SWDICTIONARY

EIN FREIES COMMUNITY-WÖRTERBUCH

DISTRIBUTED COMPUTING

SWD ’16 – SS 2018

TOMEK, WALLNER, WRESNIK

1. CONTENT / Inhalt

[2](#_gjdgxs) Einleitung 3

[2.1](#_30j0zll) Ausgangssituation 3

[2.2](#_1fob9te) Projektübersicht / Übersicht 3

[3](#_3znysh7) Aufgabenstellung 3

[3.1](#_2et92p0) Client UI 3

[3.2](#_tyjcwt) Messaging Queuing / Messaging Queue(s) 3

[3.3](#_3dy6vkm) Worker / Consumer 3

[3.4](#_1t3h5sf) Web Socket Communication 4

[4](#_4d34og8) Setup / Installation 4

[4.1](#_2s8eyo1) Wie bekommt man das Projekt zum Laufen 4

[5](#_17dp8vu) Anleitung / BEdienung 4

[5.1](#_3rdcrjn) Client UI 4

[5.2](#_26in1rg) Messaging Queuing / Messaging Queue(s) 4

[5.3](#_lnxbz9) Worker / Consumer 5

[5.4](#_35nkun2) Web Socket Communication 5

[6](#_1ksv4uv) Summary / Zusammenfassung / Resüme 5

[6.1](#_44sinio) Output / Ergebnis 5

[6.2](#_2jxsxqh) Known Problems / Bekannte Probleme 5

[6.3](#_z337ya) Outlook / Ausblick / weitere Möglichkeiten 5

[6.4](#_3j2qqm3) Opinion / Persönliche Meinung / Erkenntnisse 5

[7](#_1y810tw) Anhang 5

[7.1](#_4i7ojhp) Code Repository / Sources 5

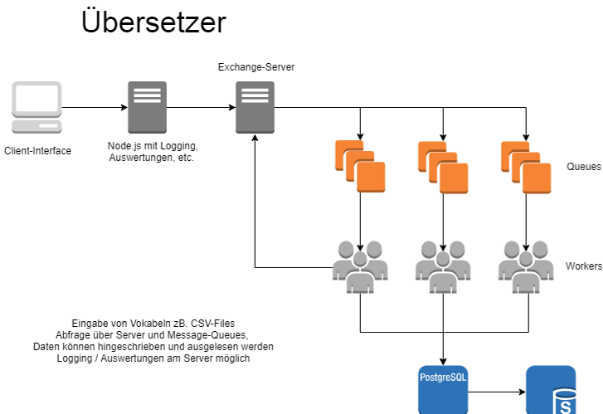
[7.2](#_2xcytpi) Links 5

# Einleitung

## Ausgangssituation

Für das Studienfach Distributed Computing bei Hr. Michael Ulm soll eine Projektarbeit erstellt werden, die den Semesterstoff des 4. Semesters des Studiengangs „Software Design (berufsbegleitende Vertiefung von Internettechnik“) umfasst und das Gelernte bei den Studenten praktisch vertieft.

## Projektübersicht / Übersicht

Erstentwurf von Mai 2018:

# Aufgabenstellung

## Client UI

–zB. Web Application mit Ein- & Ausgabe

## Messaging Queuing / Messaging Queue(s)

–Producer & Consumers

## Worker / Consumer

–Mindestens 3 verschiedene Programmiersprachen, zB. Python, Java, PHP

– Mindestens 2 verschiedene Worker, zB. bruteforce, dictionary

– Mindestens 2 verschiedene Daten (Quellen), zB. Datenbank, Dateisystem Logging

## Web Socket Communication

–Mehrfache Verbindungen von User Interface oder Workers

# Setup / Installation

## Wie bekommt man das Projekt zum Laufen

* Zuerst, kontrolliere durch Eingabe in der Kommandozeile (cmd.exe) ob und welche Version von node.js installiert ist:  
  where node  
  node --version  
  Wenn deine Version unter v7.6 ist (z.B. v6.11.3), dann musst du sie aktualisieren. Folge den Schritten: [update node version - on Windows](https://github.com/wresnikg16/Translate_DistComp/blob/master/documentation/updateNodeVersion.md).
* npm install lädt alle notwendigen Abhängigkeiten und Bibliotheken
* wechsle in den **workers** Ordner und starte **addworker.js** um die Datenbank zu initialisieren und automatisch ihre Tabelle zu kreieren.  
  cd .\workers  
  node addworker.js

1. wechsle in den **server** Ordner und starte **server.js** um das Front End zum Laufen zu bringen  
   cd ..\server  
   node server.js
2. Öffne deinen Browser auf <http://localhost:8081>
3. im add/search/delete Bereich der Webseite können Wörter hinzugefügt, gesucht und aus dem Wörterbuch gelöscht werden.

# Anleitung / BEdienung

## Client UI

Das GUI (graphical user interface) der vorgestellten Web-Applikation ist schlicht und möglichst selbsterklären aufgebaut. Im linken oberen Bereich befindet sich die Navigation zu den angebotenen Ressourcen. Klickt man auf einen der Buttons so wird man auf die Subseiten /sites/add.html, /sites/search.html und /sites/remove.html weitergeleitet, die die jeweilige Funktionalität des Wörterbuchs anbieten.

## Messaging Queuing / Messaging Queue(s)

Der Message Broker RabbitMQ wurde verwendet um die Funktionalität von Message Queues zu implementieren. Er unterstützt zahlreiche verschiedene Programmiersprache und bietet auch ein Web-Management-Tool an, das unter <http://localhost:15672/> mittels Username: „guest“ und Passwort: „guest“ aufgerufen werden kann. Jede Ressource bzw. die dort ausgeführte Javascript Methode schreibt Daten in eine eigene Queue. Diese Queue buffert die Daten, bis sie vom jeweils zuständigen Worker abgeholt und verarbeitet werden.

## Worker / Consumer

Um die drei angebotenen Funktionalitäten (hinzufügen, suchen und löschen) auszuführen, wird jede Queue von einem spezifischen Worker abgefragt und die Daten daraus verarbeitet.

## Web Socket Communication

Der Websocket…

# Summary / Zusammenfassung / Resüme

## Output / Ergebnis

### Frontend Webserver server.js

Der NodeJS Server server.js stellt im Projekt den Frontend Webserver dar, dieser stellt dem User die Benutzeroberfläche zur Verfügung. Der Server nutzt folgende Struktur diese anschließend kurz beschrieben wird.

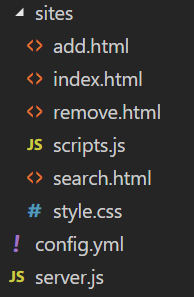


Abbildung 1 node.js Server Files

Unter dem Ordner „sites“ sind die einzelnen HTML Files zu finden. Für jede einzelne Funktion gibt es eine eigene HTML Seite, und natürlich die „index.html“ als Startseite. Die Anforderung jeder Seite wird vom NodeJS Server übernommen, über die eingegebene Url kommt wird das die gewünschte Seite an den Client zurückgegeben.

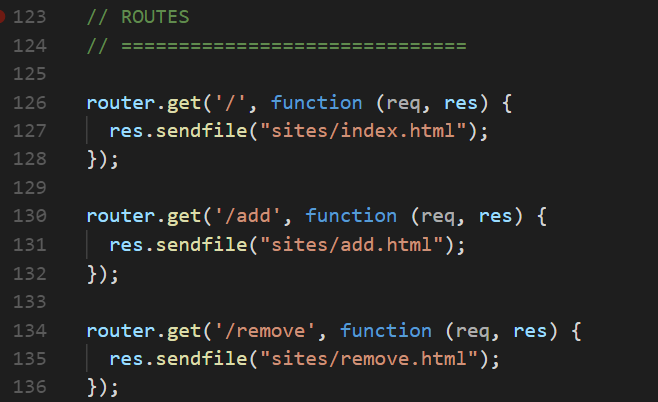


Abbildung 2 server.js routing

Für das Ausführen der Funktionen Add, Find und Delete gibt es ebenfalls ein eigenes routing. Sobald man die Url aufruft werden die Informationen an die Queues geschickt, wo sie anschließend abgearbeitet werden von den jeweiligen worker.

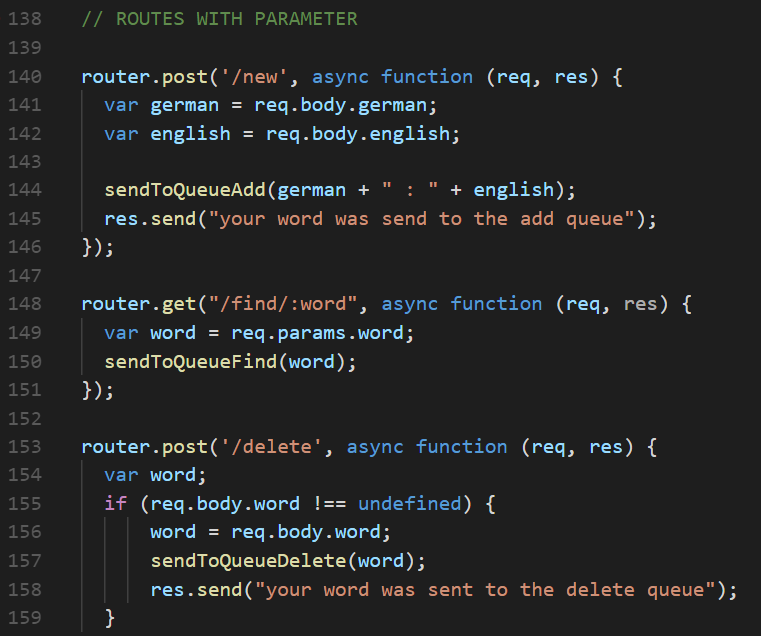


Abbildung 3 server.js function routing

Der NodeJS Server selbst wird ganz zum Schluss gestartet (Port 8081), dieser liest auch die Konfiguration aus der „config.yml“ Datei aus, was die Möglichkeit gibt die Server Einstellungen ohne Code Anpassung zu verändern.



Abbildung 4 server.js server start

### Messaging Queing RabbitMQ

Durch die Einbindung von der Library „amqplib/callback\_api“ im NodeJS Server ist es möglich Nachrichten an die Queue zu übermitteln. Es wird für jede Funktion eine eigene Queue verwendet die in der Funktion definiert wird.

* Find 🡪 findQueue
* Delete 🡪 deleteQueue
* Add 🡪 addQueue

Sollte es die Queue nicht geben wird diese vom Server angelegt.

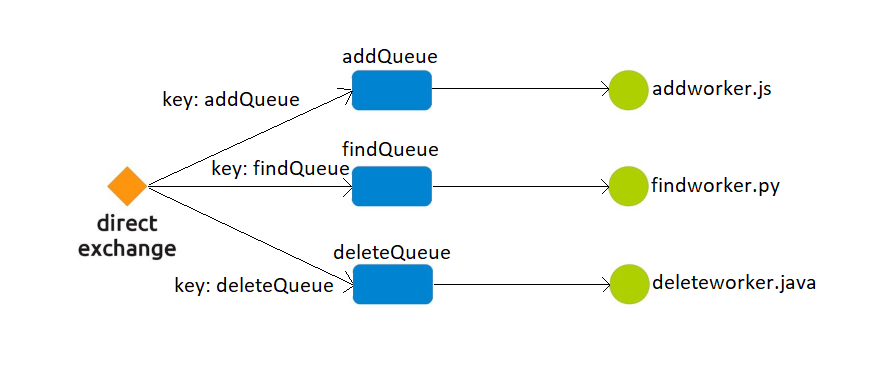


Abbildung 5 Queue Grafik

Folgendes Code Snippet zeigt das Senden an die „addQueue“.

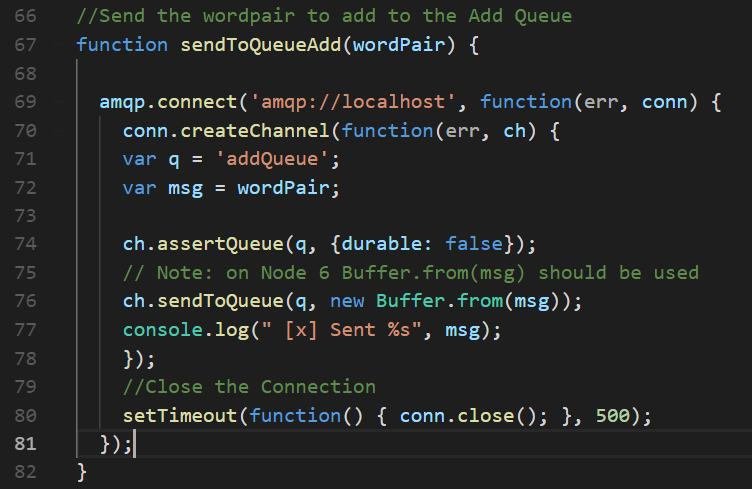


Abbildung 6 send to Queue Beispiel

### Websocket Server server.js

Um von den Workern eine Rückmeldung zu bekommen, wird ebenfalls mit dem Webserver ein Websocket Server auf Port 8082 gestartet. Die Konfiguration wird dabei gleich am Anfang im server.js vorgenommen.

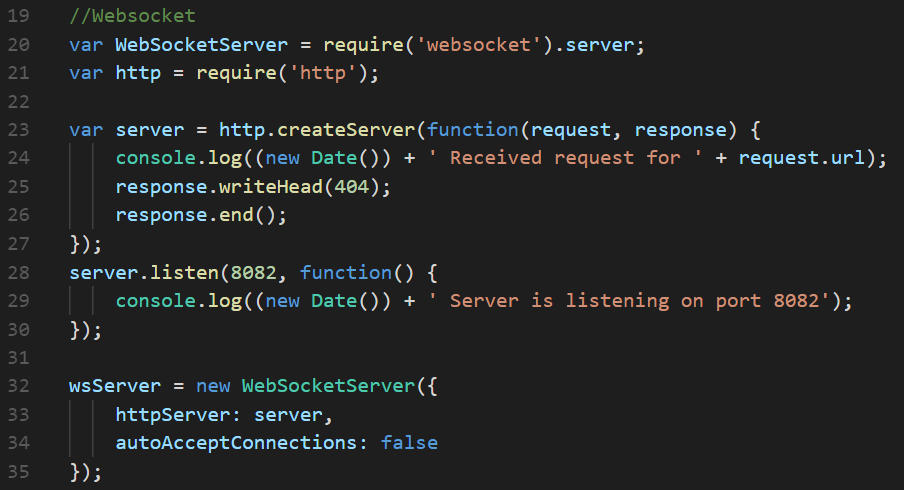


Abbildung 7 Websocket Server Konfiguration

Die EventHandler für den Websocket Server sind wie folgt definiert, die Rückmeldung vom Worker wird in der Konsole ausgegeben und ebenfalls wird dem Worker zurückgemeldet das seine Meldung angekommen ist.

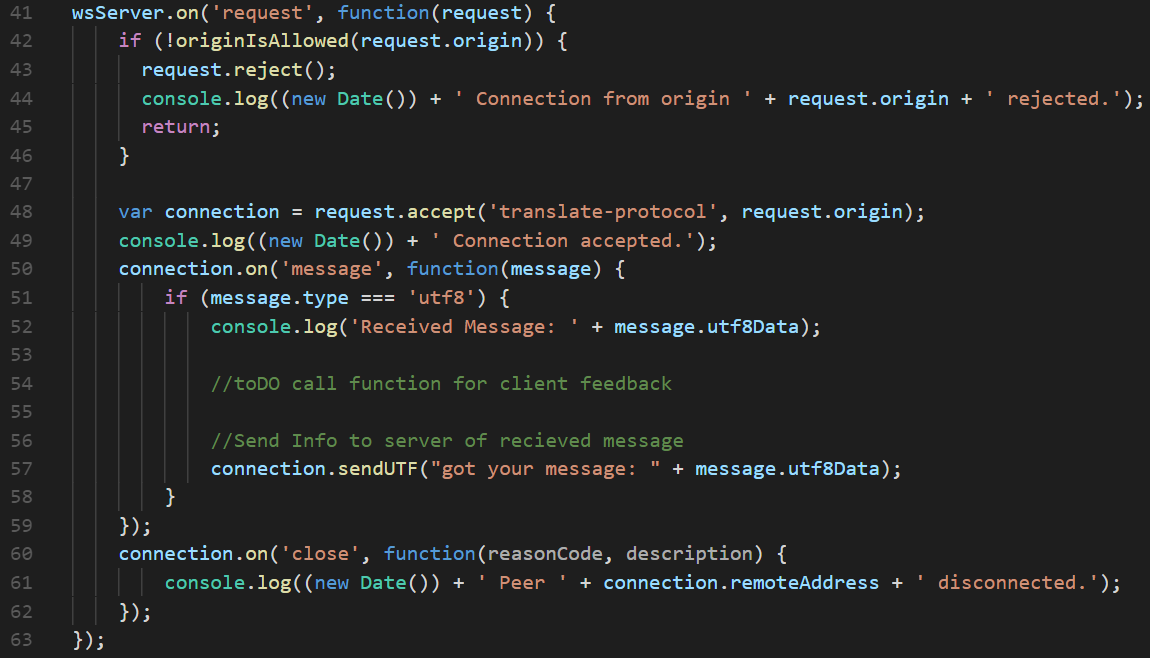


Abbildung 8 Websocket Eventhandler

### Workers

Es wurden 3 Worker erstellt um die Funktionen Find, Delete und Add abzubilden. Die Add Funktion wird von einem Javascript Worker erledigt, die Find von einem Python Worker und die Delete Funktion von einem Java Worker.

#### addworker.js

Dieser Worker ist für das hinzufügen in die Datenbank zuständig. Dieser Worker ist ebenfalls dafür da, dass die Datenbank bei erstmaliger Ausführung erstellt wird und auch die benötigte Tabelle angelegt wird. Die Funktion „init()“ legt die Datenbank an.

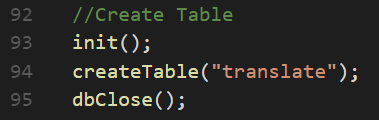


Abbildung 9 Create DB

Der Hauptteil des Workers ist das er ständig alle Messages aus der „addQueue“ holt und anschließend die Daten in die die Datenbank schreibt, dies macht er mittels entsprechendem Insert Statement. Ebenfalls hat er die Rückmeldung mittels Websocket implementiert. Folgendes Code Snippet zeigt das Abfragen aus der Queue und die Verarbeitungsschritte bis zum Rückmelden an des Websocket Server.

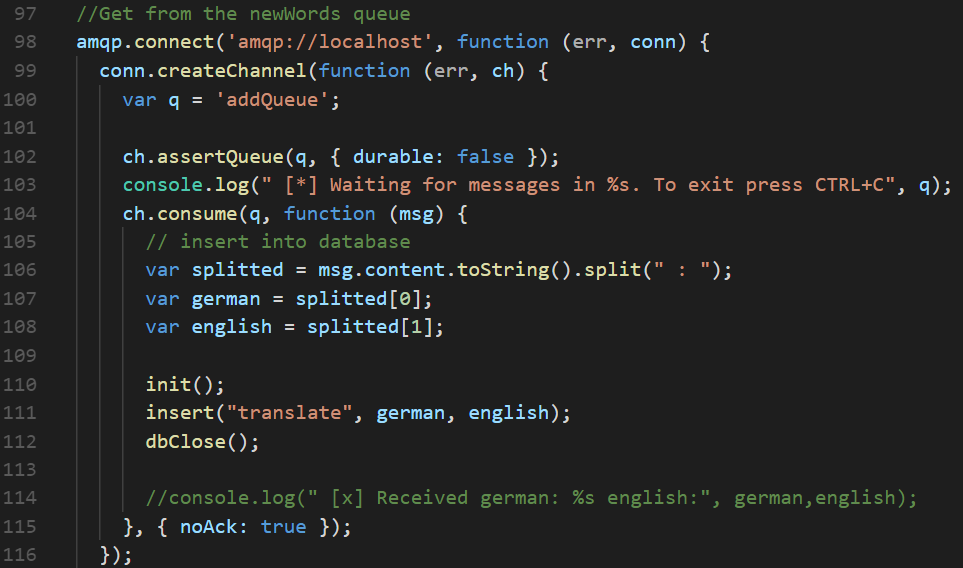


Abbildung 10 Consume addworker.js

Die Rückmeldung an den Websocket geschieht in der „insert()“ Funktion, dort wird je nach Erfolg oder Misserfolg beim Einfügen die entsprechende Nachricht an den Server gesendet.



Abbildung 11 insert and websocket message

Es folgt die Websocket Konfiguration von addworker.js, dabei ist es vor allem wichtig das über „client.connect()“ das richtige Protokoll angeben wird, wir haben das passend für unser Beispiel „translate-protocol“ genannt. Der Server weiß sonst nicht wie er damit umgehen soll, wenn er etwas nicht definiertes bekommt.



Abbildung 12 addworker.js Websocket config

#### findworkter.py

#### delelteworker.java

## Known Problems / Bekannte Probleme

### No Response zum Frontend

Aus zeitlichen Gründen war es leider nicht möglich die Rückmeldung von den Workern über Websocket am Client anzuzeigen, aber es wird am Server in der Konsole eine Meldung angezeigt.

## Outlook / Ausblick / weitere Möglichkeiten

## Opinion / Persönliche Meinung / Erkenntnisse

# Anhang

## Code Repository / Sources

GitRepo:

<https://github.com/wresnikg16/Translate_DistComp.git>

## Links

<https://www.rabbitmq.com/>