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

Projekdokumentation Microservices

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Hanna Siegfried und Nahku Saidy

12.06.2019

Bearbeitungszeitraum	nothing
Matrikelnummer, Kurs	5946066,XXXXX, STG-TINF17-ITA
Ausbildungsfirma	Daimler AG, Stuttgart
Betreuer	nothing

Inhaltsverzeichnis

Abkürzungsverzeichnis

AABB Axis-Aligned Bounding Box

Abbildungsverzeichnis

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. Darauffolgend 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 Anwedung sind. Im letzten Kapitel wird noch einmal auf die Aufgabenstellung Rückbezug genommen und eine kritische Würdigung der erzielten Ergebnisse vorgenommen. Zusätzlich dazu wird ein Ausbilck 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 Schemas, 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.

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 **??** beinhaltet eine Übersicht über die Komponenten der Anwendung.

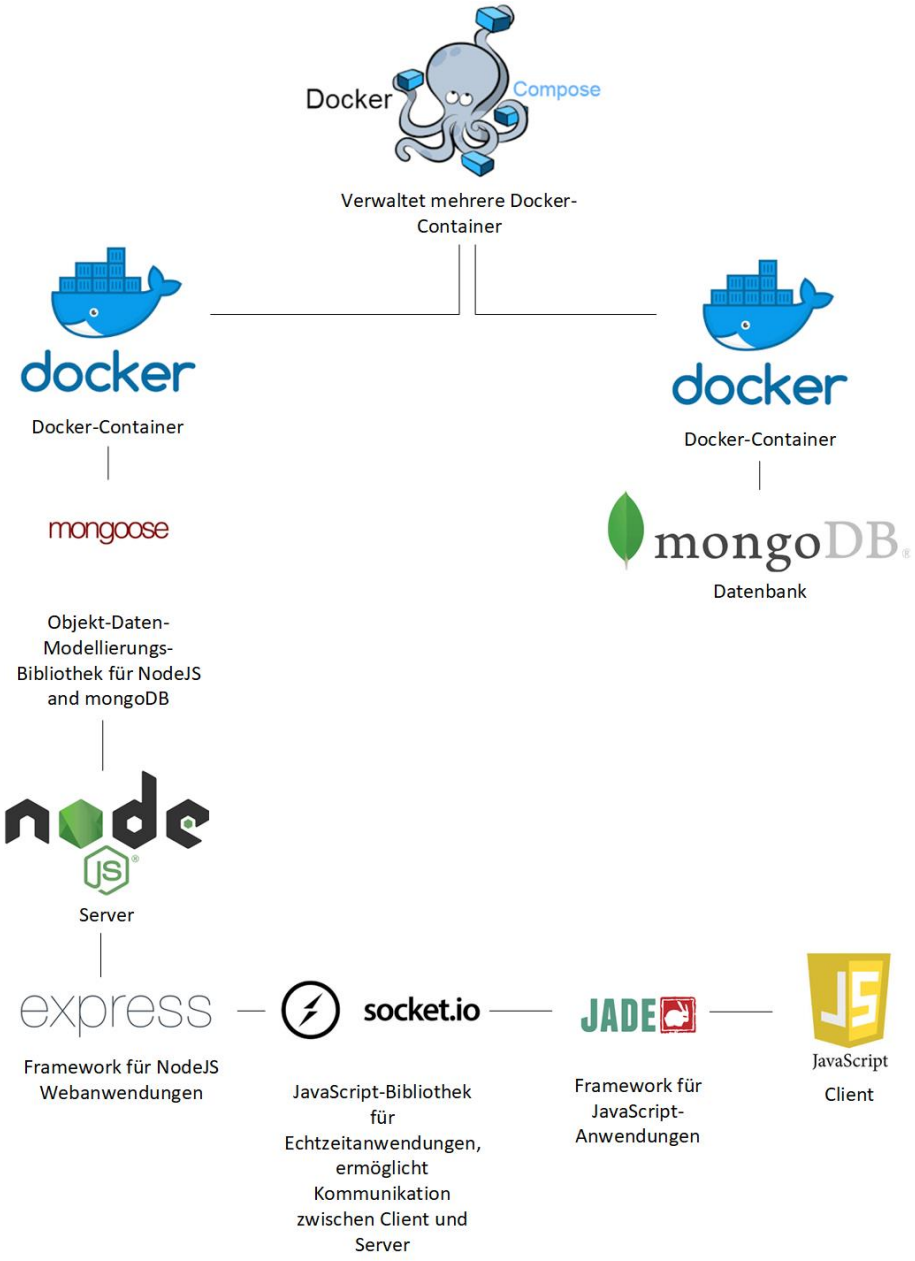

Abbildung 3.1: Übersicht der Anwendung

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 dies zu umgehen und nur den am Spiel beteiligten Spielern 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 aufgrund dessen, die Nachrichten anzeigen oder ignorieren.

3.6 Gleichzeitiges Spielen mehrerer Spieler

Um es mehr als zwei Spielern zu ermöglichen, gleichzeitig zu spielen, wird eine zweidimensionales Array genutzt. In einer Dimension sind jeweils ein Spielbrett, die zwei Spieler und der Spieler, der aktuell an der Reihe ist, gespeichert. In der anderen Dimension werden die verschiedenen Spiele gespeichert. Sobald ein Spiel beendet wird, werden die zugehörigen Variablen aus dem Array mit dem Wert null belegt. Wenn ein neues Spiel begonnen wird, wird durch das Array iteriert und die erste freie Position verwendet, um dieses Spiel zu speichern. Wenn keine freie Position gefunden werden konnte, das Array erweitert.

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:

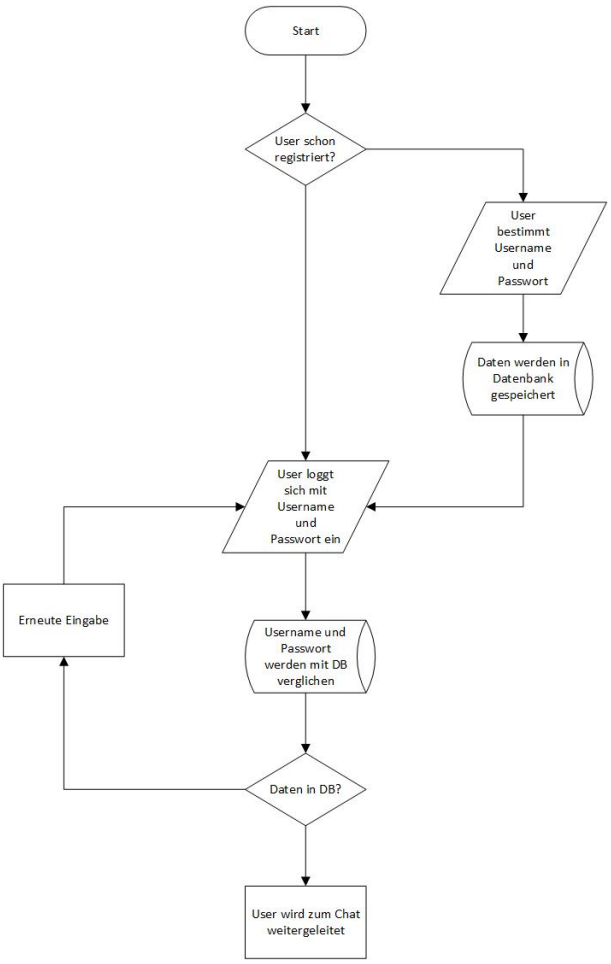

Abbildung 4.1: Flussdiagramm des Loginprozesses

Im folgenden ist der Verbindungsaufbau mit der MongoDB gezeigt. In der URL ist die Portnummer des MongoDB-Container, über den die Verbindung aufgebaut wird. Der darauffolgende Name ist der Name des Docker-Containers, aus dem die Datenbank aufgerufen wird. In der letzten Zeile wird das auf definierte Datenbankschema verwiesen.

```
1
2  mongoose
3    .connect(
4      'mongodb://mongo:27017/docker-node-mongo',
5      { useNewUrlParser: true }
6    )
7    .then(() => console.log('MongoDB Connected'))
8    .catch(err => console.log(err));
9
10 const User = require('./js/user');
```

Listing 4.1: Verbindungsaufbau mit der MongoDB

In folgendem Schema ist definiert, was für Attribute das in der Datenbank gespeicherte Objekt besitzt.

```

const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const UserSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  password: {
    type: String,
    required:true
  }
});

module.exports = User = mongoose.model('User', UserSchema);
```

Listing 4.2: Datenschema in der MongoDB

4.2 Spiellogik

Der User kann das Spiel über einen Knopf starten. Nun muss er warten bis ein anderer User das Spiel ebenfalls startet. Nachdem das Spiel gestartet ist, kann der User, der momentan an der Reihe anhand von Knöpfen über halb des Spielfeldes wählen, in welche Reihe er setzen möchte. Falls das Setzen nicht möglich ist, da die Reihe voll ist, darf er erneut setzen. Wenn ein Spieler einen Stein setzen möchte, obwohl er nicht an der Reihe ist, wird dieser nicht gesetzt. Nach jedem Zug wird geprüft, ob das Spiel endet. Wenn ja, kann ein neues Spiel begonnen werden.

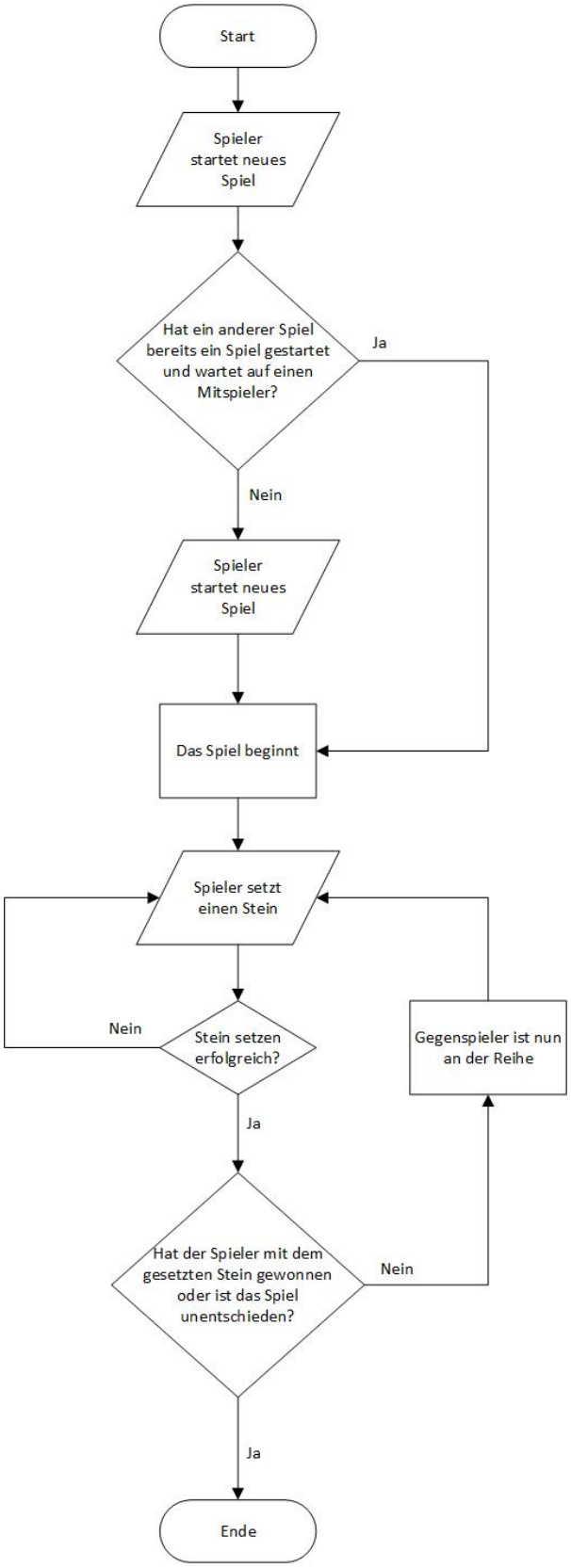

Abbildung 4.2: Flussdiagramm des Spielablaufs

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.

In der ersten Zeile wird das Docker Compose File Format genannt. Version 3 ist die aktuelle Version. In den folgenden Zeilen werden die einzelnen Services, also die einzelnen Docker-Container festgelegt. Die Portnummern beschreiben das Abbild von interne auf externe Portnummer. Mit Hilfe der externen Portnummer auf der rechten Seite können die Docker-Container miteinander kommunizieren. In Zeile 6 wird der Pfad hinterlegt, in dem das Docker-File hinterlegt ist, welches die build-Funktion spezifiert. In Zeile 16 ist der Name des erstellten Images festgelegt, aus dem der Container gestartet wird. Der Container-Name in Zeile 4 beschreibt den Namen des Docker-Containers, anstatt einen default Namen zu benutzen. Links in Zeile 10 benennt Links zu anderen Containern außerhalb des Docker-Compose-Files. Referenz: https://docs.docker.com/compose/compose-file/

```
1  version: '3'
2  services:
3    app:
4      container_name: docker-node-mongo
5      restart: always
6      build: .
7      ports:
8        - '80:3000'
9        - '8181:8181'
10     links:
```

```
11        - mongo
12    mongo:
13      container_name: mongo
14      image: mongo
15      ports:
16        - '27017:27017'
```

Listing 4.3: docker-compose.yml-File

5 Projektabschluss, Fazit & Ausblick

5.1 Fazit

5.2 Ausblick

Ein momentaner Schwachpunkt der Benutzerverwaltung ist, dass nicht vollständig erkannt wird, ob ein Benutzername schon existiert. Sobald der Username (z.B. test123) Teil eines schon bestehendes Usernames ist (z.B. test12345), kann dies nicht unterscheiden werden. Daher würde der Benutzername als schon vergeben klassifiziert werden. Die Registrierung wird also verweigert, obwohl der Username noch nicht exisitert. 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 das Speichern des Spielstandes und des Spielfeldes in der Datenbank 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. Aktuell wird bei einer fehlerhaften Anmeldung die Seite erneut 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