Dokumentation für die Projektarbeit

Im Modul „Frameworks, Dienste und Daten im Web“ (FDDW)

an der Technischen Hochschule Köln

Anton Berg • Abdurrahman Karakan • Yasha Müller

Abstract:

This paper is the documentation of the project work that was completed in the module "Frameworks, Dienste und Daten im Web" (FDDW) which is part of the focus module “Web Develpment” of the BA Mediainformatics at the Cologne University of Applied Sciences. The module focuses on the concepts of asynchronous, server-side communication and related standards and frameworks.

EINLEITUNG

Dies ist die Dokumentation der Projektarbeit im Modul "Frameworks, Daten und Dienste im Web" (FDDW) im Rahmen des Schwerpunktbereichs "Web Development". Im Folgenden sollen die Arbeitsschritte und Tätigkeiten, die im laufe der Projektarbeit durchlaufen wurden beschrieben werden, um die Arbeit am Projekt nachvollziehbar darzustellen.

Während die ersten beiden Abschnitte (Teil I + Teil II) die Arbeit am Projekt chronologisch wiedergeben, soll in Teil III eine ausführliche Anleitung zur Nutzung des Systems gegeben werden.

TEIL I: THEMENFINDUNG UND KONZETION

1 PROBLEMSTELLUNG

Die Zielsetzung des Projekts gemäß den Vorgaben des Moduls lautet wie folgt:

- Anwendung die in Microservices / Komponenten aufgeteilt ist
- Asynchrone Kommunikation zwischen Microservices (bspw. durch Events)
- Anbindung mindestens eines externen (offenen) Datendienstes
- Es muss State geben! (Ohne State ist alles möglich)
- Einfache! Web-UI zur Interaktion mit der Anwendung
- Deployment der Anwendung
- Betriebskonzept
- Dokumentation und Nachvollziehbarkeit in GitHub

Entsprechend diesen Anforderungen galt es nun, ein Konzept auszuarbeiten, das diese erfüllt. Hierfür war zunächst eine passende Domäne gesucht, für die zwei unserer Ideen im folgenden aufgelistet und auf die Modulanforderungen überprüft und Möglichkeiten der Umsetzung evaluiert werden sollen.

1.1 Karten- oder Würfelspiel

Ein Multiplayer-Spiel, das auf einem bereits bestehenden Karten oder Würfelspiel basiert.

- Microservices / Komponenten: einzelne Spieler*innen, "Spieltisch" (also Spielbrett/Kartenstapel/etc.; können je nach Granularität ggf. auch als einzelne Komponenten umgesetzt werden)
- Asynchrone Kommunikation: rundenbasierte Spielprinzip oder in Echtzeit
- Externer (offener) Datendienst: Ein externes Login-System, welches ein schon bestehenden Account in dem jeweiligen System, für unseren zur verfügung stellt. (Ory, OpenIAM, Google, Apache Syncope, Twitter, etc.)

1.2 Chatsystem

Ein Chatraum pro Spielraum, in denen sich die Chat-Teilnehmer anmelden und Nachrichten austauschen können.

- Microservices / Komponenten: Chat-Teilnehmer*innen, Chatraum, ggf. Spielreaktionen, ggf. Chatbefehle
- Asynchrone Kommunikation: ein event beim Senden, ein event beim Empfangen
- Externer (offener) Datendienst: Fosscord, oder Eigenbau mit socket.io

2 AUSSPEZIFIZIERUNG DER IDEEN & KONZEPTION

2.1 Auswahl des Spiels

Für das Auswahl des betreffenden Spiels wurde in der weiteren Konzeption das Würfelspiel "Mäxchen" gewählt, da es sowohl eine einfache Spielmechanik mit klaren Regeln aufweist und darüber hinaus von der Interaktion der Spieler lebt, wodurch der geplante Chat auch im Spiel selbst an Bedeutung gewinnt.

2.2 Erläuterung der Spielregeln

2.2.1 Spielprinzip:

Gespielt wird das Spiel mit zwei Würfeln und einer Spielerzahl die im besten Fall aus drei oder mehr Spieler:innen besteht (ein Partie mit nur zwei Spieler:innen ist jedoch auch prinzipiell möglich). Ziel ist es bei jedem Wurf, einen höheren Wert zu erzielen, als die/der Spieler:in zuvor. Da verdeckt gewürfelt wird, kann die/der Würfelnde entweder die Wahrheit sagen, oder sofern sein Ergebnis zu niedrig ist, lügen. Die/der nächste Spieler:in entscheidet, ob sie/er die Würfelzahl glaubt und ohne einen Blick unter den Würfelbecher zu werfen weiter würfelt, oder nicht. Wer beim Lügen enttarnt wird, oder eine:n Mitspieler:in fälschlicherweise der Lüge bezichtigt, verliert Punkte. Wer 0 Punkte hat, scheidet aus dem Spiel aus.

2.2.2 Wertung der Würfelzahlen:

Jede Augenkombination hat einen festen Wert, der dadurch bestimmt wird, dass die höhere Zahl die Zehner-, die niedrigere die Einer-Komponente darstellt. Wird also eine 6 und eine 2 gewürfelt, ist dies eine 62; wird eine 2 und eine 3 gewürfelt, eine 32 und so weiter. Die niedrigsten Werte haben die Zahlen 32 bis 65, sofern die beiden Würfelzahlen unterschiedlich sind und aufsteigend entsprechend ihrem Zahlenwert; gefolgt von 1er-Pasch (1 1) bis 6er-Pasch (6 6), ebenfalls aufsteigend entsprechend dem Zahlenwert. Ein Sonderfall ist das sogenannte Mäxchen (Würfelkombination: 2 1).

2.3 Funktion des Chats

Der Chat stellt hier einen Knotenpunkt dar, da er sowohl als Kommunikationsplattform für die Spielenden, als auch als Ausgabe für alle offen gespielten Aktionen dient. Der Chat sollte also der Art gestaltet sein, dass Textnachrichten der Spieler:innen visuell von den Spielaktionen abgehoben werden. Je nach weiterer Konzeption sollte es auch möglich sein, individuelle Nachrichten an einzelne Spieler:innen auszugeben, sowie Chat-Eingaben nicht als Nachrichten, sondern als Spielaktionen zu interpretieren. Wie der Chat für die Ein- und Ausgabe von Spielaktionen genutzt werden kann, soll im folgenden Kapitel beleuchtet werden.

3 INTERAKTIONSVARIANTEN

Wie zuvor beschrieben repräsentiert der Chat nicht nur die Kommunikation zwischen den Spieler:innen, sondern auch den Spieltisch selbst, d.h. neben Konversation werden auch offene Spielaktionen, sowie Eingabeaufforderungen oder Fehlermeldungen angezeigt. Die folgenden screenshot-ähnlichen Darstellungen, sollen zeigen, wie das aus Spieler:innen-Sicht aussehen könnte. Sie soll jedoch keineswegs das spätere Frontend wiedergeben, sondern lediglich Interaktionsmöglichkeiten visualisieren.

3.1 Interaktion nur über den Chat

Die Interaktion könnte beispielsweise nur über den Chat stattfinden, indem normale Nachrichten zum Zwecke der Konversation einfach nur als Text eingegeben werden, während Spielaktionen über feste Kommandos, angeführt durch ein "/" ausgeführt werden. Eingabeaufforderungen, Feedback zu einer Spielaktion, Informationen über den aktuellen Spielstand oder Fehlermeldungen werden über das Chatfenster wiedergegeben und visuell vom Konversationstext abgehoben.

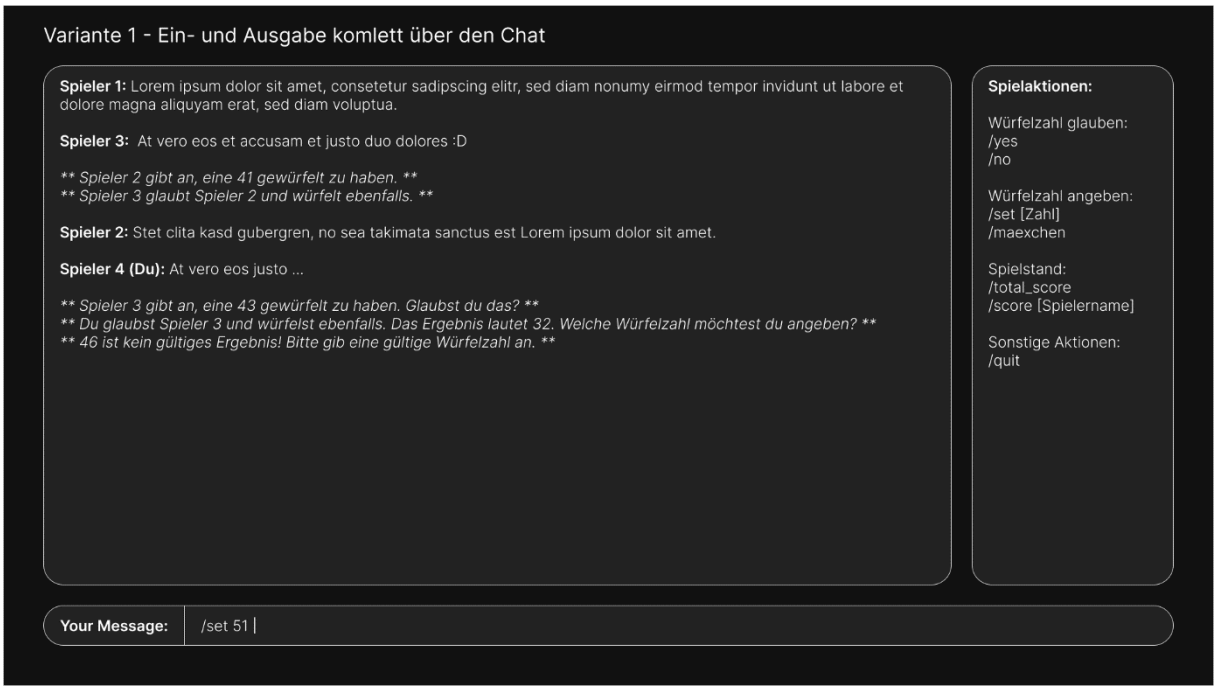

Abb.1: Beispiellayout bei reiner Chatsteuerung

Hierbei wird die Ausgabe der entsprechenden Informationen durch Aktionen der/des Spieler:in initiiert und nur angezeigt, wenn der zugrundeliegende Befehl ausgeführt wurde. Der Punktestand einer/einer Spieler:in wird also nur angezeigt, wenn er durch den Befehl angefordert wird.

3.2 Interaktion über zwei separate Bereiche

Alternativ könnte die Ausgabe der Spielaktionen weiterhin über den Chat erfolgen, Eingabeaufforderungen und Aktionen jedoch in einen sparaten Bereich ausgelagert und über andere Interaktionselemente realisiert werden. Nur Feedback über eigene oder gegnerische Spielzüge, sowie Fehlermeldungen werden, visuell vom Konversationstext abgehoben, in der Chatausgabe angezeigt. Eingabeaufforderungen und Spielstandsinformationen werden im separaten Bereich angegeben.

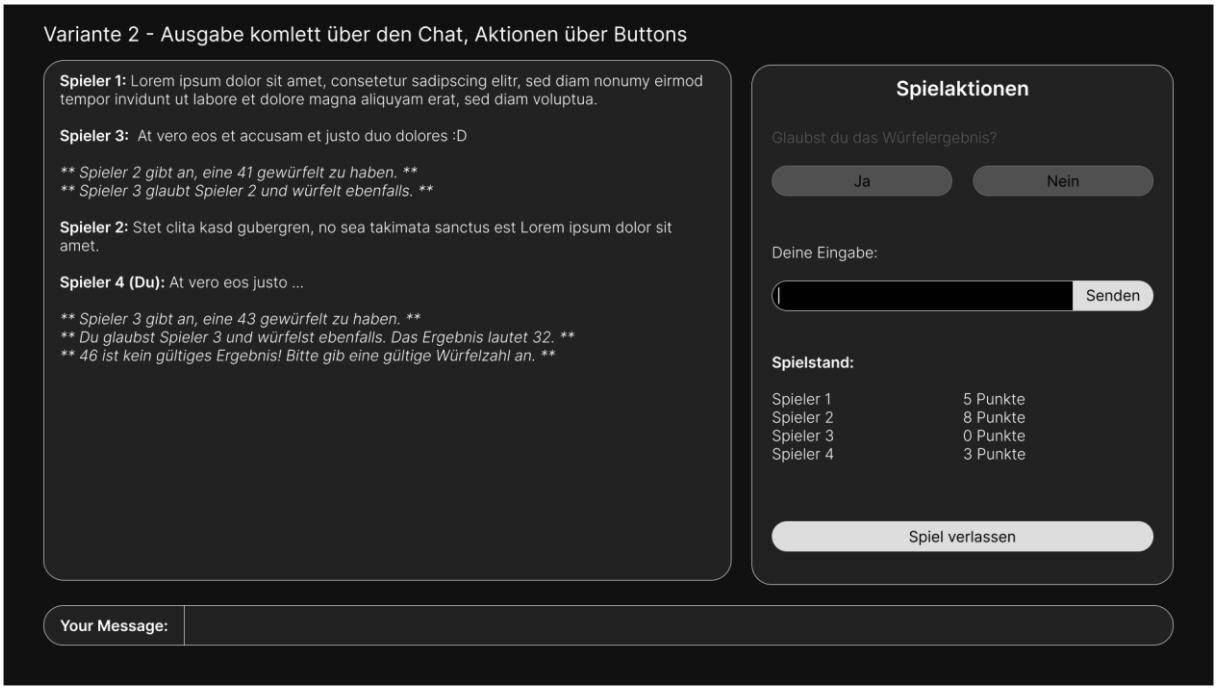

Abb.2: Beispiellayout bei Steuerung in separatem Bereich

Für die/den Spieler:in muss ersichtlich sein, welche Aktionen zum aktuellen Zeitpunkt möglich sind (hier durch Ausgrauen entsprechender Bereiche umgesetzt), d.h. es muss eine Aktualisierung der angezeigten Informationen stattfinden, sobald sich der Status eines Objekts ändert. Dasselbe gilt auch für den Spielstand.

TEIL II: UMSETZUNG UND CODE

4 AUFTEILUNG IN MICROSERVICES

Im Großen und Ganzen lässt sich das System in seiner Umsetzung in vier untereinander kommunizierende Services unterteilen, die jeweils für eine Funktion des Gesamtsystems verantwortlich sind. Das Spiel selbst besteht aus einem Chatsystem, sowie dem virtuellen Spieltisch, die beide jeweils ein Teilsystem darstellen. Die Datenverwaltung und der Login werden über extern eingebundene Services realisiert. Im Folgenden sollen die vier Services genauer beschrieben werden.

4.1 Das Chatsystem

4.1.1 Beschreibung

Über das Chatsystem führen die Spieler:innen all ihre Aktionen innerhalb des Spiels durch, was sowohl die reine Kommunikation mit Mitspieler:innen, als auch solche Aktionen, die das Spielgeschehen betreffen, beinhaltet. Dies bedeutet, Spieler:innen, die eine Spielaktion durchführen wollen, verwenden hierzu ebenfalls das Chatfenster, indem sie bestimmte, festgelegte Befehle eingeben. Diese folgen im Allgemeinen der Syntax:

```
/<Befehl> <Parameter>
```

Wird ein Befehl korrekt eingegeben, wird er als solcher erkannt und nicht als Chatnachricht im Chatfenster ausgegeben, sondern eine Funktion im Game-Service aufgerufen, die die damit verknüpfte Aktion durchführt. Voraussetzung hierzu ist, dass der entsprechende Spieler an der Reihe ist. Auf dem aktuellen Entwicklungsstand des Systems wird jedoch noch nicht nicht überprüft, ob die Aktion auch legal, also den Spielregeln entsprechend, ist (siehe auch: Buglist). Es wird also, wie auch im realen Leben, zunächst noch davon ausgegangen, dass die Spieler:innen sich an die Spielregeln halten.
In der Anleitung in Teil III findet sich eine Liste aller Chatbefehle.

4.1.2 Code

Der Chat-Service enthält drei wichtige Komponenten. Neben dem Datenbank-Model für das Nutzer:innen-Objekt in der User.js und den Views, die das Frontend - hier wird bislang noch das Standard-Frontend von socket.io verwendet - wiedergeben, bildet die chat.js, die die chatbezogenen Funktionalitäten umsetzt, das Herzstück des Services. Hier werden zunächst socket.io, express.js und mongoose implementiert und der Server gestartet.

Auch die Überprüfung der Tatsache, ob ein:e Spieler:in angemeldet ist, oder nicht und eine entsprechende Reaktion werden in der chat.js geregelt. Trifft dies zu, so wird ihr/ihm das normale Frontend mit dem Chatsystem dargestellt. Ist sie/er jedoch noch nicht eingeloggt, oder es kam zu Fehlern beim Login, so wird sie/er zum Login weiter-, bzw. zurückgeleitet.

Die letzte, aber nicht minder wichtige, Funktion der chat.js ist es, den Nachrichtenverkehr zwischen den Spieler:innen zu verwalten. Hierzu gehört neben dem Aufbereiten und Darstellen der schriftlichen Äußerungen, die allein dem Zwecke der Konversation dienen, auch das Erkennen und Analysieren von Befehlen zur Durchführung von Spielaktionen. Diese werden der REST-Architektur entsprechend in "GET-Befehle", "POST-Befehle" und "POSTPARAM-Befehle" (POST-Befehl mit zusätzlichen Parametern) unterteilt. Diese werden dann, wie im folgenden Beispiel (chat.js, Zeile 154 - 174) dargestellt, über einen Promise an den Game-Service übermittelt und dort verarbeitet.

```
function makePostRequest(path){
    return new Promise((resolve, reject) =>{
        const options = {
            hostname: 'game',
            port: 3000,
            path: path,
            method: 'POST',
        };
        const req = http.request(options, res => {
```

```
            retMsg = res.on('data', d => {
                resolve(d.toString())
            });
        });
        req.on('error', error => {
            console.error(error);
            reject(error)
        });

        req.end();
    })
}
```

4.2 Game

4.2.1 Beschreibung

Das Game ist der Service, der die Spiellogik abbildet. Hier sind die entsprechenden Funktionen angelegt, die über die Befehle aus dem Chatsystem aufgerufen werden können. Der Game-Service ist also für die Verarbeitung aller spielbezogenen Daten zuständig.

4.2.2 Code

Der Game-Service besteht aus im Großen und Ganzen aus der spieler.js und der game.js. Die spieler.js kann als Nutzerinnenrepräsentation als Spieler:in mit allen sielrelevanten Daten (bspw. Punktestand) betrachtet werden.

Die game.js implementiert zunächst express.js. Als eigentliches Herzstück enthält sie jedoch das Objekt "spielTisch", das wie der Name bereits vermuten lässt, alle wichtigen temporären Werte des Sielgeschehens verwaltet, beispielsweise die aktuellen Würfelzahlen oder die hierzu im Raum stehende Behauptung. Über States wird festgelegt, welche:r Spieler:in gerade an der Reihe ist.

Auch die von der chat.js übermittelten Funktionen werden in der game.js verarbeitet. Im folgenden Codeblock (game.js, Zeile 108 - 123) soll dargestellt werden, was im Game Service passiert, wenn ein:e Spieler:in im Chat den Befehl /wuerfeln aufruft.

```
game.post('/wuerfeln',(req,res)=>{

  if(req.body.spielerID == spielTisch.aktSpieler.token){
    let x = wuerfelWerte[Math.floor((Math.random() * wuerfelWerte.length))]
    console.log("wuerfel wert: "+x)

    spielTisch.setAktWuerfelWert(x)
    res.status(200).send({"messageToOne": "Du hast "+ spielTisch.aktWuerfelWert + "gewürfelt",
    "messageToAll": "Spieler "+spielTisch.aktSpieler.spielername+" würfelt gerade."})
  }
   else{
     res.status(401).send({"messageToOne":"Du bist gerade nicht an der Reihe! "+
  spielTisch.aktSpieler.spielername+ " ist an der Reihe!"})
  }
})
```

In der if-Abfrage wird zunächst überrüft, ob die/der Spieler:in überhaupt an der Reihe ist. Falls dies der Fall ist, wird "gewürfelt", also ein zufälliger Wert aus dem Array der möglichen Würfelergebnisse festgelegt und diese der/dem Spieler:in mitgeteilt (alle anderen Sieler:innen erhalten hingegen nur den Hinweis, dass gerade gewürfelt wird); falls nicht (else), wird der/dem Spieler:in mitgeteilt, dass sie/er gerade nicht an der Reihe ist.

4.3 Google Login (extern)

4.3.1 Beschreibung

Der Google Login Service ist dafür zuständig, dass sich die Spieler:innen mit ihrem Google Konto einloggen können. Der in dem Konto angegebene Vorname wird automatisch als Spielername im Chat verwendet. Es gibt bislang keine alternative Möglichkeit, sich auf dem System anzumelden.

5 KOMMUNIKATION

Während der Chat-Service socket-basiert kommuniziert, läuft die Kommunikation der zwischen Chat-Service und Game-Service über REST-Abfragen. Die Kommunkationsarchitektur des Systems wird in der folgenden Skizze dargestellt.

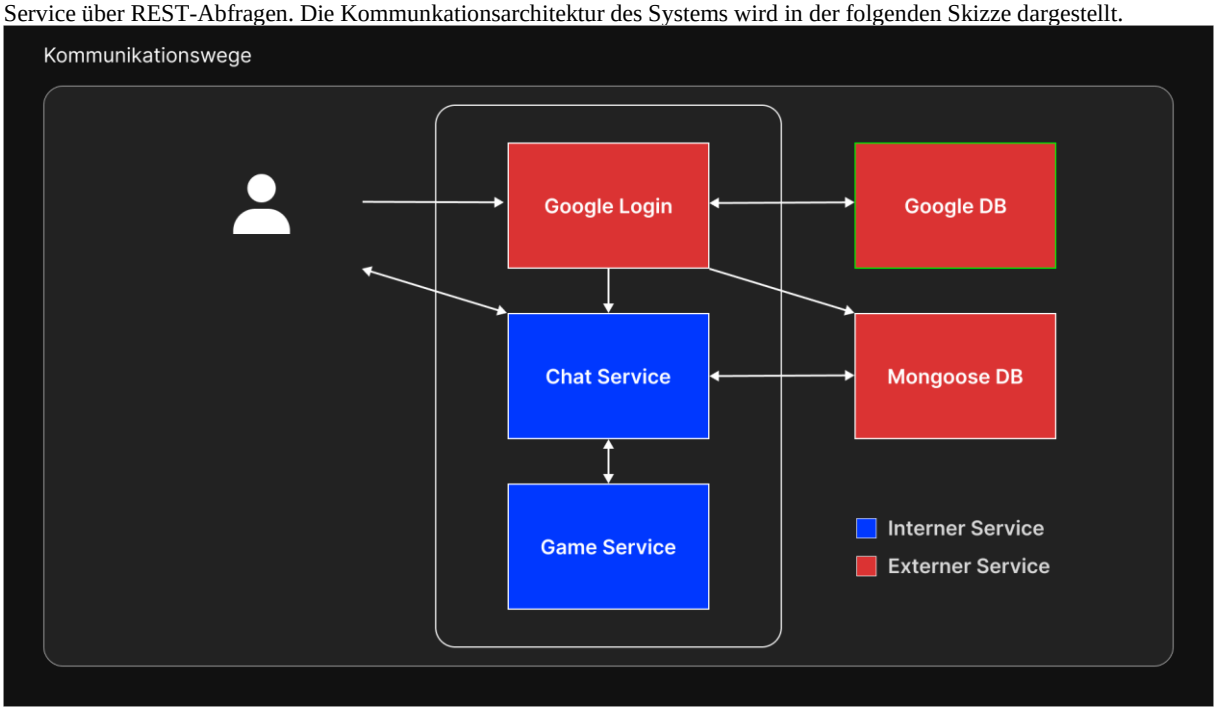

Abb.3: Kommunikationswege

Zunächst loggt sich die/der Nutzer:in über den Google Login in das System ein, wobei die Login-Daten mit der Google-Datenbank abgeglichen werden. War der Login erfolgreich, wird ein User Objekt in der Mongoose Datenbank angelegt und die/der Nutzer:in zum Chat weitergeleitet. Sie/Er kann dann mit dem Chat interagieren und über den Chat das Spiel starten. Hierbei wird ein Spieler Objekt über die spieler.js im Game angelegt. Auch die Interaktion mit dem Game Service läuft für die/den Nutzer:in immer über den Chat-Service, der mit dem Game-Service kommuniziert.

TEIL III: ANLEITUNG

6 INSTALLATION & START

Um das Projekt verwenden zu können, bedarf es nur dreier einfacher Schritte:

- Fügt die env-Dateien unter folgenden Verzeichnissen ein: ./Chat und ./GoogleLog/config
- Navigiert in der Konsole in das Projektverzeichnis und gebt den Befehl docker-compose up --build ein.
- Ruft im Browser die Adresse "localhost:3005" auf. Ihr dürftet jetzt direkt zum Login-Fenster geleitet worden sein.

7 BEFEHLE FÜR SPIELAKTIONEN

Aktion	Befehl	Parameter
Spiel starten	/start	
Würfeln	/wuerfeln	
Würfelzahl behauten	/behaupten <Parameter>	31, 32, 41, 42, …, 44, 55, 66, 21
Behauptung anzweifeln	/challenge	

Punkte aller Spieler anzeigen /aktPunkte

Ein Sonderfall stellt der Befehl "/aktStand" dar, der alle aktuellen Werte, also Würfelzahlen und Spielerstatus, des Spieltischs zurückgibt. Dieser Befehl ist lediglich für debug-Zwecke gedacht und nicht für die Verwendugn im Spiel vorgesehen.

TEIL IV: REFLEXION UND AUSBLICK

8 RÜCKBLICK

Alles in allem wurden die Modulanforderungen, die an die Projekte gestellt wurde nach bestem Wissen und Gewissen umgesetzt. Es wurde lediglich auf die Verwendung von Pull-Requests, sowie das Aufteilen in unterschiedliche Branches verzichtet, was auch der Arbeitsweise des Projektteams geschuldet war.

Zum einen machte eine themenweise Aufteilung der Aufgaben das Arbeiten in unterschiedlichen Branches nicht notwendig, da es nur wenig Konflktpotential im Code gab, zum anderen wurde auch viel gemeinsam, also mit Screensharing oder sogar vor Ort, am Code gearbeitet, wodurch auch nicht alle Projektmitglieder in der Commit History auftauchen. Um die Arbeitsbeteiligungen dennoch nachvollziehbar zu machen, wurden diese im Anhang I noch einmal im Detail aufgeschlüsselt.

9 AUSBLICK

Kurzfristig gesehen wären die nächsten Schritte des Projekts offensichtlicherweise das Deployment des Projekts, sowie Verbesserung bezüglich des Gameplays, beispielsweise ein verbessertes Frontend mit Buttons anstelle von Chatbefehlen. Auch eine richtige Spielerhistorie würde das Projekt deutlich aufwerten, da hier weitere Funktionen wie Highscores oder Statistiken möglich wären.

Auf lange Sicht könnte das Projekt insoweit erweitert werden, dass weitere textbasierte Spiele hinzugefügt werden. Dies könnte eine Sammlung verschiedener Karten- und Würfelspiele beinhalten, die dann beispielsweise über /start <Spiel> gestartet werden könnten.

ANHANG I: ARBEITSMATRIX

Hinweis: Die Tätigkeiten innerhalb der Aufgabenfelder waren unterschiedlich aufwändig. Der Gesamtanteil ergibt sich aus der geschätzten Gewichtung der Tätigkeiten.

A1.1: THEMENFINDUNG UND KONZEPTION

Zur Themenfindung und ersten Projektentwürfen wurden Teammeetings abgehalten, bei denen sich alle Teammitglieder in gleichem Maße beteiligten.

A1.2: PRAKTISCHE UMSETZUNG

Game

Abdurrahman Karakan	≈ 60%	Programmierung (≈ 80%)
Yasha Müller	≈ 40%	Pseudocode & Struktur (≈ 100%), Programmierung (≈ 20%)

Chat

Abdurrahman Karakan	≈ 70%	Programmierung (≈ 90%)
Yasha Müller	≈ 30	Struktur (≈ 100%), Programmierung (≈ 10%)

Login & Datenbank

Anton Berg	100%	komplette Implementierung

A1.3: VERSCHRIFTLICHUNG

Dokumentation

Yasha Müller	≈ 70%	Niederschrift, Abbildungen (Abb.1, Abb.2), Formatierung
Abdurrahman Karakan:	≈ 15%	Notizen, Abbildungen (Abb.3)
Anton Berg	≈ 15%	Notizen, Niederschrift

Weitere Details zum Arbeitsprozess am Code finden sich in der Commit History.

ANHANG II: BUGLIST

In der folgenden Liste werden die Bugs aufgeführt, die bisher entdeckt, aber noch nicht gefixt werden konnten.

- **Chatsystem:** Befehle, die nicht korrekt geschrieben oder angewendet werden, werden nicht als solche erkannt und als Chatnachricht ausgegeben, anstatt der/dem Nutzer:in eine Rückmeldung über die fehlerhafte Eingabe zu geben.
- **Chatsystem:** Illegale (nicht den Spielregeln entsprechende) Aktionen sind ausführbar, solange die/der Spieler:in gerade an der Reihe ist.
    - Spieler:innen können den Befehl /challenge ausführen, nachdem sie bereits gewürfelt haben.
    - Der Befehl /behaupten nimmt momentan noch jeden beliebigen Integer-Wert entgegen.
    - Wird der Befehl /aktStand aufgerufen, bevor das Spiel gestartet wurde, stürzt das Spiel ab.