

# Chat und 4-gewinnt-Spiel mit Docker und Node.js

## Projektdokumentation Microservices

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

**Hanna Siegfried und Nahku Saidy**

12.06.2019

**Bearbeitungszeitraum**  
**Matrikelnummer, Kurs**  
**Ausbildungsfirma**  
**Betreuer**

nothing  
5946066,XXXXX, STG-TINF17-ITA  
Daimler AG, Stuttgart  
nothing

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>I</b>
<b>Abbildungsverzeichnis</b>	<b>II</b>
<b>Tabellenverzeichnis</b>	<b>III</b>
<b>Listings</b>	<b>IV</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Aufgabenstellung und Ziel der Projektarbeit . . . . .	1
1.2 Aufbau der Projektarbeit . . . . .	1
1.3 Wer hat was gemacht? . . . . .	1
<b>2 Grundlagen und Stand der Technik</b>	<b>2</b>
2.1 Node.JS . . . . .	2
2.2 Websockets . . . . .	2
2.3 Docker . . . . .	2
2.4 MongoDB . . . . .	2
2.5 MongoDB und Mongoose . . . . .	2
<b>3 Konzept</b>	<b>4</b>
3.1 Übersicht . . . . .	4
3.2 Login und Registrierung . . . . .	5
3.3 Spiel . . . . .	6
3.4 Nachrichtenfilterung . . . . .	6
3.5 Gleichzeitiges Spielen mehrerer Spieler . . . . .	6
<b>4 Implementierung</b>	<b>7</b>
4.1 Datenbank, Login und Registrierung . . . . .	7
4.2 Spiellogik . . . . .	8
4.3 Erkennung des Spielendes . . . . .	8
4.4 Docker Compose . . . . .	9
<b>5 Projektabschluss, Fazit &amp; Ausblick</b>	<b>10</b>
5.1 Fazit . . . . .	10
5.2 Ausblick . . . . .	10
<b>Literatur</b>	<b>i</b>



# Abkürzungsverzeichnis

**AABB**     Axis-Aligned Bounding Box

# Abbildungsverzeichnis

3.1	Übersicht der Anwendung . . . . .	5
4.1	Flussdiagramm des Loginprozesses . . . . .	8

# Tabellenverzeichnis

# Listings

# 1 Einleitung

## 1.1 Aufgabenstellung und Ziel der Projektarbeit

Die Aufgabe bestand in der Entwicklung einer Beispielanwendung unter Verwendung zweier Aspekte aus der Vorlesung. Wir haben uns für die Entwicklung eines 4-gewinnt-Spiels auf Basis der Chat-Anwendung entschieden. Dieses soll mittels Docker deploybar gemacht werden. Die Chat-Teilnehmer sollen mit einer Registrierungsmöglichkeit in einer MongoDB verwaltet werden, welche auch in einen Docker-Container verpackt werden soll. Die Entwicklung der Anwendung wird mit Node.js durchgeführt. -docker -mongo.db -docker-compose -node.js

## 1.2 Aufbau der Projektarbeit

Diese Projektarbeit gliedert sich in fünf Kapitel. Im ersten Kapitel wird eine Einleitung in die Aufgabenstellung und die geplante Anwendung gegeben. Das zweite Kapitel beinhaltet die notwendigen Grundlagen, auf denen die Anwendung basiert. Dabei werden die einzelnen Technologien und ihre Möglichkeiten kurz vorgestellt. Darauf folgend wird in Kapitel 3 das Konzept der Anwendung und ihrer Komponenten dargestellt. Dabei wird auch auf die Vernetzung zwischen zum Beispiel der Datenbank und dem Chat eingegangen. In Kapitel 4 werden dann einzelne Teile der Implementierung vorgestellt, die essenziell für die Funktion der Anwendung sind. Im letzten Kapitel wird noch einmal auf die Aufgabenstellung Rückbezug genommen und eine kritische Würdigung der erzielten Ergebnisse vorgenommen. Zusätzlich dazu wird ein Ausblick auf mögliche Weiterentwicklungen gegeben.

## 1.3 Wer hat was gemacht?



## 2 Grundlagen und Stand der Technik

### 2.1 Node.JS

### 2.2 Websockets

«««< HEAD Websockets sind die Grundlage für die Chat-Anwendung.

### 2.3 Docker

«««< HEAD Docker ist eine Software ... In dieser Projektarbeit wird Docker verwendet, um das einfache deployen der Anwendungen, unabhängig von den konkreten Servern zu ermöglichen. ===== Websockets sind die Grundlage für die Chat-Anwendung

### 2.4 MongoDB

In der Anwendung wird eine MongoDB zum Speichern der Zugangsdaten der Chat-Teilnehmer genutzt. ===== Docker

### 2.5 MongoDB und Mongoose

In der Anwendung wird eine MongoDB zum Speichern der Zugangsdaten der Chat-Teilnehmer genutzt. MongoDB ist eine NoSQL-Datenbank, dies bedeutet, dass diese nicht relational sind, sondern Dokumenten-orientiert (S. 3). Ein Dokument besteht aus einer Menge an Keys und Values. Dies bietet unter Anderem eine

besser Skalierbarkeit, da die Daten besser auf verschiedene Server aufgeteilt werden können(S. 4). Es gibt keine vorgegeben Schema, die eingehalten werden müssen. MongoDB speichert die Daten in BSON und die Rückgabeobjekte sind in JSON.

Mongoose stellt eine Objekt-Daten-Modellierungs-Bibliothek für MongoDB und Node.JS zur Verfügung. Mit Mongoose kann das Datenschema der Datenbank im Code in JSON definiert werden.

»»»> Bericht

# 3 Konzept

## 3.1 Anforderungen

-ermöglichen spielen von spiel mit 2 spielern -gleichzeitiges chatten möglich -  
spielsteine über knöpfe setzen

## 3.2 Übersicht

Die Anwendung besteht aus zwei verschiedenen Teilen, die beide mittels Docker deployt werden. Der eine Teil beinhaltet die Chat-Anwendung und die Spiellogik, der andere beinhaltet eine MongoDB, welche zum Speichern der Benutzer-Login-Informationen verwendet wird.

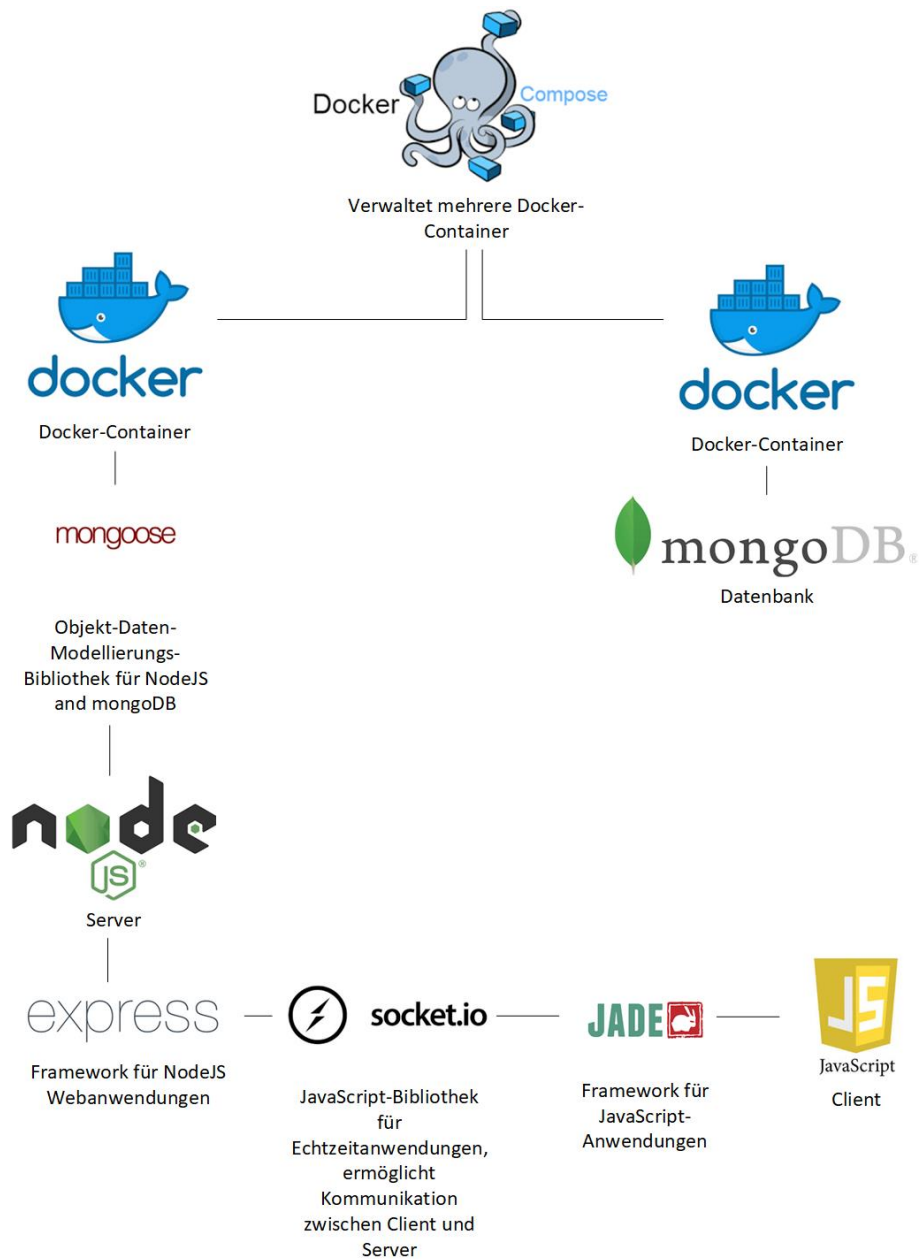


Abbildung 3.1: Übersicht der Anwendung

### 3.3 Login und Registrierung

Beim Login hat der User die Möglichkeit haben, sich zu registrieren oder sich direkt einzuloggen.

## 3.4 Spiel

## 3.5 Nachrichtenfilterung

Typischerweise werden bei einer Websocket-Anwendung allen Usern die eintreffenden Nachrichten angezeigt. Um dieses zu umgehen und nur den am Spiel beteiligten Spielern Spiel-relevante Nachrichten anzuzeigen, enthält die Nachricht die Namen der beiden am Spiel beteiligten Parteien. So kann der Client ermitteln, ob er am Spiel beteiligt ist und die Nachricht dann anzeigt oder nicht und die Nachricht ignoriert.

## 3.6 Gleichzeitiges Spielen mehrerer Spieler

Um es mehreren Spielern zu ermöglichen, gleichzeitig zu spielen, wird eine zweidimensionales Array genutzt. In einer Dimension sind jeweils ein Spielbrett, die zwei Spieler und den Spieler, der aktuell an der Reihe ist. In der anderen Dimension die verschiedenen Spiele. Sobald ein Spiel beendet wurde, werden die Daten aus dem Array mit dem Wert null belegt. Wenn ein neues Spiel begonnen wird, wird das durch das Array iteriert und die erste Position, die frei ist (also den Wert null hat) genutzt. Wenn keine freie Position gefunden werden konnte, wird ein neues Array erschaffen und angefügt.

# 4 Implementierung

## 4.1 Datenbank, Login und Registrierung

Zu Beginn verbindet sich der Server mit der Datenbank. Server und Datenbank können über einen definierten Port kommunizieren. Der Server kann mittels Datenbankabfragen auf die Daten zugreifen. Wenn sich ein neuer User registriert, sendet der Client dem Server den Usernamen und das Passwort. Der Server fordert nun von der Datenbank gespeicherte Daten an, die den Usernamen und das Passwort enthalten. Wenn die Antwort der Datenbank ein leeres Objekt ist, ist noch kein User mit dem Usernamen und dem Passwort registriert und in der Datenbank wird ein neuer Eintrag erstellt.

Folgendes Bild zeigt den Ablauf des Logins:

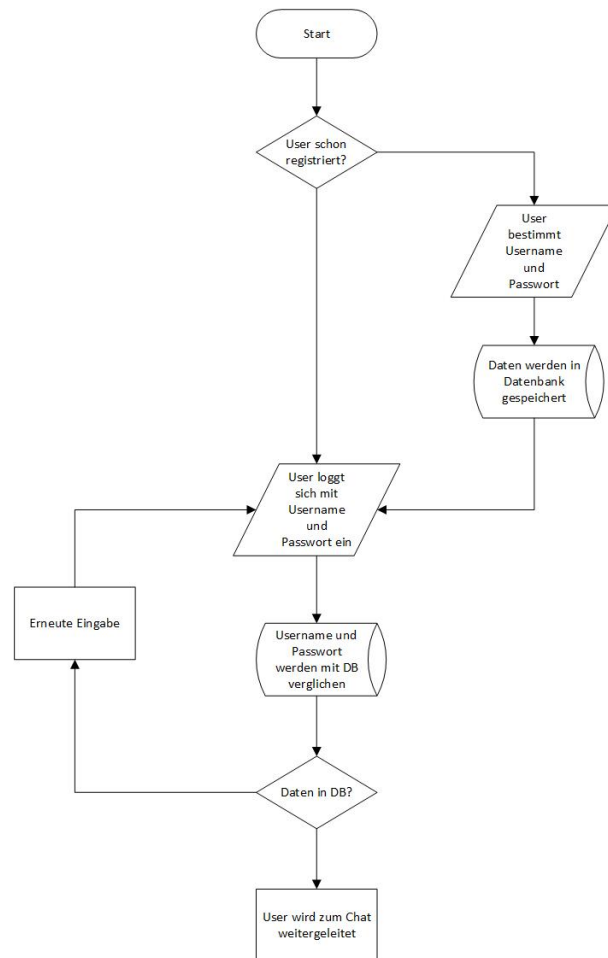


Abbildung 4.1: Flussdiagramm des Loginprozesses

## 4.2 Spiellogik

## 4.3 Erkennung des Spielendes

Es gibt zwei Szenarien, unter denen das Spiel zu Ende ist. Entweder gewinnt ein Spieler oder das Spiel endet unentschieden. Unentschieden ist das Spiel, wenn das Spielfeld komplett gefüllt ist, aber kein Spieler eine Reihe aus vier Steinen aufbauen konnte. Bei einem Gewinn sind drei verschiedene Szenarien zu unterscheiden. Eine Reihe aus vier Spielsteinen des gleichen Spielers kann horizontal, vertikal oder diagonal auftreten. Wenn die Reihe diagonal gebildet wird, kann noch zwischen von links unten nach rechts oben und von rechts unten nach links oben unterschieden werden. Alle diese Fälle müssen geprüft werden.

## 4.4 Docker Compose

Im Docker-Compose-File sind die zu verwaltenen Container aufgezählt mitsamt der Portnummern, über den auf die Container zugegriffen werden kann.



# 5 Projektabschluss, Fazit & Ausblick

## 5.1 Fazit

## 5.2 Ausblick

Ein momentaner Schwachpunkt der Datenbank ist, dass die Aussage, dass der Username schon existiert unter Umständen nicht richtig ist. Sobald der Username (z.B. test123) Teil eines schon bestehenden Usernames ist (z.B. test12345), kann die Abfrage dies nicht unterscheiden und gibt den bereits existierenden User zurück. Die Registrierung wird also verweigert, obwohl der Username noch nicht existiert. Durch eine genauere Prüfung der bereits existierenden Usernamen könnte dies verhindert werden.

Eine denkbare Erweiterung der Anwendung ist das Speichern des Spielstandes in der Datenbank. Zur Zeit ist das Spiel nur lokal zwischengespeichert und wird gelöscht, sobald das Spiel beendet ist, die Seite neu geladen wird oder der User sich ausloggt. Durch ein Speichern des Spielstandes und des Spielfeldes könnte auch bei einem erneuten Login das alte Spiel weitergespielt werden.

Um die Anwendung intuitiver zu gestalten, können dem User beim Login und der Registrierung Bildschirmausgaben über den aktuellen Stand angezeigt werden. Diese Ausgaben könnten über eine fehlgeschlagene Anmeldung informieren. Momentan wird bei einer fehlerhaften Anmeldung nur die Seite neu geladen.

# Literatur

## Publikationen

- [1] Peter Knoll. *Fahrstabilisierungssysteme und Fahrerassistenzsysteme*. Wiesbaden: Vieweg+Teubner Verlag, 2010. ISBN: 9783834813145.

# Anhang

A. Screenshot NameNode Web-Interface

B. DVD Inhalt

C. DVD

## **A. Screenshot NameNode Web-Interface**

## **C. DVD Inhalt**