ermöglicht.

Da sich die Teilnehmer einer Peer-to-Peer-Verbindung mit PeerJS anfänglich erst finden müssen, wird ein PeerJS-Server⁴⁰ als Quellcode mitgeliefert. Für Entwicklungszwecke wird eine kostenlose PeerJS-Server-Instanz⁴¹ für bis zu 50 gleichzeitige Verbindungen bereitgestellt.

Um einen Geschwindigkeitsvorteil gegenüber der WebSocket-Implementierung zu haben, müssen die verwendeten Webbrowser WebRTC auf beiden Geräten unterstützen. Diese Unterstützung ist auf einem mobilen Gerät mit Chrome Mobile Beta 30.0 und auf dem Desktop mit Chrome 29.0 gegeben.

³⁷ WebRTC <http://www.w3.org/TR/webrtc/>

³⁸ Bibliothek für einfacheren Umgang mit WebRTC: <http://peerjs.com>

³⁹ Standardisierung von WebRTC: <http://www.w3.org/TR/webrtc/>

⁴⁰ Quellcode des Peering-Server auf Github: <https://github.com/peers/peerjs-server>

⁴¹ Kostenlose Instanz des Peering-Servers für Entwickler: <http://peerjs.com/peerserver>

7.2 Installation

Da zur Paketverwaltung von Front-End JavaScript-Bibliotheken vorab Bower genutzt worden ist, gestaltet sich eine Installation von PeerJS auch nachträglich sehr simpel. Die Ausführung von „`bower install peerjs --save`“ mit einem Terminal im Wurzelverzeichnis der Applikation genügt, um die Bibliothek in „`./public/components/peerjs`“ von Bower ablegen zu lassen.

7.3 Implementierung

Die bestehenden Implementierungen der Steuerungsapplikation und Spieleapplikation wurden beide mit PeerJS um die Möglichkeit der Kommunikation mit WebRTC erweitert. Dies erfordert jedoch die Vergabe von zwei unterschiedlichen Identifikationsnummern für die Kommunikation. Dafür gilt die folgende Konvention für den Prefix:

```
1 var peerjs_steuerungsapplikation = „controller“+Sitzungsnummer;  
2 var peerjs_spieleapplikation = „host“+Sitzungsnummer;
```

Abbildung 23 : Zusammenbauen der Identifikationsnummern für eine PeerJS-Sitzung nach vorher festgelegter Konvention.

```
1 // PeerJS Communication  
2 // PeerJS API Key => "fcdc4q2kljqc5mi", 50 concurrent connections max  
3 //  
4 // ID naming convention  
5 // Pluto Controller ID = 'controller' + Pluto Session ID  
6 // Pluto Host ID = 'host' + Pluto Session ID  
7 //  
8 var peer = new Peer('host'+pluto_id, {key: 'fcdc4q2kljqc5mi'});  
9 peer.on('connection', function(connection){  
10     connection.on('data', function(data){  
11         pluto_event_emitter(data);  
12     });  
13 });
```

Abbildung 24: Eingehende PeerJS-Verbindung im Teil der Spieleapplikation. Hierbei werden die Daten an den Event-Emitter „`pluto_event_emitter`“ übergeben.

Um die Verbesserung umfassend zu integrieren, wurden die Anzeigen des aktuellen Verbindungsstatus um den Punkt „WebRTC“, sowohl in der

Spieleanwendung als auch in der Steuerungsapplikation erweitert. Die Funktion zum Übermitteln der Daten innerhalb der Steuerungsapplikation wurde so umgebaut, dass diese sowohl über PeerJS als auch über Socket.io senden kann. Dies geschieht in Abhängigkeit der übergebenen Daten.

```
1 var pluto_emit_data = function(){
2   // Left Joystick
3   pluto_key_data["up"] = joystick_left.up();
4   pluto_key_data["down"] = joystick_left.down();
5
6   // Right Joystick
7   pluto_key_data["a"] = joystick_right.left(); // softer steering
8   pluto_key_data["b"] = joystick_right.right(); // softer steering
9
10  // Setting connection Type in JSON
11  pluto_key_data["connection_type"] = pluto_connection_type;
12
13  switch (pluto_connection_type) {
14    case "WebSocket":
15      io.emit('pluto_data', pluto_key_data);
16      break;
17    case "WebRTC":
18      pluto_send_webrtc(pluto_key_data);
19      break;
20  }
21 }
```

Abbildung 25: Funktion zum Senden der Daten in der Steuerungsapplikation. Je nach Verfügbarkeit wird über WebRTC oder WebSocket gesendet.

8 Installationsanleitung und Deploy

8.1 Virtuelle Maschine

Die auf der DVD in Form einer Open Virtualization Format Datei, `pluto_vm.ovf`, gespeicherte virtuelle Maschine kann in einer VirtualBox-Installation auf einem Desktop Computer importiert und danach gestartet werden. Dabei ist nach dem Import in den Netzwerkeinstellung der virtuellen Maschine zu prüfen, auf welcher Netzwerkschnittstelle die Maschine eingehängt werden soll. Es ist in diesem Fall empfehlenswert das Interface zu selektieren, über welches das Gerät mit dem Internet verbunden ist (siehe Abbildung 26).

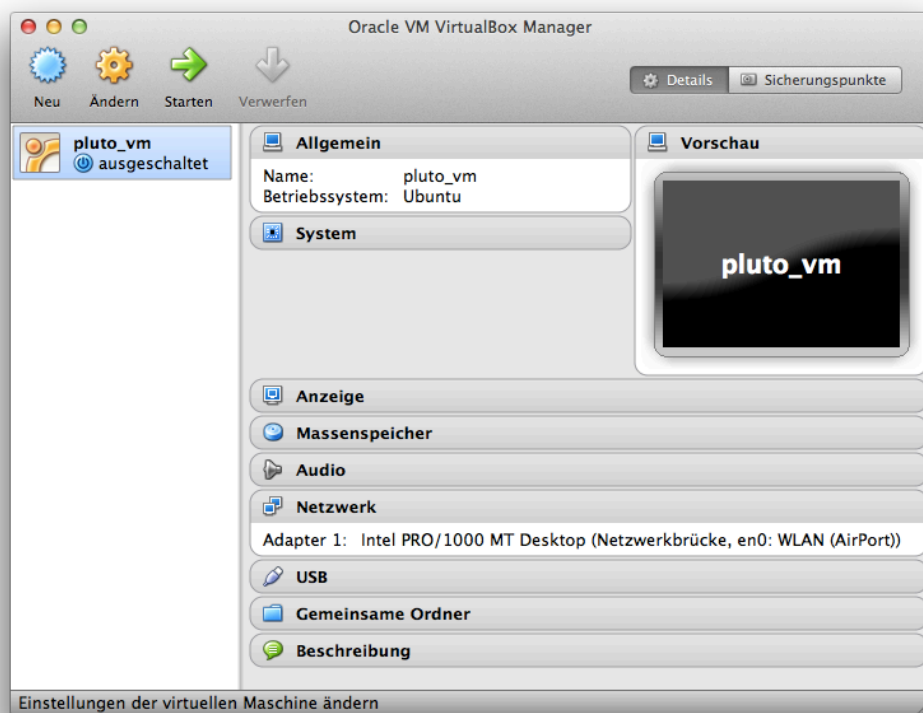


Abbildung 26: So sieht die virtuelle Maschine nach dem Import auf einem Mac OS X 10.8 aus. Um in die Konfiguration zu gelangen, ist es notwendig auf das Icon „Ändern“ zu klicken.

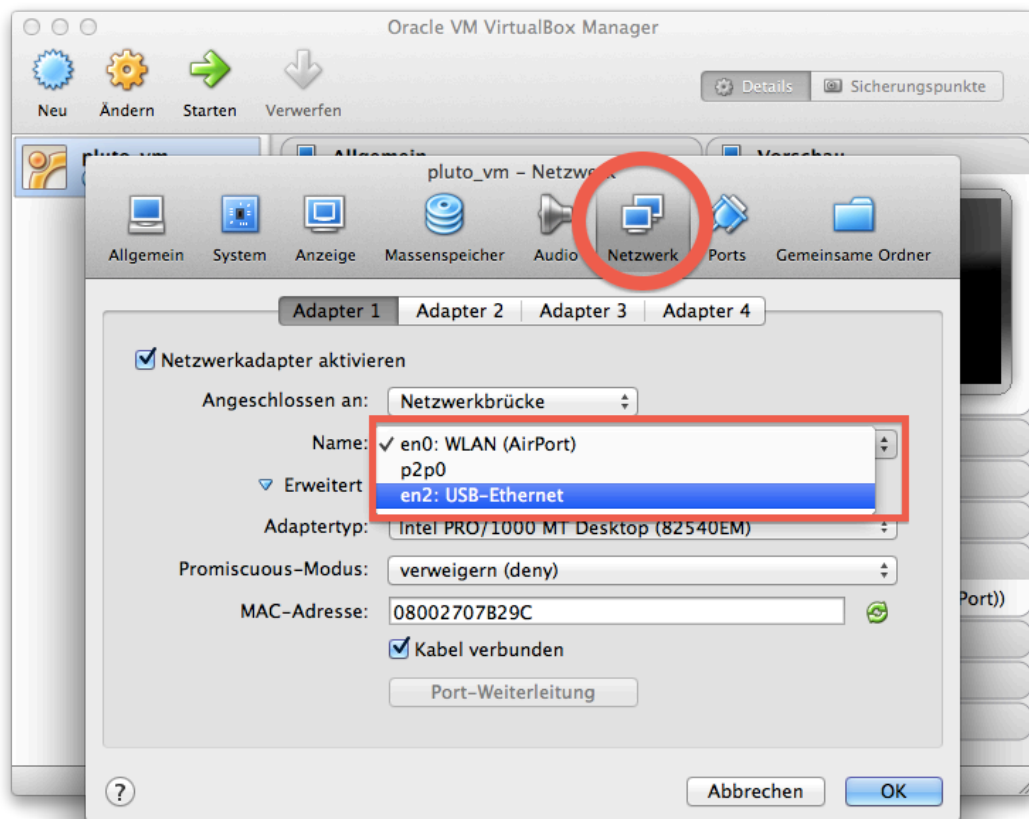


Abbildung 27: In dem Reiter für die Netzwerkeinstellungen ist bei den Einstellungen für den „Adapter 1“ beim Namen die gewünschte Netzwerkschnittstelle zu selektieren.

Nach dieser Anpassung kann die virtuelle Maschine gestartet werden. Die folgende Benutzername-/Passwortkombination wurde für die Anmeldung festgelegt:

Benutzername: deploy

Passwort: deploy

Es ist noch darauf hinzuweisen, dass die Autostartfunktionalität des Softwaresystems nur im „Lubuntu-Netbook“-Desktop der virtuellen Maschine ausgeführt wird.

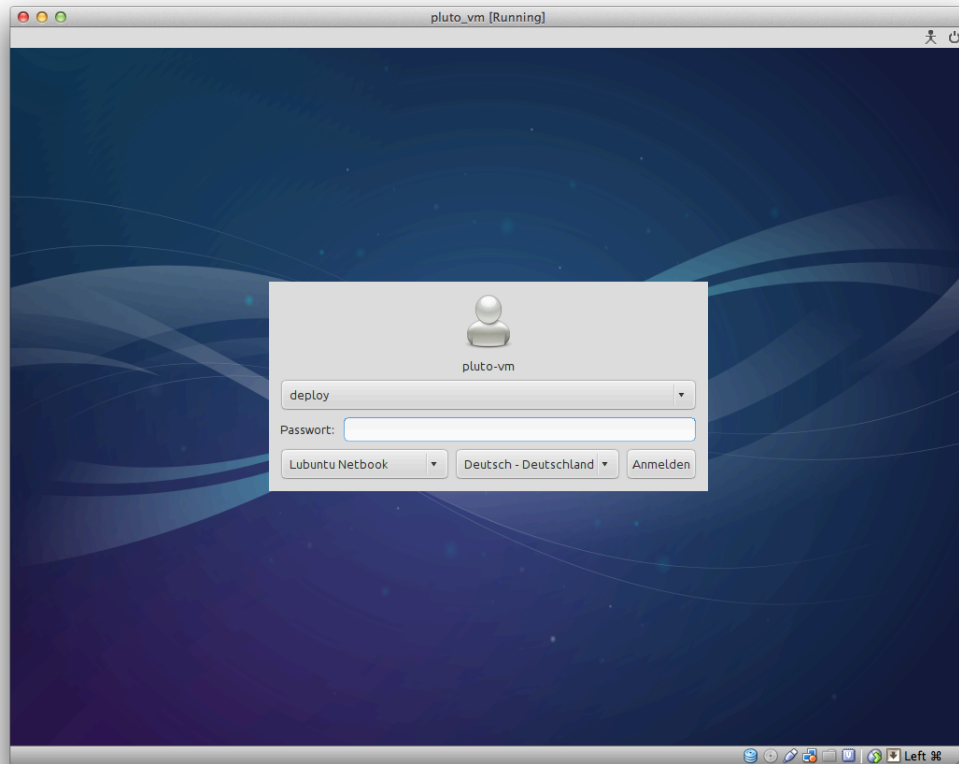


Abbildung 27: Anmeldemaske der virtuellen Maschine des Softwaresystems.

Nach dem Login ist es empfehlenswert durch das Ausführen des Scripts „/home/deploy/Desktop/myip“ im Terminal, die verwendete IP-Adresse des Servers herauszufinden. Die Adresse kann nun außerhalb der virtuellen Umgebung, auf dem Spiele-System oder einem anderen die Spezifikationen⁴² erfüllenden Computer im Netzwerk, unter der IP auf dem Port „3000“ aufgerufen werden. Gleichzeitig sollte die selbe Adresse auf dem mobilen Gerät geöffnet werden. Anschließend sollten die Sitzungsnummern, die als PIN dazu dienen die laufenden Spiele einander zuzuweisen, verglichen werden. Nun kann auf dem Desktop das Spiel, und kurz danach auf dem mobilen Gerät der Controller geöffnet werden. Wichtig dabei ist die Port-Nummer nicht zu vergessen.

Die Adresse könnte dabei wie folgt aussehen: „http://192.168.178.1:3000“

⁴² Die zu erfüllenden Softwarespezifikationen: siehe dazu /US10/
Die zu erfüllenden Hardwarespezifikationen: siehe dazu /UH10/

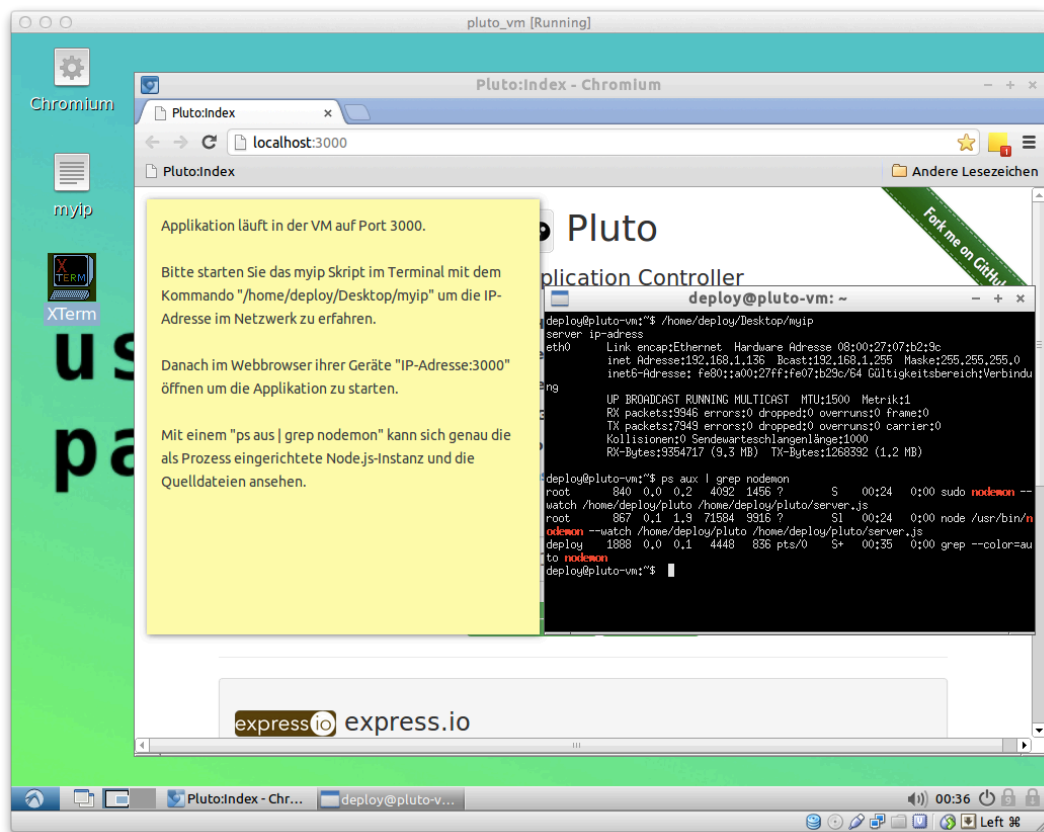


Abbildung 28: Hochgefahrne Applikation und die zugehörige IP-Adresse im Terminal.

8.2 Native Installation

Um das Softwaresystem auf einem Gerät zu installieren, sind die folgenden Schritte notwendig:

- Installation der folgenden Werkzeuge:
 - Git (Download : <http://git-scm.com/downloads/>)
 - Node.js (Download : <http://nodejs.org/download/>)
- Diese Befehle führen die Installation durch:
 - `git clone https://github.com/eugenpirogoff/pluto.git`
Herunterladen des Quellcodes
 - `cd pluto`
Verzeichniswechsel in „pluto“
 - `npm install`
Installation der in package.json definierten Abhängigkeiten

- `node server.js`
Start des Softwaresystems

```

Eugen noir ~ $ git clone https://github.com/eugenpirogoff/pluto.git
Cloning into 'pluto'...
Username for 'https://github.com': eugenpirogoff@me.com
Password for 'https://eugenpirogoff@me.com@github.com':
remote: Counting objects: 1412, done.
remote: Compressing objects: 100% (1224/1224), done.
remote: Total 1412 (delta 283), reused 1256 (delta 131)
Receiving objects: 100% (1412/1412), 10.88 MiB | 2.24 MiB/s, done.
Resolving deltas: 100% (283/283), done.
Checking connectivity... done
Eugen noir ~ $ cd pluto/
Eugen noir ~ > pluto master $ npm install
npm http GET https://registry.npmjs.org/express.io
npm http 304 https://registry.npmjs.org/express.io
npm http GET https://registry.npmjs.org/underscore/1.4.3
npm http GET https://registry.npmjs.org/socket.io
npm http GET https://registry.npmjs.org/async/0.1.22
npm http GET https://registry.npmjs.org/coffee-script/1.4.0
npm http GET https://registry.npmjs.org/express
npm http 304 https://registry.npmjs.org/express
npm http 304 https://registry.npmjs.org/async/0.1.22
npm http 304 https://registry.npmjs.org/underscore/1.4.3
npm http 304 https://registry.npmjs.org/coffee-script/1.4.0

```

Abbildung 29: Lokale Installation des Softwaresystems mit `git clone` und `npm install` in einem Terminal.

8.3 Deploy auf Nodejitsu

Um einen Deploy des Softwaresystems auf dem Plattform as a Service Anbieter Nodejitsu zu erreichen, ist es notwendig die „package.json“ um einige Punkte zu erweitern und das von Nodejitsu bereitgestellte Werkzeug „jitsu“⁴³ auf seinem Computer zu installieren.

Da es sich hierbei um ein webbasiertes Softwaresystem handelt, ist die Adresse in Form einer Subdomain für die Verbindungen essenziell.

Die Top Level Domain „jit.su“ wird dabei von Nodejitsu bereitgestellt und verwaltet.

```

1 "subdomain": "pluto",
2 "domains": [
3   "pluto.jit.su",
4   "www.pluto.jit.su/"
5 ]

```

Abbildung 30: Domains in der „package.json“

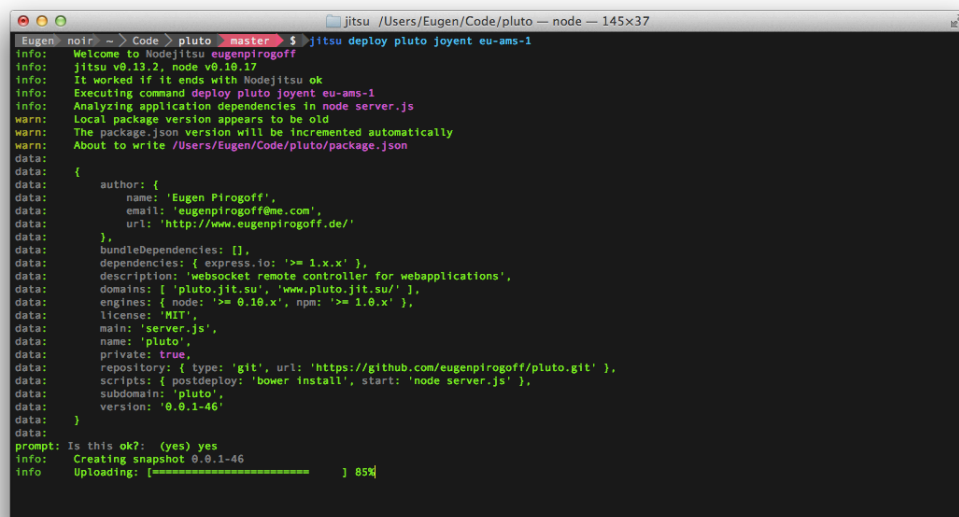
⁴³ Jitsu ist eine Konsolenapplikation zum Ausliefern der Software auf die Nodejitsu-Plattform.
Projektseite: <https://github.com/nodejitsu/jitsu>

Des Weiteren wird das Softwaresystem durch die Umgebungsvariablen von Nodejitsu für eine Verwendung auf Port 80 eingestellt.

```
1 app.listen(process.env.VCAP_APP_PORT || 3000);
```

Abbildung 31 : Konfiguration der `server.js` für Nodejitsu.

Anschließend kann mit einem „jitsu deploy pluto joyent eu-ams-1“ im Terminal der Deploy der Applikation „pluto“ auf den Server von Joyent in dem Datacenter „eu-ams-1“ in Amsterdam durchgeführt werden.



```
jitsu /Users/Eugen/Code/pluto — node — 145x37
Eugen@noir ~$ jitsu deploy pluto joyent eu-ams-1
info: Welcome to Nodejitsu eugenpirogoff
info: jitsu v0.13.2, node v0.10.17
info: It worked if it ends with Nodejitsu ok
info: Executing command deploy pluto joyent eu-ams-1
info: Analyzing application dependencies in node server.js
warn: Local package version appears to be old
warn: The package.json version will be incremented automatically
warn: About to write /Users/Eugen/Code/pluto/package.json
data: {
data:   author: {
data:     name: 'Eugen Pirogoff',
data:     email: 'eugenpirogoff@me.com',
data:     url: 'http://www.eugenpirogoff.de/'
data:   },
data:   bundleDependencies: [],
data:   dependencies: { express.io: '>= 1.x.x' },
data:   description: 'websocket remote controller for webapplications',
data:   domains: [ 'pluto.jit.su', 'www.pluto.jit.su/' ],
data:   engines: { node: '>= 0.10.x', npm: '>= 1.0.x' },
data:   license: 'MIT',
data:   main: 'server.js',
data:   name: 'pluto',
data:   private: true,
data:   repository: { type: 'git', url: 'https://github.com/eugenpirogoff/pluto.git' },
data:   scripts: { postdeploy: 'bower install', start: 'node server.js' },
data:   subdomain: 'pluto',
data:   version: '0.0.1-46'
data: }
data:
prompt: Is this ok?: (yes) yes
info: Creating snapshot 0.0.1-46
info: Uploading: [=====] 85%
```

Abbildung 32: Deploy aus der Kommandozeile mit „jitsu“.

Anschließend kann das Softwaresystem unter der folgenden Adresse erreicht werden: <http://pluto.jit.su>

9 Fazit

Zusammenfassend lässt sich sagen, dass die eingesetzten Technologien, WebSockets und WebRTC, in Form von Socket.io- und PeerJS-Bibliotheken, zur Lösung der in Kapitel 3 formulierten Problemstellung geführt haben. Leider werden die eingesetzten Standards noch nicht von allen Webbrowsern vollständig unterstützt. Eine schnellere Implementierung dieser Standards durch Browserhersteller ist wünschenswert und würde zu einer einheitlichen Basis und bei Anwendungsfällen die bisher nur nativen Applikationen vorbehalten waren zu besseren Browseranwendungen führen.

Im Verlauf der Entwicklung des Softwaresystems hat sich gezeigt, dass der Einsatz von WebSockets für zeitlich unkritische Applikationen praktikabel ist. Eine Verwendung in zeitkritischen Anwendungen, wie für die Steuerung eines Rennspiels, ist hingegen suboptimal. Allein durch den Transport der Steuerungsbefehle zum Server und anschließend zur Spieleanwendung entsteht eine Latenz, die in manchen Fällen für ein Rennspiel nicht vertretbar ist. Diese ist nur bei einer Paketumlaufzeit von weniger als 32ms im Softwaresystem akzeptabel.

Aus diesem Grund wurde eine Verbesserung der Implementierung mit WebRTC vorgenommen, welche keinen Server zur Weiterleitung der Kommunikation erfordert, sondern eine direkte Verbindung zwischen Webbrowsern herstellt. Die fehlende Verbreitung der Unterstützung des WebRTC-Standards in mobilen Webbrowsern macht eine Lösung, die nur auf WebRTC basiert, nicht erstrebenswert. Eine Variante, in der beide Technologien Anwendung finden, wurde deshalb ausgewählt.

Durch die Bearbeitung dieser Abschlussarbeit haben sich verschiedene Möglichkeiten der Optimierung des Softwaresystems ergeben. Im Bereich der Gerätekopplung sollte die Sitzungsnummernvergabe durch eine Interaktion mit dem Benutzer ersetzt werden und nicht wie bisher von der Zeit abgeleitet werden. Ein weiterer wünschenswerter Änderungspunkt ist das einheitliche Erkennen der verfügbaren Browserfunktion. Hilfreich für die Übertragung könnte

eine Optimierung der JSON-Datenstruktur sein, mit der man Bandbreite beim Betrieb der Applikation sparen würde.

Es bleibt zu hoffen, dass die Umsetzung der geplanten Webstandards schneller von den Herstellern der Webbrowser implementiert werden, um bessere Applikationen auf dem neuesten technischen Stand im Browser zu ermöglichen.

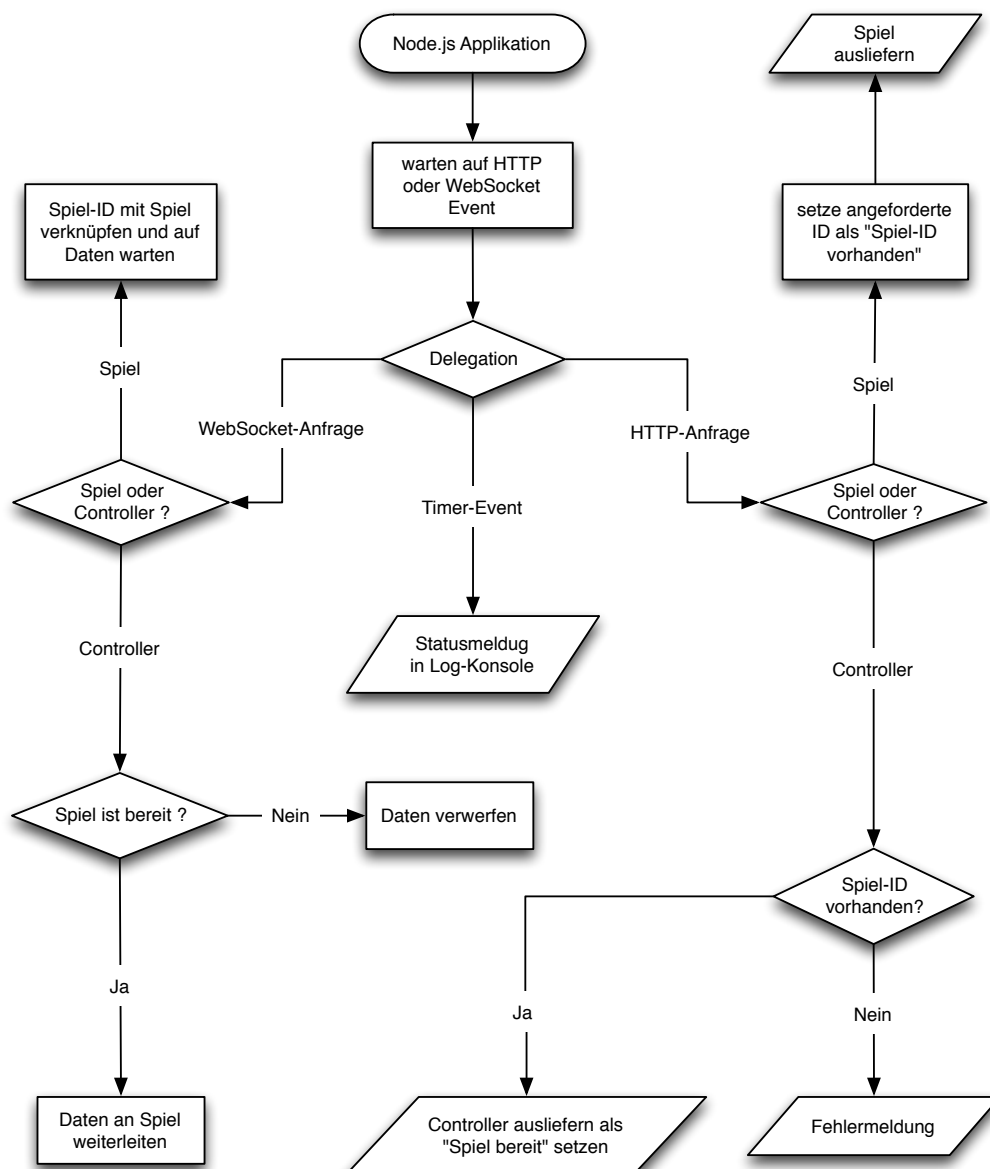
Abbildungsverzeichnis

Abbildung: Flussdiagramm

Diese Abbildung zeigt den Eventfluss der Serverapplikation bei jedem eingehenden Event und den dazugehörigen Verlaufsmöglichkeiten. Um die exakte Konfiguration des Servers zu erfahren, sehen sie bitte den Quellcode ein.

Quellcode: „/server.js“ im Wurzelverzeichnis des Softwaresystems.

Verwendet in: Kapitel 5.3, eigene Grafik, OmniGraffle, 20 August 2013



Appendix

Appendix 1: Vollständige „package.json“ des Softwaresystems.

Quelle: „./package.json“, selbst erstellt.

```
{
  "name": "pluto",
  "description": "websocket remote controller for webapplications",
  "version": "0.1.0",
  "author": {
    "name": "Eugen Pirogoff",
    "email": "eugenpirogoff@me.com",
    "url": "http://www.eugenpirogoff.de/"
  },
  "license": "MIT",
  "repository": {
    "type": "git",
    "url": "https://github.com/eugenpirogoff/pluto.git"
  },
  "dependencies": {
    "express.io": ">= 1.x.x",
    "bower": ">= 0.3.x"
  },
  "engines": {
    "node": ">= 0.10.x",
    "npm": ">= 1.0.x"
  },
  "subdomain": "pluto",
  "domains": [
    "pluto.jit.su",
    "www.pluto.jit.su/"
  ],
  "bundleDependencies": [],
  "private": true,
  "scripts": {
    "start": "node server.js"
  },
  "main": "server.js"
}
```

Appendix 2: Vollständige „bower.json“ des Softwaresystems.

Quelle: „./bower.json“, selbst erstellt.

```
{
  "name": "pluto",
  "version": "0.1.0",
  "ignore": [
    "**/*.txt",
    "**/*.*",
    "node_modules"
  ],
  "dependencies": {
    "jquery": "latest",
    "peerjs": "~0.2.5",
    "bootstrap": "latest",
    "components-font-awesome": "latest"
  },
  "private": true
}
```


Quellenverzeichnis

- Thibaut Despoulain, Entwickler von HexGL
<http://bkcore.com>
- Paketmanager für FrontEnd-JavaScript-Bibliotheken
<http://bower.io>
- Auflistung von unterstützten Technologien pro Browser
<http://caniuse.com>
- JavaScript-Engine von Google
<http://code.google.com/p/v8/>
- Git Versionskontrollsystem
<http://git-scm.com>
- Projektseite des HexLG-Rennspiels
<http://github.com/BKcore/HexGL>
- Browsertest
<http://html5test.com>
- Jerome Etienne, Entwickler von VirtualJoystick.js
<http://jetienne.com>
- Platform as a Service Anbieter
<http://nodejitsu.com>
- Erklärung zu package.json
<http://package.json.nodejitsu.com>
- Browserfingerprint
<http://panopticklick.eff.org>
- Framework für WebRTC
<http://peerjs.com>
- Semantische Versionierung
<http://semver.org>
- Framework für WebSocket
<http://socket.io>
- Projektseite von libev
<http://software.schmorp.de/pkg/libev.html>
- WebSocket RFC
<http://tools.ietf.org/html/rfc6455> -
- Informationen zu Node.js und Internas
<http://www.fh-wedel.de/~si/seminare/ss11/Ausarbeitung/08.nodejs/konzepte.html>
- JSON Projektseite
<http://www.json.org>
- WebGL Spezifikationen
<http://www.khronos.org/registry/webgl/specs/latest/>
- Vorlesungsfolien von Prof. Sascha Alda zum Thema REST
http://www.sascha-alda.de/lehre/soa/vorlesung/Kapitel_8_-_REST.pdf
- W3C WebRTC Standardisierung
<http://www.w3.org/TR/webrtc/>
- W3C WebSockets Standardisierung
<http://www.w3.org/TR/websockets/>