

ia3.Netz&Data Projekt: Multiuser Chat

FH Augsburg | Interaktive Medien 3 | ia3.Data und ia3.Netz | Prof. Dr. Nik Klever

Repository online unter: <http://code.google.com/p/ia3data-chat/>

Ein Projekt von:

Simon Heimler
Matrikelnummer: 924317



Emanuel Kössel
Matrikelnummer: 924338



Inhaltsverzeichnis:

- [Inhaltsverzeichnis:](#)
- [Projektbeschreibung:](#)
- [Client:](#)
- [Server:](#)
- [Wahl des Themas:](#)
- [Wahl der Technologie:](#)
- [Aufteilung:](#)
- [Installation:](#)
 - [Eigener Rechner](#)
 - [VirtualBox Appliance](#)
- [Dokumentation Server \(Simon\)](#)
 - [Wahl der Technologie & Motivation](#)
 - [Funktionsweise des Servers](#)
 - [Der Server dient zwei Funktionen:](#)
 - [Typischer Ablauf zwischen Client und Server:](#)
 - [Modularisierung & Wiederverwendbarkeit:](#)
 - [Testen des Servers](#)
 - [Platform as a Service \(PaaS\) für Node.js Anwendungen](#)
- [Dokumentation Client \(Emanuel\)](#)
- [Ausblick:](#)
- [Verwendete Literatur und Tutorials:](#)
- [Sourcecode Quellenhinweise:](#)
- [Erstellungserklärung](#)

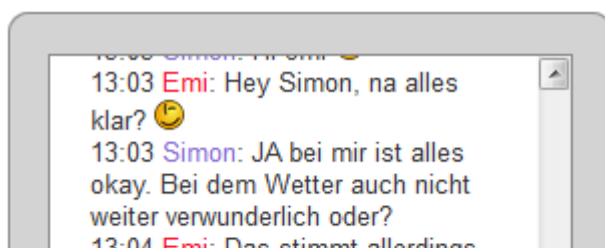
Projektbeschreibung:

Es handelt sich um eine webbasierte Chatanwendung welche es Usern erlaubt plattformunabhängig mit anderen Usern in Kontakt zu treten.

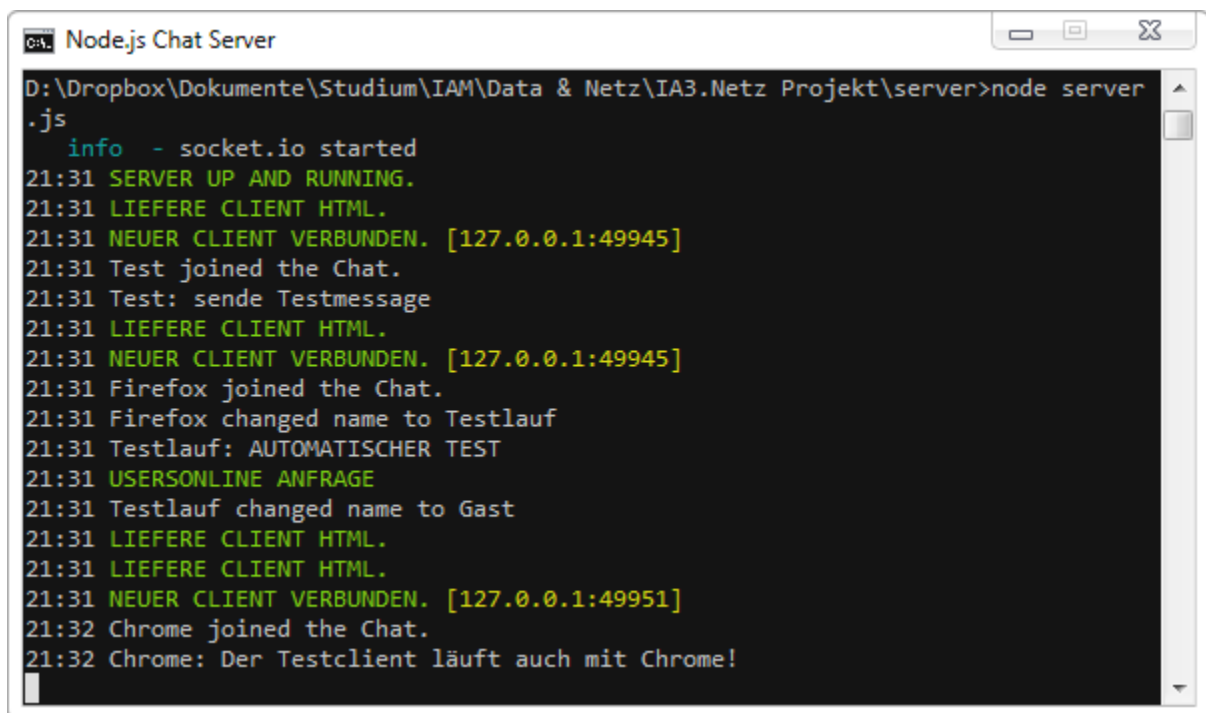
Dabei wird der User am Anfang jeder Session aufgefordert seinen Usernamen zu setzen, und hat dann im Nachhinein noch die Möglichkeit diesen abzuändern. Weiterhin ist es möglich Nachrichten zu schreiben und zu empfangen. Der vorhergehende Chatverlauf wird dem neu angemeldeten User als History angezeigt.

Jeder Username hat eine einzigartige Zufallsfarbe welche es erlaubt, die User im Laufe des Gesprächs schnell voneinander zu unterscheiden.

Client:



Server:



```
Node.js Chat Server
D:\Dropbox\Dokumente\Studium\IAM\Data & Netz\IA3.Netz Projekt\server>node server
.js
  info - socket.io started
21:31 SERVER UP AND RUNNING.
21:31 LIEFERE CLIENT HTML.
21:31 NEUER CLIENT VERBUNDEN. [127.0.0.1:49945]
21:31 Test joined the Chat.
21:31 Test: sende Testmessage
21:31 LIEFERE CLIENT HTML.
21:31 NEUER CLIENT VERBUNDEN. [127.0.0.1:49945]
21:31 Firefox joined the Chat.
21:31 Firefox changed name to Testlauf
21:31 Testlauf: AUTOMATISCHER TEST
21:31 USERSONLINE ANFRAGE
21:31 Testlauf changed name to Gast
21:31 LIEFERE CLIENT HTML.
21:31 LIEFERE CLIENT HTML.
21:31 NEUER CLIENT VERBUNDEN. [127.0.0.1:49951]
21:32 Chrome joined the Chat.
21:32 Chrome: Der Testclient läuft auch mit Chrome!
```

Wahl des Themas:

Beim Durchforsten der vorgegebenen Beispielthemen sind wir auf den "Live Chat" gestoßen und fanden das Thema auf Anhieb interessant. Zum einen weil sich ein webbasierter Chat, aus gesellschaftlicher Hinsicht, als Schnittstelle zwischen verschiedenen Menschen sieht; mit dem Ziel sich weltweit, live und in unserem Fall plattformunabhängig unterhalten zu können. Und zum Anderen weil verschiedenste Programmiertechniken zur Verwendung kommen. So haben wir die client- sowie serverseitige Programmierung, den Datenaustausch zwischen Client und Server und die Behandlung der bisherigen Chat-History als sehr reizvoll und im Zweierteam als gut aufteilbar empfunden.

Wahl der Technologie:

Wir hatten zuerst vor das Projekt mit dem Web2Py Framework umzusetzen.

Nach der ersten Recherche kamen wir zu dem Schluss dass ein Chat serverseitig am besten mittels asynchroner Programmierung umgesetzt werden kann. Zudem eignet sich die "Comet"¹ Architektur, wo der Server auch den Client kontaktieren kann - nicht nur umgekehrt - besser für eine Chatanwendung. Wir haben uns hier für "**Node.js**" (<http://nodejs.org/>) entschieden, das einen sehr modernen Ansatz verfolgt. Node.js hat aktuell noch kein 1.0 Release, wir werden also mit einer Technologie arbeiten die sich in rapider Entwicklung befindet und noch größere Veränderungen durchmachen könnte.

Dazu wählte ich das Modul "**socket.io**" (<http://socket.io/>) aus um den Datentransport zwischen Server und Client herzustellen.

Clientseitig entschieden wir uns für eine HTML/CSS & Javascript Umsetzung des Client. Als Framework kam **jQuery** (<http://jquery.com/>) zum Einsatz.

Um gleichzeitig an unserem Projekt zu arbeiten haben wir uns für das Versionskontrollensystem **SVN** mit einem bei Google Code gehostetem Repository entschieden: <http://code.google.com/p/ia3data-chat/>

Aufteilung:

Emanuel Kössel: Client Design, Client-Programmierung

Simon Heimler: Serverprogrammierung, Konzeption

Installation:

Eigener Rechner

1. Node.js installieren: <http://nodejs.org/#download>
2. Eventuell Computer neustarten, falls der Befehl "node" nicht in der Konsole verfügbar ist.
3. In das Projekt/server Verzeichnis wechseln
4. In der Konsole den Server mit "node main.js" starten.
5. Jetzt kann der Client im Projekt/client Verzeichnis gestartet werden.
Lokal sollte der *Firefox Browser* verwendet werden!
6. Eventuell muss die URL und der Port sowohl im Client als auch Server angepasst werden.
Anmerkung: Es gibt eine Same-Origin-Policy die den Datentransfer zwischen Client und Server blockieren kann. Siehe auch <http://de.wikipedia.org/wiki/Same-Origin-Policy>

VirtualBox Appliance

1. Appliance von DVD entpacken.
2. In VirtualBox importieren
3. Appliance starten
4. Auf Desktop das Bash Script START Server ausführen
5. Auf Desktop den URL Link START Client ausführen
6. Username ist "demo" und Passwort ebenfalls "demo"

Dokumentation Server (Simon)

Wahl der Technologie & Motivation

Der Server ist mithilfe des "**Node.js**" **Framework**² umgesetzt. Serverseitig wird also mit JavaScript und möglichst ausschließlich asynchron programmiert.

Node.js besteht aus einem auf dem Server laufenden Google V8 Javascript Interpreter inklusive einem integrierten Framework, dass das programmieren von Webservern erleichtert. Es ist eine relativ neue Technologie, die aber für starkes Aufsehen gesorgt hat und mittlerweile auch von den größten Firmen verwendet wird.

¹ Flanagan 2011 #1: 515–521

² <http://nodejs.org/>

Technisch ist Node.js speziell für Webanwendungen im Echtzeitbereich ausgelegt und damit optimal für einen Realtime-Chat. Ein Grund für meine Entscheidung zu Node.js ist, dass ich gerne eine "Cutting-Edge" Technologie ausprobieren wollte.

Bei dem **Transport-Protokoll** war die Überlegung folgende: Am besten eignet sich dafür die neue HTML5 Technologie "Websockets³" die aber noch in der aktiven Entwicklung steht und leider von vielen Browsern noch nicht unterstützt wird. (Und noch dazu der Standard dazu nicht definitiv festgegossen ist). Um Kompatibilität zu gewährleisten wählte ich das Node.js Modul "**socket.io**"⁴ aus um den Datentransport zwischen Server und Client herzustellen.

Socket.IO ist also ein API für Client <-> Server Kommunikation, das es ermöglicht verschiedene Protokolle zu verwenden.

Funktionsweise des Servers

Aller Code des Servers findet sich in der server.js im Projekt/server Verzeichnis.
Der Sourcecode ist im JSDoc Format kommentiert.

Wichtig zu beachten ist, dass der Server asynchron programmiert ist: Der Server wird einmalig gestartet und bleibt in der Schleife. Er reagiert in Echtzeit auf die Clientanfragen und kann daher auch (z.B.) die History im Arbeitsspeicher halten. Wenn der Server beendet wird, ist auch keine Anfrage von aussen mehr möglich. Im Produktiveinsatz muss also dafür gesorgt werden, dass er - wenn nötig - automatisch neugestartet wird.

Um die Chance eines Absturz des Servers zu vermindern (auch durch unsachgemäße Bedienung des Clients) habe ich fast alle Verarbeitungsprozesse mit try und catch abgesichert.

In der Server Konsole tauchen eventuelle Fehler und Infomeldungen auf.

Der Server dient zwei Funktionen:

- Er liefert den Chat Client an den User aus und entsprechend des Browsers die clientseitige Transportlogik. (Websocket, Flashsocket, AJAX...)
- Er nimmt vom Client verschiedene Anfragentypen mit entsprechendem Datensatz entgegen (nur Text) und sendet nach Verarbeitung an alle oder einzelne verbundene Clienten verschiedene Typen von Datensätzen zurück (meist im JSON Format).

Typischer Ablauf zwischen Client und Server:

1. Client sendt eine Socket.IO Message (unterschiedlichen Typs) zum Server
2. Server hat für jeden Typ eine asynchron laufende Funktion, die den Messageinhalt entsprechend des Typs verarbeitet, meist zu dem Endformat eines JSON Strings.
3. Server sendet eine Socket.IO Message des selben Typs zu allen Clienten oder dem speziell anfragenden Client.
4. Client dekodiert die Message und verarbeitet sie zu HTML, das angezeigt wird.

Die Funktionen und Variablen sind im Sourcecode dokumentiert.

Modularisierung & Wiederverwendbarkeit:

Der Server ist hauptsächlich in der server.js umgesetzt. Um einige Hilfsfunktionen auszulagern und die Möglichkeit einer Modularisierung mit Node.js anzudeuten habe ich diese in utilities.js ausgelagert. Damit könnten diese Hilfsfunktionen sehr einfach auch für andere Node.js Projekte verwendet werden.

Da der Server in JavaScript programmiert ist, ist auch eine direkte Wiederverwendbarkeit des Codes in der clientseitigen Programmierung möglich!

Testen des Servers

Zum testen des Servers habe ich einen rohen Testclienten geschrieben (/server/client/testclient.htm),

³ <https://developer.mozilla.org/en/WebSockets>

⁴ <http://www.socket.io/>

der die JSON Ausgaben vom Server unformatiert ausgibt. So kann man genau sehen was der Server exakt zurückgibt. Es gibt auch einen Button "Testlauf" der die wichtigsten bisher implementierten Funktionen automatisch durchführt. Das erlaubt einen schnellen Test ob der Server im groben lauffähig ist.

Auch eingebaut ist ein rudimentärer HTTP Server, der die Testclient HTML ausliefert wenn man die Server URL ansteuert (<http://localhost:8000/>). So lässt sich die Same Origin Policy zu Testzwecken einfach umgehen.

Platform as a Service (PaaS) für Node.js Anwendungen

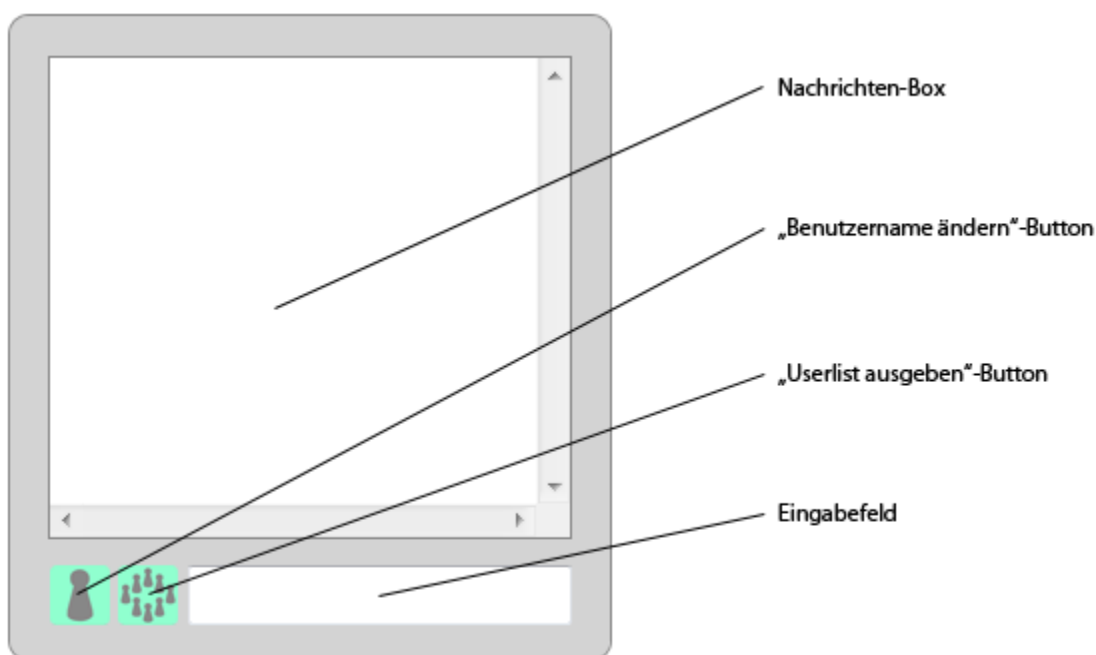
Um den Server nicht nur lokal testen zu können, habe ich nach einer kostenlosen möglichkeit recherchiert Node.js Anwendungen zu hosten. Ich habe mich für den Open Source PaaS Service Nodester.com (<http://nodester.com>) entschieden. Die Anwendungen und die Sandbox in der sie läuft werden über ein API verwaltet. Die Anwendung selbst wird mittels GIT (Ein Versionkontrollensystem) auf den Server eingesetzt "Deploy Vorgang".

Hier ein Testaccount, wo der Server und Testclient online geliefert werden:

<http://neuertest.nodester.com/>

Dokumentation Client (Emanuel)

Der Aufbau des Client ist rein in HTML/CSS und in Javascript programmiert. Im Client selbst ist das socket.io Client Library implementiert, das sich um das Transportprotokoll kümmert. Nachrichten in jeglicher Form zwischen Client und Server sowie umgekehrt werden per JSON übertragen.



Arbeitsweise Client:

1. User betritt den Chat
2. Sofort wird dem Client die bisherige History (falls vorhanden) geschickt und der User nach seinem Namen gefragt
3. Der User gibt seinen Namen ein und bestätigt mit der Entertaste. Gibt dieser keinen Namen ein und bestätigt das leere Eingabefeld dennoch mit der Entertaste so wird ihm ein beliebiger Username (Gast #) zugewiesen. Nach setzen des Namens wird allen anderen Nutzern mitgeteilt dass der neuer Nutzer soeben den Chatraum betreten hat.
4. Nun hat der neue User die Möglichkeit den Chatpartnern durch Eingabe im Textfeld

und Bestätigung durch die Entertaste Nachrichten zu senden und auch Nachrichten zu Empfangen.

5. Neben dem Senden und Empfangen von Nachrichten bietet der Chat dem User noch die Möglichkeit durch Klick auf einen der jeweils neben dem Eingabefeld liegenden Buttons

seinen Usernamen im Nachhinein zu ändern



oder sich die Anzahl und Namen der im



Chat befindlichen Nutzer anzeigen zu lassen.

6. Nach Beendigung der Chat-Session des jeweiligen Nutzers durch Schließen des Browsers wird den anderen Nutzern eine Nachricht über das Verlassens des jeweiligen Chatpartners geschickt.

Ausblick:

Weitere denkbare Features wären:

- Die Möglichkeit den Chat zu moderieren. Man müsste sich beim Server authentifizieren. Dann kann man uid (UserID's) und mid (MessageID's) verwenden um einzelne Messages, aber auch User zu moderieren. Diese Möglichkeit ist vom Server schon prinzipiell gegeben, es müssten allerdings weitere Transport-Typen sowohl server- als auch clientseitig geschrieben werden.
- Den Chat einfach in fremde Seiten einbindbar machen. (z.B. als jQuery Plugin dass auf einer beliebigen (leeren) DIV Box aufgerufen werden kann und Parameter wie Server URL und Port entgegennimmt)

Verwendete Literatur und Tutorials:

- Manuel Kiessling: The Node Beginner Book. Online verfügbar unter <http://www.nodebeginner.org>, zuletzt geprüft am 19.03.2012.
- Flanagan, David (2011): JavaScript. The definitive guide. 6. Aufl. Beijing ;, Sebastopol, CA: O'Reilly.
- Lawson, Bruce; Sharp, Remy (op. 2011): Introducing HTML 5. Berkeley, CA: New Riders.
- Koch, Stefan (2011): JavaScript. Einführung, Programmierung und Referenz. 6. Aufl. Heidelberg, Neckar: dpunkt.
- Jan Van Ryswyck: Taking Baby Steps with Node.js – WebSockets. Online verfügbar unter <http://elegantcode.com/2011/05/04/taking-baby-steps-with-node-js-websockets/>, zuletzt geprüft am 19.03.2012.
- Martin Sik: Node.js & WebSocket - Simple chat tutorial. Online verfügbar unter <http://www.martinsikora.com/nodejs-and-websocket-simple-chat-tutorial>, zuletzt geprüft am 19.03.2012.
- Mikito Takada: 12. Comet and Socket.io deployment. Online verfügbar unter <http://book.mixu.net/ch13.html>, zuletzt geprüft am 19.03.2012.

Sourcecode Quellenhinweise:

Im Source-Code sollten alle wichtigen Quellen genannt sein!

Erstellungserklärung

Hiermit erkläre ich, dass ich die vorgelegte Arbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel verwendet, keine Urheberrechtsschutz-Verletzungen begangen sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Name, Datum, Unterschrift

Name, Datum, Unterschrift