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

15.06.2019

Bearbeitungszeitraum	13.05.2019 - 15.06.2019
Matrikelnummer, Kurs	6430174, 5946066, STG-TINF17-ITA
Ausbildungsfirma	Daimler AG, Stuttgart
Dozent	Ingolf Buttig

Inhaltsverzeichnis

Abkürzungsverzeichnis I

Abbildungsverzeichnis II

Listings III

1 Einleitung 1
- 1.1 Aufgabenstellung und Ziel der Projektarbeit . . . 1
- 1.2 Aufbau der Projektarbeit . . . 1
- 1.3 Wer hat was gemacht? . . . 2

2 Grundlagen und Stand der Technik 3
- 2.1 Node.js . . . 3
- 2.2 Websockets . . . 3
- 2.3 Docker . . . 4
- 2.4 MongoDB und Mongoose . . . 4

3 Konzept 5
- 3.1 Anforderungen . . . 5
- 3.2 Übersicht . . . 5
- 3.3 Login und Registrierung . . . 7
- 3.4 Spiel . . . 7
- 3.5 Nachrichtenfilterung . . . 7
- 3.6 Gleichzeitiges Spielen mehrerer Spieler . . . 8

4 Implementierung 9
- 4.1 Datenbank, Login und Registrierung . . . 9
- 4.2 Spiellogik . . . 11
- 4.3 Erkennung des Spielendes . . . 14
- 4.4 Docker Compose . . . 15

5 Projektabschluss, Fazit & Ausblick 17
- 5.1 Fazit . . . 17
- 5.2 Ausblick . . . 17

Literatur i

Anhang ii

Abkürzungsverzeichnis

HTTP	Hypertext Transfer Protocol
NPM	Node Packet Manager

Abbildungsverzeichnis

3.1 Übersicht der Anwendung 6

4.1 Flussdiagramm des Loginprozesses 10
4.2 Flussdiagramm des Spielablaufs 13

Listings

4.1 Verbindungsaufbau mit der MongoDB 11
4.2 Datenschema in der MongoDB 11
4.3 Vertikale Game-Over Überprüfung 14
4.4 docker-compose.yml-File 16

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. Es soll mehreren Spielern das gleichzeitige Spielen ermöglichen. Die Anforderungen an die Anwendung werden in Kapitel 3.1 beschrieben.

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 ausschnittweise 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?

Das Konzept der Anwendung wurde gemeinsam entwickelt. Der Großteil des Quellcodes wurden nach der pair-programming Arbeitstechnik erstellt, wobei Hanna vorrangig für die Kommunikation verantwortlich war und Nahku die Spiellogik behandelt hat.

2 Grundlagen und Stand der Technik

2.1 Node.js

Node.js ist eine JavaScript-Plattform, die JavaScript außerhalb des Browsers ausführt. Node.js wird häufig serverseitig benutzt, um Daten zu Senden/zu Empfangen. [vgl. 1]

Der Unterschied zwischen Node.js und anderen Sprachen zur Serverprogrammierung liegt darin, dass Node.js keinen neuen Thread für neue Requests startet, sondern alle Requests auf einem Single-Thread ausführt [vgl. 2, S. 3]. Node.js arbeitet asynchron und verarbeitet neu eintreffende Befehle unmittelbar. Umgesetzt wird das mit nicht-blockierenden I/O-Anfragen. Während des Wartens auf die Peripherie können andere Befehle ausgeführt werden. Ein weiterer Vorteil, der sich daraus ergibt, ist, dass keine Deadlocks auftreten können, weil Ressourcen nicht blockiert werden. Aus diesem Grund bietet sich Node.js für skalierbare Netzwerkanwendungen an. [vgl. 2, S. 4]
Mit Hilfe des Node Packet Manager (NPM) können Pakete installiert werden, die zusätzliche Funktionalitäten bieten. Dies folgt einem ähnlichen Konzept wie Bibliotheken in anderen Programmiersprachen.

2.2 Websockets

Websockets sind die Grundlage für die Chat-Anwendung. Der Vorteil von Websockets gegenüber z. B. einem reinen Hypertext Transfer Protocol (HTTP)-Protokoll, liegt darin, dass die Verbindung vom Client nur einmal geöffnet werden muss. Dann kann der Server dem Client Informationen senden ohne dafür eine neue Verbindung zu benötigen. Dies ist bei Chats wichtig, da jederzeit neue Nachrichten eintreffen können, die ohne Verzögerung an den Client weitergeleitet werden sollen. Müsste

der Client dafür eine neue Verbindung einrichten, so würden die Nachrichten nicht ohne Zeitverzögerungen ankommen.

2.3 Docker

In dieser Projektarbeit wird Docker verwendet, um das einfache Deployen der Anwendungen unabhängig von den konkreten Servern zu ermöglichen. Docker ermöglicht es, Container auf Server zu deployen und stellt dabei sicher, dass die Umgebung für die Software im Container insoweit gleichbleibt, dass die Software genauso funktioniert wie auf anderen Servern mit anderer Hardware. So kann Software allgemein, ohne Probleme beim Deployen auf verschiedene Server, entwickelt werden.
Ein Container-Image enthält eine ausführbare Software, einschließlich aller Abhängigkeiten, die zur Ausführung der Software benötigt werden. Ein auf der Docker-Engine ausgeführtes Container-Image wird als Container bezeichnet. Dadurch, dass alle zur Ausführung der Software benötigten Daten und Funktionen im Container-Image vorhanden sind, können die Container einfach auf verschiedene Server deployt werden, ohne dass es zu Problemen kommt. [vgl. 3]

2.4 MongoDB und Mongoose

In der Anwendung wird eine MongoDB zum Speichern der Zugangsdaten der Chat-Teilnehmer genutzt. MongoDB ist eine NoSQL-Datenbank. Im Gegensatz zu relationalen Datenbanken ist MongoDB dokumentenorientiert [vgl. 4, S. 3], es werden also Dokumente gespeichert. Ein Dokument besteht aus einer Menge an Keys und Values. Dieser Aufbau der Datenbank bietet unter anderem eine besser Skalierbarkeit als relationale Datenbanken, da die Daten besser auf verschiedene Server aufgeteilt werden können [vgl. 4, S. 4]. Es gibt keine vorgegeben Schemas, die eingehalten werden müssen.

Mongoose stellt eine Objekt-Daten-Modellierungs-Bibliothek für MongoDB und Node.js zur Verfügung. Mit Mongoose kann das Datenschema der Datenbank im Node.js-Code in JSON definiert werden. [vgl. 2, S. 10]

3 Konzept

3.1 Anforderungen

Im Folgenden werden die Funktionalitäten erläutert, die die Anwendung besitzen soll. Hauptziel ist es, dass Nutzer, parallel zur Nutzung des Chats, ein 4-gewinnt-Spiel spielen können. Dabei spielen jeweils zwei Spieler gegeneinander, wobei die Anwendung von mehr als zwei Nutzern gleichzeitig genutzt werden können soll. Vor der Teilnahme an dem Chat und einem Spiel sollen sich die Nutzer einloggen. Dafür soll eine Registrierungsmöglichkeit geschaffen werden und die Registrierungsdaten sollen in einer Datenbank gespeichert werden. Wenn ein Spieler ein Spiel starten möchte, so soll dieser mit dem nächsten Spieler spielen können, der auch ein Spiel starten möchte.

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 beinhaltet eine Übersicht über die Komponenten der Anwendung.

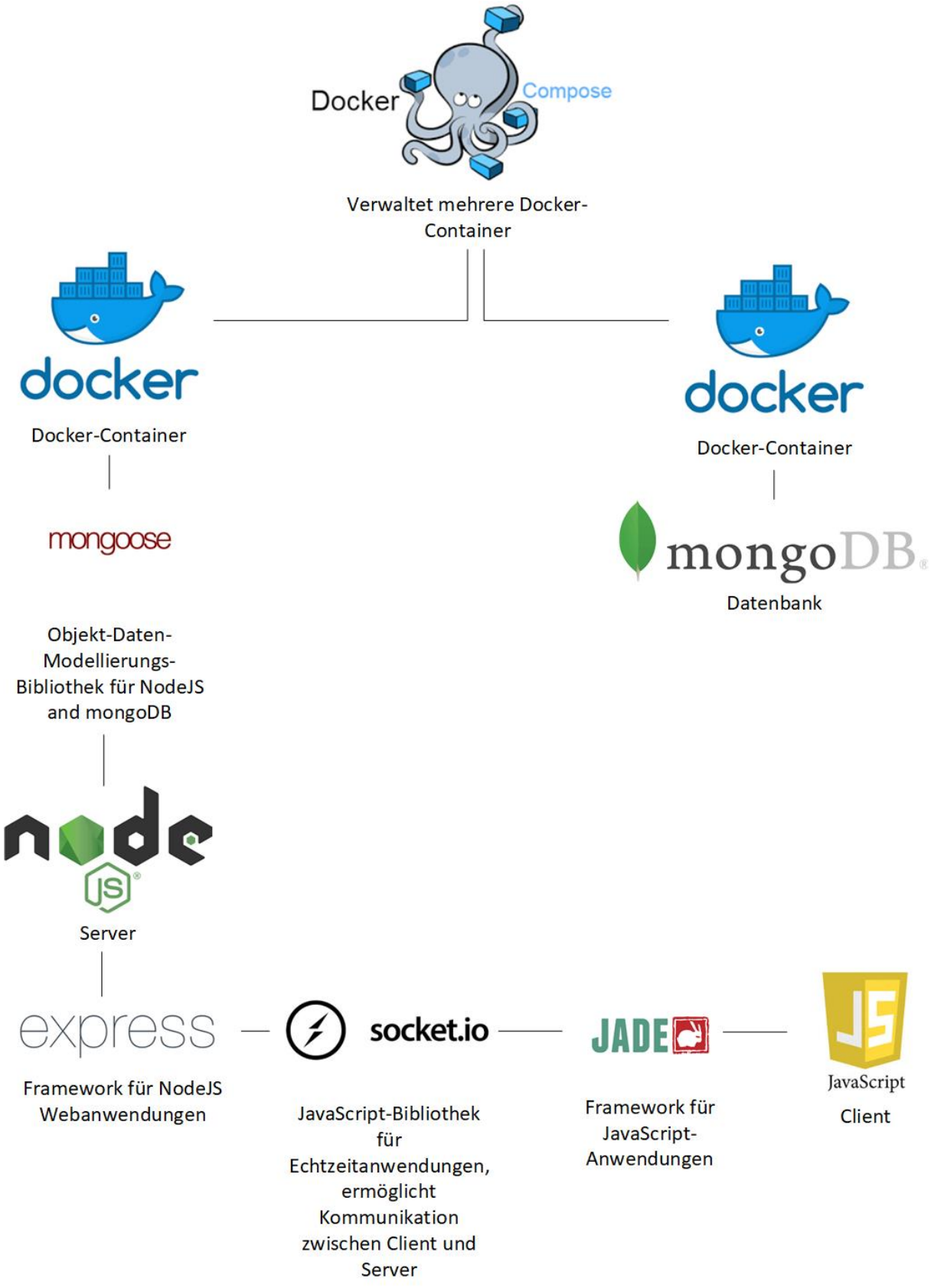

Abbildung 3.1: Übersicht der Anwendung

3.3 Login und Registrierung

Beim Login hat der User die Möglichkeit, sich zu registrieren oder sich direkt einzuloggen. Die Login-Informationen werden bei der Registrierung in der MongoDB gespeichert und verwaltet.

3.4 Spiel

Nachdem zwei Spieler ein Spiel begonnen haben, können sie gegeneinander spielen. Mittels Buttons kann ausgewählt werden, in welche Spalte sie setzen möchten. Hierbei muss überprüft werden, ob der Spieler an der Reihe ist und ob die Spalte, in der der Stein gesetzt werden soll, nicht voll ist. Außerdem muss geprüft werden, ob es ein Spieler geschafft hat, vier Steine in eine vertikale, horizontale oder diagonale Reihe zu platzieren. Nachdem das Spiel entweder gewonnen oder unentschieden beendet wird, wird das aktuelle Spielfeld deaktiviert und es besteht die Möglichkeit, ein neues Spiel zu starten. Um dem Spieler Informationen über den Spielstand mitzuteilen, schickt ein sogenannter Gamemaster Nachrichten an die Spieler. Diese Nachrichten werden wie die Chat-Nachrichten über den Websocket geschickt.

Das Spielfeld besteht aus 7 Spalten und 6 Zeilen, dieses wird als zweidimensionales Array gespeichert. Wenn ein Element den Wert Null hat, bedeutet das, dass kein Spieler an dieser Position einen Stein gesetzt hat.

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 ein 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.

4 Implementierung

4.1 Datenbank, Login und Registrierung

Zu Beginn verbindet sich der Server mit der Datenbank. Server und Datenbank können über einen vorher definierten Port kommunizieren. Der Server kann mittels Datenbankabfragen auf die gespeicherten 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 alle gespeicherte Daten an, die mit dem Usernamen und dem Passwort übereinstimmen und entscheidet anhand dessen, ob der Username und das Passwort bereits vergeben sind oder noch frei sind. 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. In der folgenden Abbildung 4.1 ist der Login-Prozess schematisch dargestellt.

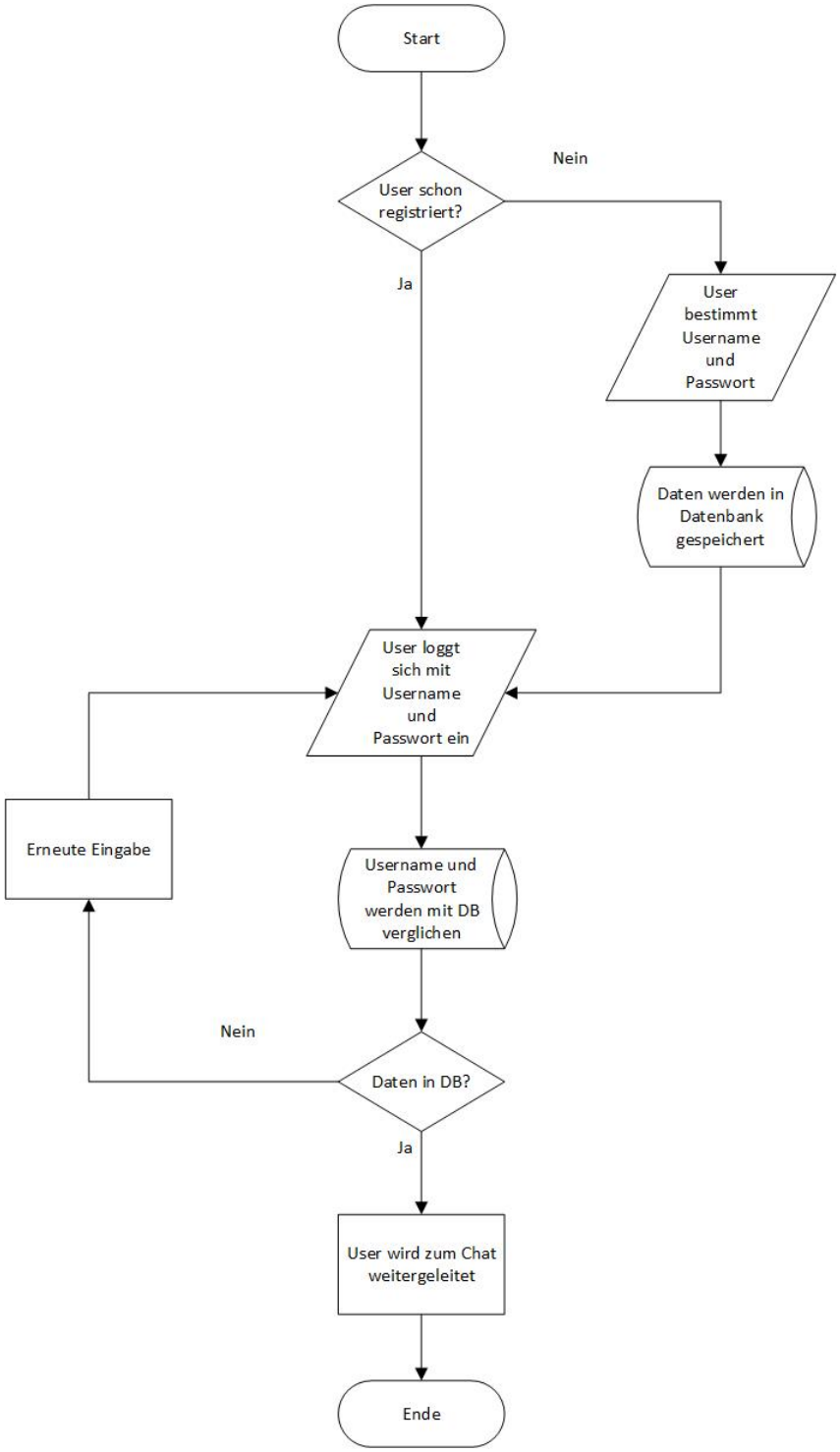

Abbildung 4.1: Flussdiagramm des Loginprozesses

In folgenden Listing 4.1 ist der Verbindungsaufbau mit der MongoDB gezeigt. In der URL ist die Portnummer des MongoDB-Container, über den die Verbindung aufgebaut wird, enthalten. Der darauffolgende Name ist der Name des Docker-

Containers, aus dem die Datenbank aufgerufen wird. In der letzten Zeile wird auf das definierte Datenbankschema verwiesen.

```
mongoose
  .connect(
    'mongodb://mongo:27017/docker-node-mongo',
    { useNewUrlParser: true }
  )
  .then(() => console.log('MongoDB Connected'))
  .catch(err => console.log(err));

const User = require('./js/user');
```

Listing 4.1: Verbindungsaufbau mit der MongoDB

In Listing 4.2 ist das Datenschema der Daten definiert, die in der MongoDB gespeichert werden. Die User-Informationen bestehen in dieser Anwendung aus einem Nutzernamen und einem Passwort. Beides wird benötigt und darf nicht leer bleiben. Außerdem sind beide vom Datentyp String.

```
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. Danach kann der User, der momentan

an der Reihe ist, anhand von Knöpfen überhalb des Spielfeldes wählen, in welche Spalte er setzen möchte. Falls das Setzen nicht möglich ist, weil die Reihe schon voll ist, wird er zum erneuten Setzen an einer anderen Stelle aufgefordert. Wenn ein Spieler einen Stein setzen möchte, obwohl er nicht an der Reihe ist, wird diese Operation nicht durchgeführt. Nach jedem Zug wird geprüft, ob das Spiel vorbei ist. Wenn dies der Fall ist, kann ein neues Spiel begonnen werden. Dieser Ablauf ist in Abbildung 4.2 im Folgenden schematisch dargestellt.

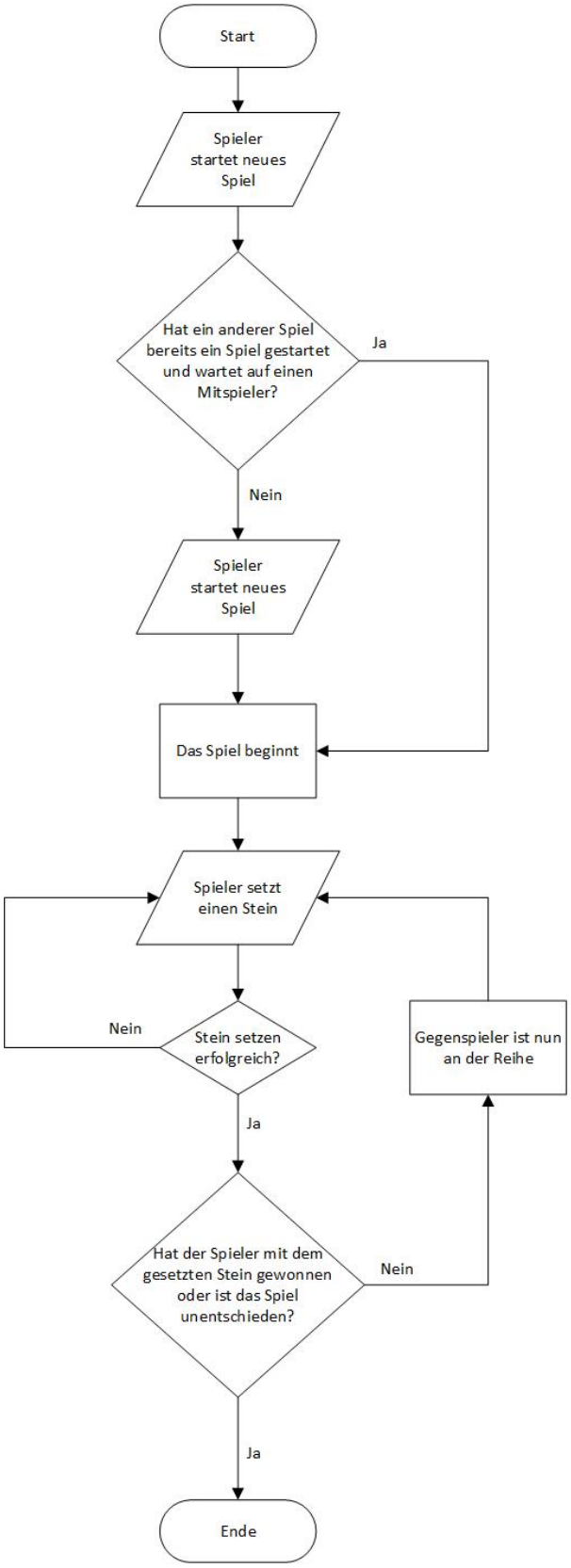

Abbildung 4.2: Flussdiagramm des Spielablaufs

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. Für jeden dieser Fälle hat der Server eine Funktion, um das Spielfeld zu überprüfen.
Als Beispiel wird im Folgenden in Listing 4.3 die Erkennung einer vertikalen Reihe aus vier Spielsteinen des gleichen Spielers erläutert. Wenn kein Spieler vertikal gewonnen hat, gibt diese Funktion den Wert 0 zurück, ansonsten die ID des Spielers, der vertikal gewonnen hat. Dazu werden zwei verschachtelte For-Schleifen verwendet, wobei die äußere die Spalten des Spielfelds durchläuft und die innere ein Offset in den Zeilen erzeugt (Zeile 2 und 3). Die boolsche Variable $flag$, die in Zeile 4 deklariert und mit *true* initialisiert wird, gibt an, ob im aktuellen Durchlauf ein Gewinner gefunden werden konnte. Ist $flag$ bei der if-Abfrage in Zeile 14 immer noch wahr, so wurde ein Gewinner gefunden. Dieser wird dann zurückgegeben. Hat der Spielstein ($lowestPlayerTile$) bei dem die Prüfung jeweils anfängt den Wert 0, so muss nicht weiter geprüft werden, da dieses Feld noch von keinem Spieler besetzt wurde. Ist dies nicht der Fall, so wird die For-Schleife in Zeile 7 durchlaufen. Ausgehend von dem Spielstein $lowestPlayerTile$ werden dort die drei rechts davon liegenden Spielsteine überprüft. Ist einer davon nicht von dem selben Spieler wie $lowestPlayerTile$, so wird die Schleife abgebrochen und $flag$ auf falsch gesetzt, da dann kein Gewinn vorliegt. Gehören alle vier Spielsteine dem gleichen Spieler, so liegt ein Gewinn vor und die ID des Spielers wird zurückgegeben.

```
1 function checkVerticalGameOver(game){
2 for (let column=0; column<7; column++){
3   for (let row=0; row<3; row++) {
4    var flag = true;
5    var lowestPlayerTile = this._gameManager[game][0][row)[column];
6      if (lowestPlayerTile != 0){
7       for (let i=1; i<4; i++){
8        if(lowestPlayerTile !=
```

```
9           this._gameManager[game][0][row+1][column]){
10          flag = false;
11          break;
12           }
13        }
14       if(flag == true){
15        return lowestPlayerTile;
16       }
17      }
18     }
19    }
20 return 0;
21 }
```

Listing 4.3: Vertikale Game-Over Überprüfung

4.4 Docker Compose

In einem Docker-Compose-File sind die zu verwaltenen Container mit den Portnummern, über die auf die Container zugegriffen werden kann, enthalten. In Listing 4.4 ist das Docker-Compose-File dieser Anwendung dargestellt. 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. Der Container-Name in Zeile 4 beschreibt den Namen des Docker-Containers, der anstatt eines automatisch festgelegtem Namen benutzt wird. In Zeile 6 wird der Pfad hinterlegt, in dem das Docker-File hinterlegt ist, welches die build-Funktion spezifiziert. *Links* in Zeile 10 benennt Links zu anderen Containern, in diesem Fall zu dem MongoDB-Container. Die Portnummern beschreiben die Abbildung von Container-internen auf externe Portnummern. Mit Hilfe der externen Portnummer können die Docker-Container miteinander kommunizieren, bzw. kann mit den Docker-Containern von außen kommuniziert werden. In Zeile 14 ist der Name des erstellten Images festgelegt, aus dem der Container gestartet wird.

```
version: '3'
services:
  app:
    container_name: docker-node-mongo
    restart: always
    build: .
    ports:
      - '80:3000'
      - '8181:8181'
    links:
      - mongo
  mongo:
    container_name: mongo
    image: mongo
    ports:
      - '27017:27017'
```

Listing 4.4: docker-compose.yml-File

5 Projektabschluss, Fazit & Ausblick

5.1 Fazit

Das Ziel dieser Projektarbeit war die Entwicklung eines 4-gewinnt-Spiels auf Basis der in der Vorlesung vorgestellten Chat-Anwendung. Dazu wurde eine Anwendung entwickelt, die aus einem Client, einem Server und einer Datenbank besteht. Um die Echtzeitfähigkeit, die sowohl für den Chat als auch das Spiel benötigt wird, zu gewährleisten, wurde ein Websocket eingesetzt. Außerdem wurde die Anwendung und die MongoDB in jeweils einen Docker-Container verpackt. Die Mehrbenutzerfähigkeit wurde durch eine Verwaltung mehrerer Spielfelder zugehörig zu den Spielern erreicht. Registrierung und Login werden ermöglicht, indem die Benutzerdaten in der Datenbank gespeichert werden. Dadurch können neue Nutzer hinzugefügt werden und es kann überprüft werden, ob ein Nutzer die richtigen Zugangsdaten eingegeben hat.
Somit erfüllt die Anwendung alle definierten Anforderungen, die in Kapitel 1 und Kapitel 3.1 genannt wurden.

5.2 Ausblick

Ein Punkt, an dem die Benutzerverwaltung noch verbessert werden kann, ist, dass nicht vollständig erkannt wird, ob ein Benutzername schon existiert. Sobald der gewünschte Username (z.B. test123) Teil eines schon bestehendes Usernamens ist (z.B. test12345), wird nicht erkannt, dass die Nutzernamen unterschiedlich sind. Daher würde der Benutzername als schon vergeben klassifiziert werden. Die Registrierung wird also verweigert, obwohl der Username noch nicht existiert. Durch eine genauere Prüfung der bereits existierenden Usernamen könnte dies verhindert werden. Bei der Prüfung, ob der Nutzername bereits vergeben ist, schickt die Datenbank alle Datensätze, die entweder gleich sind oder den eingegebenen Namen enthalten, an den Server. Diese Datensätze werden als JSON-Array gesendet. Der Server müsste für eine vollständige Prüfung, das JSON-Array parsen und den

Namen mit den eingegeben Daten vergleichen und anhand dessen feststellen, ob diese identisch sind oder nicht.

Außerdem werden die Passwörter der Nutzer momentan als Klartext in der MongoDB gespeichert. Für die Zukunft wäre es denkbar, die Passwörter verschlüsselt zu speichern. Dazu könnte man beispielsweise einen aus dem Nutzer-Passwort berechneten Hash-Code in der Datenbank speichern. Dies würde sich aus Gründen des Datenschutzes anbieten.

Eine denkbare Erweiterung der Anwendung ist das Speichern des Spielstandes in der Datenbank. Zurzeit 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

[2] Simon Holmes. *Mongoose for Application Development*. Birmingham, UK: Packt Publishing, 2013. ISBN: 9781782168195.

[4] Kristina Chadorow. *MongoDB: The Definitive Guide*. 2., vollst. überarb. Aufl. Sebastopol, USA: O'Reilly Media USA, 2013. ISBN: 9781449344689.

Online-Quellen

[1] Node.js Foundation. *About Node.js*. URL: https://nodejs.org/en/about/ (besucht am 11. 06. 2019).

[3] Docker Inc. *What is a Container?* URL: https://www.docker.com/resources/what-container (besucht am 13. 06. 2019).

Anhang

A. Nachweis der Funktionalität

Die Screenshots für den Nachweis der Funktionalität befinden sich im Ordner Images.

B. Starten der Anwendung

Folgende Anweisungen müssen zum Starten der Anwendung ausgeführt werden:

Um die Anwendung zu starten, muss Docker auf dem Rechner installiert sein.

Diese Anwendung wurde mit Docker Toolbox, Kitematic und einer Linux Virtual Maschine (Oracle VM VirtualBox) auf einem Windows 64-bit Rechner erstellt. Eventuell kann es bei einer anderen Umgebung zu Fehlern kommen.

1) Ermitteln der IP-Adresse der Docker default maschine z.B. localhost bei Docker Desktop oder 192.168.99.100 bei Dockertoolbox
2) Im File *app/views/chat.jade* muss die IP-Adresse, unter der der Server gefunden werden kann, richtig angegeben
werden. Bitte prüfen und ggf. einfügen.
3) Öffnen der Docker Konsole
4) Den Pfad des Ordners, in dem die Anwendung gespeichert ist, in die Konsole kopieren, sodass nun Kommandos auf den Ordner ausgefuehrt werden können
5) docker-compose build in die Konsole eingeben
6) docker-compose up in die Konsole eingeben
7) In der Konsole müsste nun Server running und MongoDB Connected angezeigt werden
8) Die Anwendung kann im Browser unter localhost:80 gefunden werden