HTL - Perg
Höhere Abteilung für Informatik

Diplomarbeit

ScrumPoker

Projektteam: Marcel Hochgatterer
Michael Klaus
Manuel Mayrhofer
Niklas Schnabel

Projektbetreuer: Prof. Ing. Dominik Raffetseder, MSc

In Zusammenarbeit mit Porsche Informatik
Betreuer Herr Markus Watzl

Bearbeitungszeitraum: 01.10.2024. – 03.04.2025.

HTL Perg | Machlandstr. 48 | A-4320 Perg | Tel.: +43 (0) 7262 53926

Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.
Diese Versicherung umfasst auch in der Arbeit verwendete bildliche Darstellungen, Tabellen, Skizzen und Zeichnungen.

Unterzeichner Marcel Hochgatterer

Datum und Uhrzeit 02.04.2025, 15:37 (GMT+02:00)

ID Austria

Dieses Dokument ist digital signiert

Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr. 910/2014 vom 23. Juli 2014 ("EIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.

Informationen zur Prüfung der elektronischen Signatur finden Sie unter: **https://www.signaturpruefung.gv.at**

Unterzeichner Michael Klaus

Datum und Uhrzeit 03.04.2025, 11:05 (GMT+02:00)

ID Austria

Dieses Dokument ist digital signiert

Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr. 910/2014 vom 23. Juli 2014 ("EIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.

Informationen zur Prüfung der elektronischen Signatur finden Sie unter: **https://www.signaturpruefung.gv.at**

Signiert von: Manuel Mayrhofer

Datum: 02.04.2025 15:30:07

A-TRUST

Dieses Dokument ist digital signiert!

Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.

Prüfinformation: Informationen zur Prüfung der elektronischen Signatur finden Sie unter: www.a-trust.at/pdf

Unterzeichner Niklas Schnabel

Datum und Uhrzeit 03.04.2025, 10:03 (GMT+02:00)

ID Austria

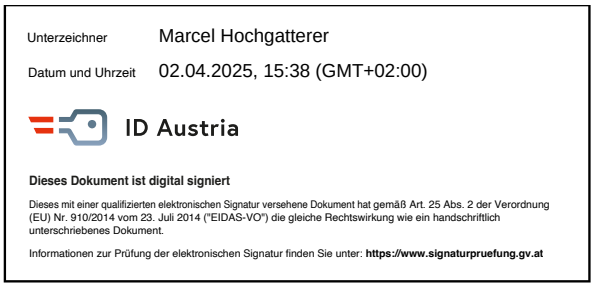
Dieses Dokument ist digital signiert

Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr. 910/2014 vom 23. Juli 2014 ("EIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.

Informationen zur Prüfung der elektronischen Signatur finden Sie unter: **https://www.signaturpruefung.gv.at**

Gendererklärung

Im Sinne der besseren Lesbarkeit werden in dieser Diplomarbeit personenbezogene Bezeichnungen, die sich zugleich auf Frauen und Männer beziehen, generell nur in der im Deutschen üblichen maskulinen Form angeführt. Dies soll jedoch keinesfalls eine Geschlechterdiskriminierung oder eine Verletzung des Gleichheitsgrundsatzes zum Ausdruck bringen.


Unterzeichner Marcel Hochgatterer

Datum und Uhrzeit 02.04.2025, 15:38 (GMT+02:00)

ID Austria

Dieses Dokument ist digital signiert

Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr. 910/2014 vom 23. Juli 2014 ("EIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.

Informationen zur Prüfung der elektronischen Signatur finden Sie unter: **https://www.signaturpruefung.gv.at**


Unterzeichner Michael Klaus

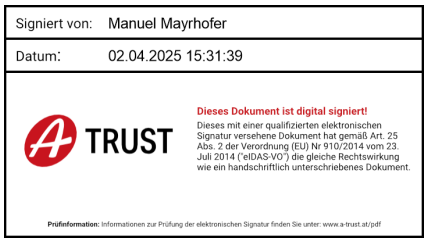
Datum und Uhrzeit 03.04.2025, 11:08 (GMT+02:00)

ID Austria

Dieses Dokument ist digital signiert

Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr. 910/2014 vom 23. Juli 2014 ("EIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.

Informationen zur Prüfung der elektronischen Signatur finden Sie unter: **https://www.signaturpruefung.gv.at**


Signiert von: Manuel Mayrhofer

Datum: 02.04.2025 15:31:39

A-TRUST

Dieses Dokument ist digital signiert!

Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.

Prüfinformation: Informationen zur Prüfung der elektronischen Signatur finden Sie unter: www.a-trust.at/pdf


Unterzeichner Niklas Schnabel

Datum und Uhrzeit 03.04.2025, 10:06 (GMT+02:00)

ID Austria

Dieses Dokument ist digital signiert

Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr. 910/2014 vom 23. Juli 2014 ("EIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.

Informationen zur Prüfung der elektronischen Signatur finden Sie unter: **https://www.signaturpruefung.gv.at**

Danksagung

Wir bedanken uns bei allen Personen der HTL-Perg und der Porsche Informatik, die uns dabei unterstützt haben, diese Diplomarbeit umzusetzen.
Besonders möchten wir unseren Diplomarbeitsbetreuer Herrn Professor Ing. Dominik Raffetseder, MSc erwähnen, welcher uns immer eine große Hilfe bei technischen und organisatorischen Fragen war.
Des Weiteren bedanken wir uns bei unserem Auftraggeber Markus Watzl und unserem Betreuer Schmidt Rainer, welche uns bei der Umsetzung der Anwendung stets beiseite gestanden sind.
Einen Dank richten wir auch an unsere Familien und Freunde, welche uns stets unterstützt und motiviert haben.

Zusammenfassung

Aufgabenstellung

Die Aufgabe bei unserem Projekt ScrumPoker besteht darin, ein auf die Porsche Informatik zugeschnittenes Tool zu entwickeln, bei welchem man das ScrumPoker Spiel ohne Einschränkungen spielen kann und so die Aufwandsschätzung für ein Arbeitspaket in einem Sprint optimiert. Dieses ScrumPoker Spiel hebt sich von anderen ab, indem es keinerlei Beschränkungen gibt, welche die Anzahl der Spieler betrifft, sowie die sehr einfache Bedienung unseres Spiels, welches ein sehr schnelles Starten eines Spiels erlaubt und jedem Mitarbeiter ermöglicht, einer Runde schnell und unkompliziert beizutreten. Zusätzlich bietet unser Projekt die Möglichkeit, nachdem man eine Runde beendet hat, die Ergebnisse der einzelnen Runden als PDF zu exportieren und erspart so die aufwendige Arbeit, die gesamten Runden mit den Ergebnissen selber mitprotokollieren zu müssen. Dieses Schätztool bietet der Porsche Informatik eine einfachere Aufwandsschätzung als zuvor, indem sie nicht mehr abhängig sind von online verfügbaren Schätzpokersystemen, bei welchen sie begrenzte Möglichkeiten hatten, der Einstieg in eine Runde kompliziert war und sie nicht mit dem gesamten Team schätzen konnten, da die Anzahl an Mitspielern beschränkt war.

Realisierung

Die Realisierung erfolgt durch die Entwicklung einer Angular-Website und einer Android-App, bei welchen man in einen gemeinsamen Raum beitreten kann und so miteinander spielen kann. Die Grundlage dafür ist die Java Spring Boot REST-API, welche die Logik des Frontends bereitstellt und die Daten der Räume und der User in unserer PostgreSQL-Datenbank speichert. Wenn man die Analysedaten abruft, wird die Python-API aufgerufen und diese erstellt dann ein PDF-Dokument mit den wichtigsten Daten, welche während der Runde gespeichert wurden, und ermöglicht so einen guten Überblick über die gespielte Runde. Die Herausforderung beim Entwickeln des Frontends ist, dass die Bedienung sowohl in der App als auch auf der Website sehr unkompliziert und intuitiv funktionieren soll. Dies wird durch regelmäßige Absprache mit unserem Auftraggeber sichergestellt.

Ergebnis

Das Ergebnis ist eine voll funktionsfähige Webapplikation und Mobile App, in welcher man ein ScrumPoker-Spiel mit einem verteilten Team durchführen kann, seine Arbeitspakete gemeinsam mit seinem Team einschätzen kann und so zu einer qualifizierten gemeinsamen Meinung bezüglich des Arbeitsaufwandes kommt. Wir haben unserem Auftraggeber den gesamten Code sowie einen Docker-Container, in welchem die gesamte lauffähige Applikation läuft, übergeben. Dazu haben wir auch noch ein Script bereitgestellt, mit welchem unser Auftraggeber die Anwendung sehr einfach starten kann und es zu keinen Problemen bei der Installation unserer Software kommt. Wir haben das Produkt umfangreich getestet, um zu gewährleisten, dass keine unerwarteten Fehler auftreten.

Abstract

Task

The task in our project ScrumPoker was to develop a tailored tool for the company Porsche Informatik. The desired result is an application where our client can play a ScrumPoker game without any restrictions and plan their Scrum sprints. This ScrumPoker game stands out from others because there are no restrictions depending on the number of players in a room, as well as the simple handling of the app. Because of this simple handling, it is possible to start a game very fast and to start voting as quickly as possible, because every team member can join very easily and does not necessarily have to configure his user. Additionally, our project offers the opportunity to download all the statistics in PDF format. This is very useful for our client because it saves the work that would be needed to protocol all the answers made. This guessing tool offers our client a much easier effort estimation than before, as they are no longer dependent on guessing tools available online, where they had limited options, joining a room was complicated, and they could not guess with their whole team when there were too many persons involved in the project.

Implementation

The implementation was made through the development of an Angular website and an Android mobile app where you can join a room together, independent of whether you are on the website or in the ScrumPoker app. The foundation of this is the Java Spring Boot REST-API, where the logical part of the application is based, and the data of the users and the rooms gets saved in the PostgreSQL database. If the statistics of the round are requested, the Python API creates a PDF document with the relevant data that was saved during the round and offers a great overview of the played rounds. The biggest challenge in our project was the requirement for the frontend to be very uncomplicated and very intuitive to use, both on the website and in the mobile app. This requirement was ensured through regular meetings with our client.

Result

The result is a fully functioning web application and mobile application in which a ScrumPoker game can be carried out. It is possible for our client to guess the estimated effort that one work package will take with the whole team. Like that, our client gets a qualified opinion referring to the needed work effort. We handed the total code of our application as well as a Docker container with the running application to our client. Additionally, we provided a script that starts the total application so that it is easy to start the project and no problems occur starting the software. We tested the product extensively to ensure no unexpected errors occurred.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

1.1.1 Geschäftlicher Hintergrund

Der Hintergrund unserer Diplomarbeit wurde geprägt durch die immer häufigere Nutzung des Vorgehensmodells Scrum in Projekten bei unserem Auftraggeber, der Porsche Informatik, und die damit verbundene Schwierigkeit, den Aufwand von einzelnen Arbeitspaketen richtig und in Absprache mit dem gesamten Team zu schätzen. ScrumPoker ist für agile Teams sinnvoll, da dadurch realistische Zeitpläne erstellt werden können, das Risiko minimiert wird und Ressourcen effizient genutzt werden können. Durch ScrumPoker werden Entscheidungen schneller getroffen, da man durch den strukturierten Prozess eine Schätzung abgeben und man sich im Zuge dessen mit seinem Team besprechen muss. Dadurch wird die Planung verbessert. Durch genaue Schätzungen kommt es ebenso zu weniger Verzögerungen bei Softwareprojekten. [1] Die herkömmlichen, frei verfügbaren Systeme stellten unseren Auftraggeber nicht zufrieden, da diese entweder kompliziert zu benutzen sind, eine Obergrenze von möglichen Mitspielern haben oder zu viele unnötige Funktionen beinhalten. Diese verlangsamen den Schätzvorgang.

1.1.2 Projektauslöser

Zur Schätzung hat man bisher herkömmliche Systeme des ScrumPoker-Spiels verwendet, jedoch stellte keines dieser Spiele unseren Auftraggeber zufrieden. Daher haben wir im Umfang der Diplomarbeit ein ScrumPoker-Spiel entwickelt, welches genau auf die Bedürfnisse der Porsche Informatik abgestimmt ist. Ein verteiltes Team kann dadurch ohne komplizierte Anmeldung sowie ohne eine Obergrenze von Mitspielern schnell miteinander abschätzen, wie lange sie für ein Arbeitspaket brauchen und das auch mit dem gesamten Team. Zusätzlich ist unser Auftraggeber so nicht mehr abhängig von frei verfügbaren Spielen und kann bei Bedarf die Anwendung noch einmal ändern für etwaige spätere Anforderungen.

1.2 Zielsetzung

1.2.1 Geschäftsziel

Das Geschäftsziel unserer Diplomarbeit besteht darin, die Genauigkeit und Schnelligkeit der Schätzung eines Arbeitspakets zu verbessern, sodass jeder, der an einem Projekt mitarbeitet, sehr einfach abstimmen kann, wie viel Aufwand das Arbeitspaket seiner Meinung nach haben wird. Das alles, ohne sich kompliziert registrieren oder anmelden zu müssen. Außerdem soll eine Möglichkeit bestehen, dass der Scrum Master nicht dauerhaft mitprotokollieren muss, sondern man sich, wenn man fertig ist, ein Dokument mit den Antworten aller Mitglieder und dem Durchschnitt bei jeder Abfragerunde speichern kann und somit der Vorgang der Schätzung noch weiter beschleunigt wird.

1.2.2 Projektvorgehensziele

Zu den wichtigsten Zielen bei der erfolgreichen Entwicklung von ScrumPoker gehören:

1. **Abstimmung der Anforderungen**
   Abstimmung mit Porsche Informatik, welche Anforderungen sie an das ScrumPoker System stellen, basierend auf ihren Zielen.

2. **Technologie Auswahl**
   Auswahl der verwendeten Technologien für die Datenbank und die Entwicklung von Frontend und Backend, in Abstimmung mit Porsche Informatik.

3. **Implementierung der Spielmechanik** Implementierung der Spielmechanik, um das System spiel-tauglich zu machen.

4. **Implementierung der Auswertung** Implementierung der Auswertung der gespielten Runden, um die Dokumentation zu vereinfachen.

5. **Testung** Umfassende Testung der Applikation und der REST-API, um die Funktion und Anforderungen sicherzustellen.

1.3 Projektinhalt - Überblick

ScrumPoker ist eine digitale Lösung zur Aufwandsschätzung bei Scrum. Die Anwendung nutzt ein Kartenspielprinzip, in Verbindung mit der agilen Scrum-Methode. Auf diese Weise finden Teams eine gemeinsame Einschätzung des Aufwands, basierend auf den individuellen Bewertungen. Dieses Verfahren ermöglicht eine verbesserte Planung von Prozessen und Arbeitspaketen. Unsere Diplomarbeit beinhaltet eine Web- und App-Anwendung, welche es den Teammitgliedern ermöglicht, unabhängig vom Standort an einem Sprint-Planning-Meeting teilzunehmen. Zusätzlich zu den Schätzungsrunden erstellt das System eine PDF-Auswertung, die die Ergebnisse übersichtlich zusammenfasst.

1.3.1 Funktionsweise von ScrumPoker

Die ScrumPoker-Anwendung ermöglicht eine strukturierte Schätzung von Arbeitspaketen unter Verwendung der Fibonacci-Zahlen. Ein Vorteil dieser Zahlen besteht im exponentiellen Wachstum. Die kleineren Arbeitspakete können von Teams genauer eingeschätzt werden, weshalb die Zahlensprünge zu Beginn kleiner sind. Bei größeren Aufgaben wird es schwieriger, eine präzise Schätzung abzugeben. Deshalb sind hier die Sprünge zwischen den Zahlen größer. Innerhalb der Anwendung gibt es zwei unterschiedliche Rollen: den Scrum Master und die Teammitglieder. Nachdem eine Frage gestellt wurde, können die Teammitglieder anonym eine Schätzung abgeben. Die Schätzung wird in weiterer Folge visualisiert. Nun können die Ergebnisse innerhalb des Teams besprochen werden, um am Ende eine einheitliche Schätzung zu erhalten. Das Programm synchronisiert dabei alle Aktionen und Abstimmungen in Echtzeit, was die Durchführung direkt während eines Meetings ermöglicht. Nach Beendigung der Sitzung kann ein PDF-Bericht für weitere Planungs- und Analysezwecke heruntergeladen werden.

1.3.2 Scrum Master

Der Scrum Master hat während einer Sitzung folgende Aufgaben und Funktionen:

- **Erstellen und Verwalten von Schätzrunden:** Der Scrum Master hat die Möglichkeit, einen 'Raum' zu erstellen und die Teammitglieder mittels eines Links in diesen Raum einzuladen. Sobald alle Teilnehmer anwesend sind, kann der Scrum Master eine Frage stellen, wodurch die Schätzung gestartet wird.
- **Steuerung des Ablaufs:** Während der Sitzung kann der Scrum Master entscheiden, ob über die Schätzungen diskutiert wird, eine neue Frage gestellt wird oder ob dieselbe Frage

nochmals abgeschätzt werden soll. Am Ende kann er den PDF-Bericht herunterladen und zu weiteren Analysen und Planungen verwenden.

- **Verwaltung von Teilnehmern:** In einem Raum hat der Scrum Master das Recht, ein Mitglied aus diesem Raum zu entfernen oder aber auch zu einem Scrum Master zu befördern. Bei einer Beförderung verliert der derzeitige Scrum Master seine Rechte und erhält die Rolle 'Teammitglied'.

1.3.3 Teammitglieder

Teammitglieder können, mittels eines Links, dem dazugehörigen Raum beitreten. Ihre Funktionen umfassen:

- **Interaktive Schätzung:** Nach dem Stellen einer Frage kann jedes Teammitglied verdeckt eine Karte auswählen. Die Ergebnisse werden nach der Abstimmung visualisiert.
- **Feedback und Diskussion:** Nach jeder Schätzrunde können die Teilnehmer ihre Schätzung erläutern und gegebenenfalls in einer Wiederholungsrunde anpassen.

1.3.4 Ergebnisanalyse und PDF-Berichte

Ein weiterer Bestandteil ist die automatische Ergebnisanalyse. Nach Abschluss einer Schätzrunde werden die Daten zu jedem Arbeitspaket dokumentiert. Dabei können die Schätzungen jedes einzelnen Teammitglieds zurückverfolgt werden. Aufgelistet werden somit alle Teilnehmer sowie die von ihnen ausgewählten Karten, die sie in der Runde verwendet haben. Eine Runde speichert zusätzlich die gestellte Frage und wie viele Durchläufe benötigt wurden. Jeder Eintrag wird mit einem Zeitstempel versehen, um den zeitlichen Ablauf nachvollziehbar zu machen. Am Ende werden alle Ergebnisse und Analysen zusammengefasst und können als PDF-Datei exportiert werden.

1.4 Projektumfeld

1.4.1 Projektteam

Wir sind ein Team mit vier Schülern der HTL Perg aus der Abteilung für Höhere Informatik.

Niklas Schnabel

Projektleiter

Michael Klaus

Projektmitglied

Marcel Hochgatterer

Projektmitglied

Manuel Mayrhofer

Projektmitglied

1.4.2 Auftraggeber

Unser Auftraggeber ist die Porsche Informatik Gesellschaft m.b.H. mit Hauptsitz in Salzburg. Weitere Standorte befinden sich in Hagenberg und in Wien. Das Unternehmen ist auf die Digitalisierung des Automobilhandels spezialisiert und entwickelt Softwarelösungen, die weltweit von Autohäusern genutzt werden.

PORSCHE
INFORMATIK

Abbildung 1: Porsche Informatik Logo
Quelle: https://tr.linkedin.com/company/porsche-informatik-fr

1.4.3 Betreuung

Prof. Ing. Dominik Raffetseder, MSc

Kontakt: `d.raffetseder@htl-perg.ac.at`

Unsere Diplomarbeit wird von Herrn Professor Raffetseder betreut, der uns im Schuljahr 2023/24 in dem Gegenstand Webprogrammierung und Mobile Computing (WMC) unterrichtet hat. Durch seine fundierten Kenntnisse in den Bereichen Datenbanken, Mobile Computing, Projektentwicklung, Webprogrammierung und Java ist er für uns eine wertvolle Unterstützung bei der Planung und Umsetzung unserer Diplomarbeit.

Abbildung 2: Prof. Ing. Dominik Raffetseder, MSc
Quelle: https://www.htl-perg.ac.at

2 Theoretische und fachpraktische Grundlagen und Methoden

2.1 Verwendete Konzepte und Architekturen

2.1.1 RESTful APIs

Eine REST-API (Representational State Transfer Application Programming Interface) ist eine Architektur für die Kommunikation zwischen Systemen über das HTTP-Protokoll. Sie ermöglicht es, Daten und Funktionalitäten von Servern für Anwendungen bereitzustellen. Die API dient als grundlegendes Kommunikationsmittel zwischen Backend und Frontend.

2.2 Grundlegende Fachbegriffe

2.2.1 Scrum

Scrum ist ein agiles Rahmenwerk (Framework) für Projektentwicklung. Durch definierte Rollen und Aufgaben sorgt Scrum dafür, dass jeder klare Aufgaben hat und man eine gut strukturierte Projektführung hat. Ursprünglich ist Scrum aus der Softwareentwicklung entstanden, wird aber immer mehr auch in anderen Bereichen verwendet. Das Ziel von Scrum ist, dass man schnell kundenspezifische Produkte erstellt. Scrum arbeitet sehr kundennah. Das heißt, der Kunde ist bei manchen Meetings dabei und kann das aktuelle Ergebnis immer begutachten.

Rollen in Scrum

- Product Owner
  Der Product Owner oder auch Project Owner genannt, hat die Aufgabe, die Kundenwünsche wie auch die Unternehmensbedingungen zu erfüllen. Er kommuniziert mit den Kunden und mit den Entwicklern. Der PO (Product Owner) sorgt formuliert die langfristige Vision des Vorhabens. Der Product Owner ist nur eine Person. Er benutzt ebenfalls das Backlog und hat dann die Aufgabe, die Pakete richtig zu priorisieren.

- Scrum Master

  Der Scrum Master hat die Aufgabe den Produkt Owner zufrieden zustellen und das Entwicklerteam korrekt zu leiten. Er leitet das ScrumPoker Spiel.

- Entwickler

  Das Entwicklerteam sorgt dann für die Umsetzung des Projektes. Ein Entwickler wird oder kann sich eine Aufgabe nehmen. Die Entwickler sind bei einigen Meetings dabei. Die Entwickler spielen auch im ScrumPoker mit.

Meetings in Scrum

- Sprint

  Der Sprint ist eines der wichtigsten Elemente in Scrum. Ein Sprint ist ein vorgegebenes Zeitintervall. In der Regel kann ein Sprint zwischen einer und vier Wochen liegen. Ebenso muss jeder Sprint gleich lange dauern. Wenn das festgelegt ist, müssen dann die festgelegten Aufgaben in der festgelegten Dauer geschafft werden. In der Abbildung 3 werden alle Meetings grafisch veranschaulicht.

- Sprint Planning

  Das Sprint Planning Meeting dauert in etwa acht Stunden und ist das Meeting wo man den Sprint plant. Es nehmen alle an Scrum beteiligten Person am Meeting teil. In diesem Meeting werden ebenfalls die Aufgaben priorisiert. Das Team hat dann die Aufgabe, das Ziel im festgelegten Intervall zu erledigen.

- Daily Scrum

  Der Daily Scrum ist grundsätzlich für die Entwickler gedacht. Der Product Owner oder der Scrum Master können optional teilnehmen. Die Entwickler können miteinander kommunizieren. Wenn jemand zum Beispiel Hilfe bei etwas braucht. Der Daily Scrum wird jeden Tag gemacht und die Dauer entspricht in etwa 15 Minuten.

- Review

  Das Review Meeting findet am Ende des Sprints statt. Hier wird dann die Arbeit präsentiert. Bei diesem Meeting kann dann auch der Kunde oder interessierte Stakeholder beitreten. Dauer des Meetings ist in der Regel vier Stunden.

- Retrospektive

  Schließlich kommt das Retrospektiv Meeting. In diesem Meeting wird alles besprochen, was gut oder schlecht gelaufen ist und wie man den Sprint verbessern kann. Dauer des Meetings ist in etwas drei Stunden.

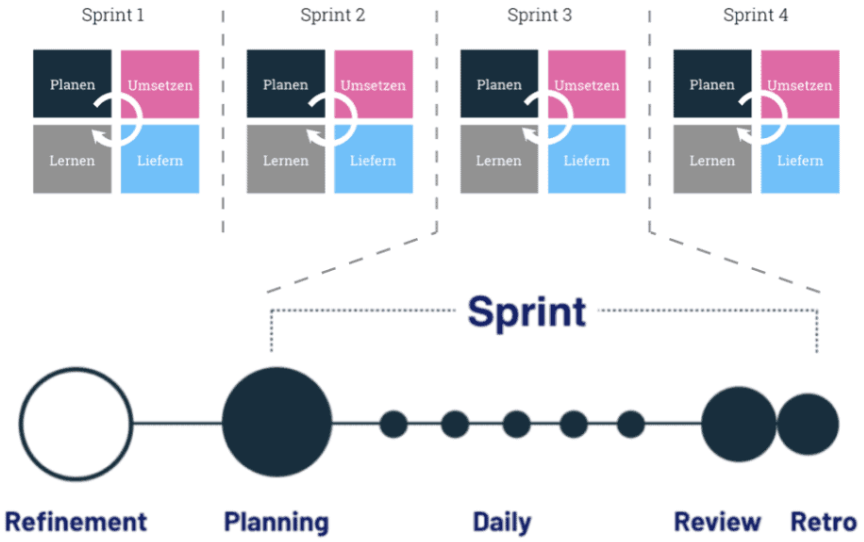


Abbildung 3: Scrum Meetings
Quelle: https://www.pureconsultant.de/de/scrum/scrum-poker/

Elemente in Scrum

- Product Backlog
  Das Product Backlog ist eine Liste aller anstehender Aufgaben. Die Liste ist priorisiert. Die Aufgaben werden oft in einem Kanban Board dargestellt.
- Product Backlog Item (User Stories)
  Eine User Story ist dann eine einzelne Aufgabe in einem Backlog. Das können zum Beispiel konkrete Todos sein. Die Dauer einer User Story wird dann geschätzt. Früher hat man die diese mit Fibonacci Spielkarten in der Realität geschätzt. Heutzutage verwendet man online Tools (ScrumPoker).
- Sprint Backlog
  Das Sprint Backlog enthält alle Aufgaben, die das Entwicklerteam im aktuellen Sprint umsetzt. Es entsteht im Sprint Planning, bei dem Product Owner und Team gemeinsam entscheiden, welche Product Backlog Items übernommen werden. In der Abbildung 4 wird der Workflow veranschaulicht. Die Verantwortung für das Sprint Backlog liegt ausschließlich beim Entwicklerteam.

ScrumPoker

ScrumPoker, auch bekannt als Planning Poker, ist eine Schätzmethode zur Bewertung des Aufwands von Aufgaben im Product Backlog. Das Spiel dient dazu, eine gemeinsame Einschätzung für das Team über die Komplexität und den Aufwand einer User Story zu bekommen. Ein

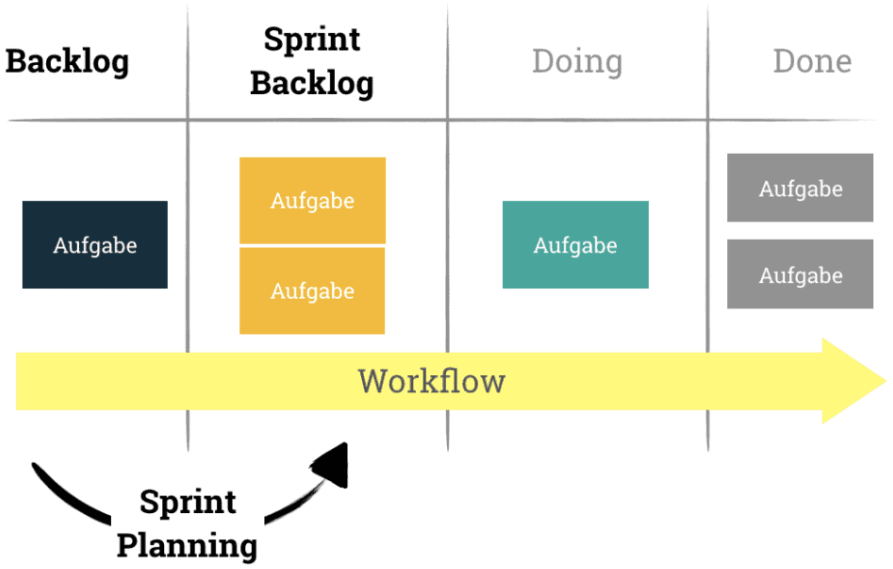


Abbildung 4: Scrum Backlog
Quelle: https://www.pureconsultant.de/de/scrum/scrum-poker/

Vorteil ist, dass man in der Regel herausfindet, wie lange man für eine User Story braucht und ein anderer Vorteil ist, dass die verschiedenen Entwickler über die verschiedenen User Stories informiert werden. Jedes Mitglied bekommt dann Karten, bei denen die Fibonaccizahlen abgebildet sind. Der Scrum Master stellt dann eine Frage über einen Story Point und man schätzt dann mit den Karten die Dauer oder Komplexität dieses Story Points.[2]

2.3 Verwendete Technologien

2.3.1 Angular

In unserer Arbeit wird Angular für das Frontend verwendet. Angular ist ein weit verbreitetes Framework zur Entwicklung von Webanwendungen, das von Google entwickelt und gepflegt wird. Es basiert auf TypeScript, einer statisch typisierten Programmiersprache, die auf JavaScript aufbaut. Angular ermöglicht die Erstellung von dynamischen, skalierbaren und leistungsstarken Single-Page Applications (SPAs) und verfügt über eine umfangreiche Sammlung an Werkzeugen und Funktionen für die Entwicklung moderner Webanwendungen. [3]

2.3.2 Ionic

Für unsere Android/iOS-App verwenden wir Ionic. Ionic ist ein Open-Source-Webframework für die Erstellung von Hybrid-Apps und Progressive Web Apps. Die Basis bildet dafür HTML5,

SASS, JavaScript/TypeScript. Ionic verfügt über eine große Menge an Bibliotheken und UI-Komponenten, die das Programmieren erleichtern. [4]

2.3.3 Python

Um die Daten nach den Spielen in einem schönen PDF herunterladen zu können, haben wir uns für Python entschieden. Python ist eine sehr bekannte und gut verwendete Programmiersprache. Sie zeichnet sich durch ihre einfache Syntax und eine Vielzahl an Bibliotheken aus. Python wird gerne zur Datenvisualisierung verwendet. [5]

2.3.4 Java

Für unsere REST-API haben wir uns für JAVA entschieden. Java ist eine objektorientierte Programmiersprache. In Kombination mit Spring Boot bietet Java umfangreiche Möglichkeiten, eine gute REST-API zu programmieren. Java ist sehr weit verbreitet und besitzt eine große Anzahl an Bibliotheken, die das Programmieren erleichtern. [6]

2.3.5 PostgreSQL

Für die Datenhaltung haben wir uns für PostgreSQL entschieden. PostgreSQL ist ein objektrelationales Datenbankmanagementsystem. PostgreSQL ist Open-Source und wird von der Community kontinuierlich weiterentwickelt. Es erfüllt auch die ACID-Kriterien und bietet dadurch eine hohe Datenintegrität. [7]

2.3.6 Docker

Für die Projektübergabe haben wir uns für Docker entschieden. Docker ist eine Containervirtualisierung. Mit Docker kann man Programme/Projekte virtualisieren, dadurch laufen die Programme nicht mehr direkt auf einem Betriebssystem. [8]

2.3.7 GitLab

Für das Lagern unserer Codes haben wir uns für GitLab entschieden. GitLab ist ein sehr bekanntes Tool, das die Programmierung in großen wie auch in kleinen Teams drastisch verbessert. Mithilfe von Branches ist die gleichzeitige Programmierung einfacher. GitLab bietet neben diesen Funktionen auch organisatorische Hilfen, wie zum Beispiel eine Meilenstein-Liste oder ein Issue Board. [9]

2.3.8 Clockify

Um unsere Arbeitsstunden zu protokollieren, haben wir uns für Clockify entschieden. Clockify ist eines der bekanntesten Zeitaufzeichnungs-Softwares, die es gerade gibt. Es hat eine einfache GUI mit der man sich schnell zurechtfindet und es bietet eine Reihe von Analysefunktionen für das Team. [10]

2.4 Verwendete Entwicklungssysteme

2.4.1 IntelliJ

IntelliJ IDEA ist eine Entwicklungsumgebung für Java, Kotlin, Groovy und Scala. Sie vereinfacht das Programmieren mithilfe von bestimmten Features wie 'Code Completion' oder das Verwalten von vielen verschiedenen Plugins. Die IDEA verfügt über eine 'Community Edition' die gratis ist, und eine 'Ultimate Edition', die kostenpflichtig ist. [11]

2.4.2 Webstorm

Webstorm ist ebenfalls eine IDEA von derselben Firma wie IntelliJ. Der Schwerpunkt dieser IDEA liegt auf der Entwicklung von Websites. Sie unterstützt Programmiersprachen wie HTML, Javascript/Typescript, CSS, aber auch ganze Frameworks wie Angular, React, Vue.js und Node.js. Git zählt auch zu den wichtigsten Funktionen, die unterstützt werden. [12]

2.4.3 Visual Studio Code

Visual Studio Code ist einer der weitverbreitetsten Code-Editoren der Welt. Der Editor ist Open-Source und somit gratis. Er verfügt über eine enorm große Community, die kontinuierlich neue Funktionen entwickelt, die den Editor immer besser machen. Visual Studio Code unterstützt eine Vielzahl an Programmiersprachen von Python bis zu HTML. [13]

2.4.4 Docker Desktop

Mit Docker Desktop wird die Verwaltung von mehreren Docker-Containern enorm vereinfacht. Docker Desktop ist auf Windows, Linux und Mac verfügbar. Die grafische Oberfläche der Applikation ist sehr einfach, aber auch sehr umfangreich. [14]

2.5 Verwendete Bibliotheken und Plugins

Wir verwenden mehrere Plugins und Bibliotheken. Hier werden von jedem Thema die wichtigsten Plugins/Bibliotheken erklärt.

2.5.1 IONIC

FontAwesome

FontAwesome ist eine Sammlung von Vektor-Icons für Webprojekte. Dies ermöglicht das Einfügen von Symbolen per CSS-Klassen, zum Beispiel für Buttons. [15]

RxJS

RxJS (Reactive Extensions for JavaScript) ist eine Bibliothek für die Programmierung mit Observables. Sie hilft bei asynchronen Ereignissen, wie HTTP-Anfragen oder Benutzerinteraktionen. [16]

Jasmine & Karma

Jasmine und Karma werden für das Testen verwendet. Mit Jasmine kann man Unit-Tests schreiben. Karma ist das Gegenstück für die Tests, denn Karma führt diese Tests im Browser aus und gibt ein schnelles Feedback zu dem getesteten Code. [17]

2.5.2 Angular

Angular Core und BrowserModule

Angular Core ist das Herzstück von Angular und stellt grundlegende Funktionen und Mechanismen bereit. Ein Beispiel dafür ist der Component-Dekorator, dieser wird benötigt, um Komponenten zu definieren. [18]

FormsModule

FormsModule unterstützt die Arbeit mit Template-Driven Forms in Angular. Es ermöglicht eine Zwei-Wege-Datenbindung. mit [(ngModel)] und die Prüfung von Benutzereingaben. [19]

2.5.3 Backend

Spring Framework

Spring Framework ist ein Framework zur Entwicklung von Java-Anwendungen. Es bietet Funktionen für Webentwicklung und Datenbankzugriffe. [20]

JDBC

JDBC ist eine Java-API zur Anbindung von Datenbanken. Mit dem PlugIn lassen sich SQL-Befehle direkt aus Java-Code ausführen, um Daten entweder zu lesen oder zu schreiben.[21]

2.5.4 Python

psycopg2

psycopg2 ist ein PostgreSQL-Treiber für Python. Er ermöglicht es, SQL-Abfragen aus Python heraus auf PostgreSQL-Datenbanken auszuführen.

psycopg2 ist ein PostgreSQL-PlugIn für Python. Dieses PlugIn ermöglicht einen Datenbankzugriff, mit dem man SQL-Abfragen direkt aus Python heraus ausführen kann.[22]

reportlab

ReportLab ist eine Python-Bibliothek für die Erstellung von PDF-Dokumenten. Es ermöglicht die Erstellung von Layouts, Tabellen und Text direkt aus dem Python-Code. [23]

3 Planung und Realisierung

3.1 Projektorganisation

Unser Projekt teilt sich in vier einzelne Teile, wodurch jeder von uns seinen eigenen Teilbereich besitzt, für den er verantwortlich ist. Um unsere Meilensteine perfekt umsetzen zu können, wurde die agile Projektmanagement-Methode Scrum verwendet.
Während der Entwicklung wurden zweiwöchige Sprints angewendet. Zu Beginn jedes Sprints erfolgte die Planung der Aufgaben sowie das Definieren der zu erreichenden Ziele. In dieser Phase wurden wöchentlich interne Meetings abgehalten, in denen die Arbeitsfortschritte und Herausforderungen besprochen wurden.
Am Ende jedes Sprints wurden die Ergebnisse dem Auftraggeber präsentiert. Dies wurde online über Microsoft Teams durchgeführt. Mittels dieser Meetings konnte sichergestellt werden, dass die Anwendung der vollsten Zufriedenheit des Auftraggebers entsprach.
Für das Speichern und Austauschen des Codes haben wir GitLab verwendet. Durch diese Plattform konnten wir unseren Code strukturiert verwalten. Ebenso konnte dadurch sichergestellt werden, dass jeder am neuesten Stand war. Durch unsere vordefinierten Merge Requests konnten Änderungen jederzeit nachverfolgt werden. Zusätzlich ermöglichte es uns, bei den Meetings regelmäßige Code-Reviews durchzuführen. Dadurch konnten wir sicherstellen, dass der Code effizient und fehlerfrei bleibt.

3.2 Meilensteine

3.2.1 Backend (Schnabel Niklas)

Meilensteine	**Datum**
Start	01.06.2024
Abschluss der Technologiewahl und Fertigstellung des Prototyps	15.06.2024
Fertigstellung der API-Endpunkte für die Erstellung/Verwaltung von Räumen	30.06.2024
Mechanismus zur Generierung der URLs ist implementiert	07.07.2024
API-Endpunkte für Benutzerregistrierung und -login sind funktionstüchtig	10.07.2024
Mechanismus zur Verwaltung von Benutzeranpassungen ist implementiert	10.07.2024
API-Endpunkt zur Erstellung/Verwaltung von Runden und die Funktionalität zum Hinzufügen/Entfernen von Mitgliedern ist fertigstellt	15.07.2024
API-Endpunkt zum Aufdecken von Karten und Verwaltung des Spielstatus ist implementiert	30.07.2024
Logik zur Berechnung von Durchschnittswerten und API-Endpunkt zum Bereitstellen des Durchschnittswertes ist funktionstüchtig	10.08.2024
Logik zur Analyse und Auswertung der Runden ist entwickelt	15.08.2024
API-Endpunkt zur Bereitstellung von Grafiken und zum Export von PDFs ist implementiert	17.08.2024
Getestet	01.09.2024

Tabelle 1: Projektmeilensteine für das Backend

3.2.2 Datenbank (Klaus Michael)

Meilensteine	**Datum**
Start	01.06.2024
Abschluss der Technologieauswahl und Fertigstellung des Prototyps	15.06.2024
Datenbank Schema ist ausgearbeitet	26.06.2024
Datenbankmodelle und -tabellen für Räume sind erstellt und verknüpft, ebenso die CRUD-Operationen angefertigt	02.07.2024
URLs für Räume werden verwaltet und gespeichert	07.07.2024
Datenbankmodelle für Benutzer und Authentifizierungstokens sind angefertigt	10.07.2024
Tabelle zur Verwaltung von Benutzersitzungen ist angelegt	10.07.2024
Ein Datenbankmodell für Runden ist angefertigt	20.07.2024
Datenbankmodell für Karten und deren Status ist fertig	30.07.2024
Logik zum Entfernen von Benutzern einer Session ist abgeschlossen	05.08.2024
Ein Datenbankmodell zur Speicherung von Durchschnittswerten ist angefertigt	10.08.2024
Ein Datenbankmodell zur Speicherung der Auswertungsdaten ist fertig	17.08.2024
Eine Tabelle zur Verwaltung von Kalendereinladungen ist erstellt	20.08.2024
Getestet	01.09.2024

Tabelle 2: Projektmeilensteine für die Datenbank

3.2.3 Web-Frontend (Mayrhofer Manuel) APP-Frontend (Hochgatterer Marcel)

Meilensteine	**Datum**
Start	01.06.2024
Abschluss der Technologieauswahl und Fertigstellung des Prototyps	15.06.2024
Benutzeroberfläche für die Erstellung von Räumen ist implementiert	30.06.2024
Benutzeroberfläche zum Teilen von URLs ist fertiggestellt	07.07.2024
Benutzeroberfläche zur Benutzerregistrierung und -login ist vorhanden	10.07.2024
Verwaltung von Benutzersitzungen ist möglich	10.07.2024
Benutzeroberfläche zur Verwaltung von Mitgliedern einer Runde ist erstellt (Hinzufügen/Entfernen)	20.07.2024
Anzeige von Berechtigungen ist sichtbar	20.07.2024
Benutzeroberfläche für die Anzeige und Verwaltung der Karten ist erstellt	28.07.2024
Anzeigen und Verwalten von inaktiven Spielern ist möglich	30.07.2024
Benutzeroberfläche zum Anzeigen der Durchschnittswerte ist erstellt	10.08.2024
Benutzeroberfläche zur Auswertung von Runden, zum Anzeigen von Grafiken und das Exportieren dessen ist möglich	17.08.2024
Benutzeroberfläche zur Integration und Verwaltung von Kalendereinladungen funktioniert	20.08.2024
Getestet	01.09.2024

Tabelle 3: Projektmeilensteine für das Frontend

3.3 Projektzeitplan

Die Aufgaben wurden nach einem genauen Zeitplan abgearbeitet, welcher durch die Projektmanagement-Methode Scrum unterstützt wurde. Zur effizienteren Planung der Sprints nutzten wir Microsoft Planner. Außerdem wurden damit unsere User-Stories und Backlogs erstellt, um den Fortschritt regelmäßig überprüfen zu können.

3.3.1 Unser Sprint-Backlog

Unsere Sprints hatten eine Laufzeit von zwei Wochen. Zu Beginn jedes Sprints wurden, auf Grundlage eines Meetings mit dem Auftraggeber, die User Stories erstellt. Diese Stories definierten, welche Ziele erreicht werden sollten. Die User Stories wurden schließlich in Arbeitspakete zerlegt und den Teammitgliedern zugeteilt. Diese Aufgaben sollten bis zum Ende des jeweiligen Sprints abgeschlossen sein.

Als Beispiel zeigt die Abbildung 5 unseren Scrum-Backlog vom zweiten Sprint (15.06.2024 - 30.06.2024).

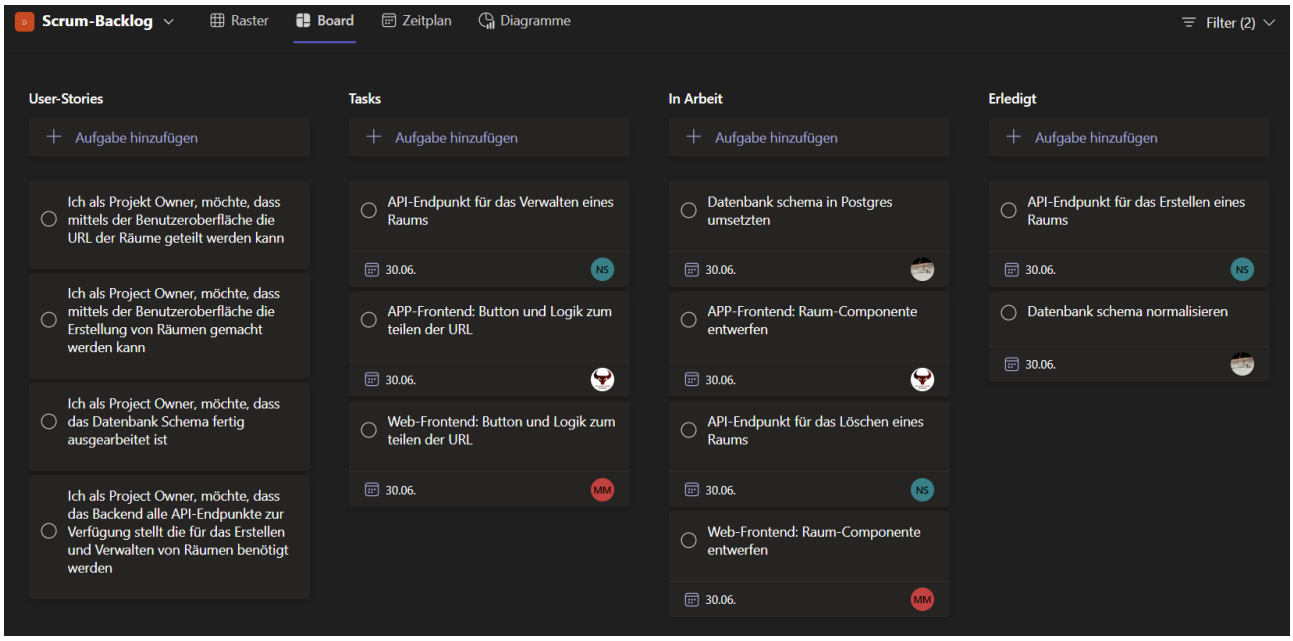


Abbildung 5: Microsoft Planner - Scrum Backlog

3.3.2 Vorgehensweise

- **Planung:** Wie bereits in Unterabschnitt 3.3.1 erwähnt, wurden alle User-Stories definiert, unterteilt und an die Mitglieder zugeteilt.
- **Durchführung:** Während eines Sprints wurden die Fortschritte regelmäßig überprüft und die Aufgaben in die Kategorien 'in Arbeit' und 'Erledigt' eingeteilt.
- **Abschluss:** Dabei wurden dem Auftraggeber die Fortschritte präsentiert und er konnte Rückmeldungen dazu geben.

4 Implementierung

4.1 Technischer Überblick

Unsere Anwendung besteht aus einer Ionic-Mobile-App und einer Angular-Webapplikation für die ScrumPoker-Schätzungen. Das Backend umfasst eine in Java implementierte REST-API, die die Geschäftslogik abbildet, sowie eine PostgreSQL-Datenbank, in der alle relevanten Daten gespeichert werden. Zusätzlich gibt es eine separate Python-API, die Analysen auf den gespeicherten Daten durchführt und weiterführende Auswertungen ermöglicht. Die Technologien wie Angular, Ionic, Java, Python und PostgreSQL arbeiten gut miteinander und können mühelos ineinander greifen. Gemeinsam bilden sie eine funktionierende Systemarchitektur. Abbildung 6 gibt einen Überblick über die eingesetzten Komponenten und deren Zusammenarbeit.

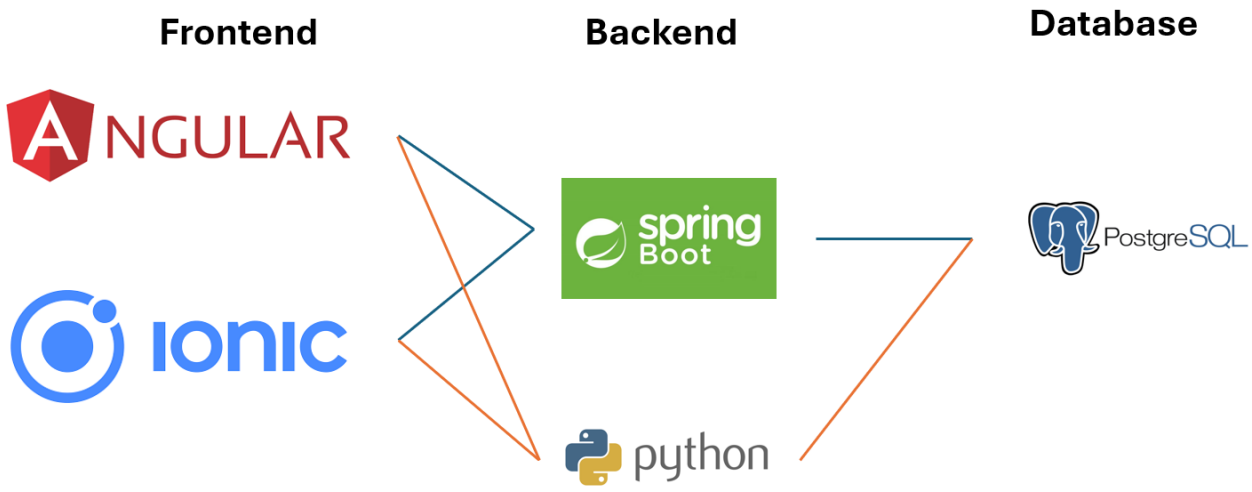


Abbildung 6: Technischer Überblick

4.2 Mobile-Anwendung

4.2.1 Einführung

In unserer Diplomarbeit haben wir uns dazu entschieden, eine zusätzliche mobile App zu entwickeln. Die Implementierung dieser App hat in erster Linie dazu beigetragen, unseren Projektumfang anzuheben. Somit hatte jeder seinen eigenen Themenbereich zur Verfügung. Darüber hinaus bietet die App auch viele Vorteile. Die Anwendung ermöglicht den Benutzern einen sehr nutzerfreundlichen und ortsunabhängigen Zugang zu einem Meeting. Dies ist vor allem für die Porsche Informatik wichtig, da ihre Teammitglieder oft aus verschiedenen Standorten teilnehmen.

Für die Entwicklung dieser App wurde das Framework Ionic verwendet. Dies bietet die Möglichkeit, eine einfache, plattformübergreifende, mobile Anwendung zu entwickeln.

Der genaue Aufbau der Architektur ist in der Abbildung 7 grafisch dargestellt. Die Funktionsweise des Frontends und die Kommunikation mit den anderen Komponenten werden im Abschnitt 4.2 ausführlich aufgelistet und erklärt.

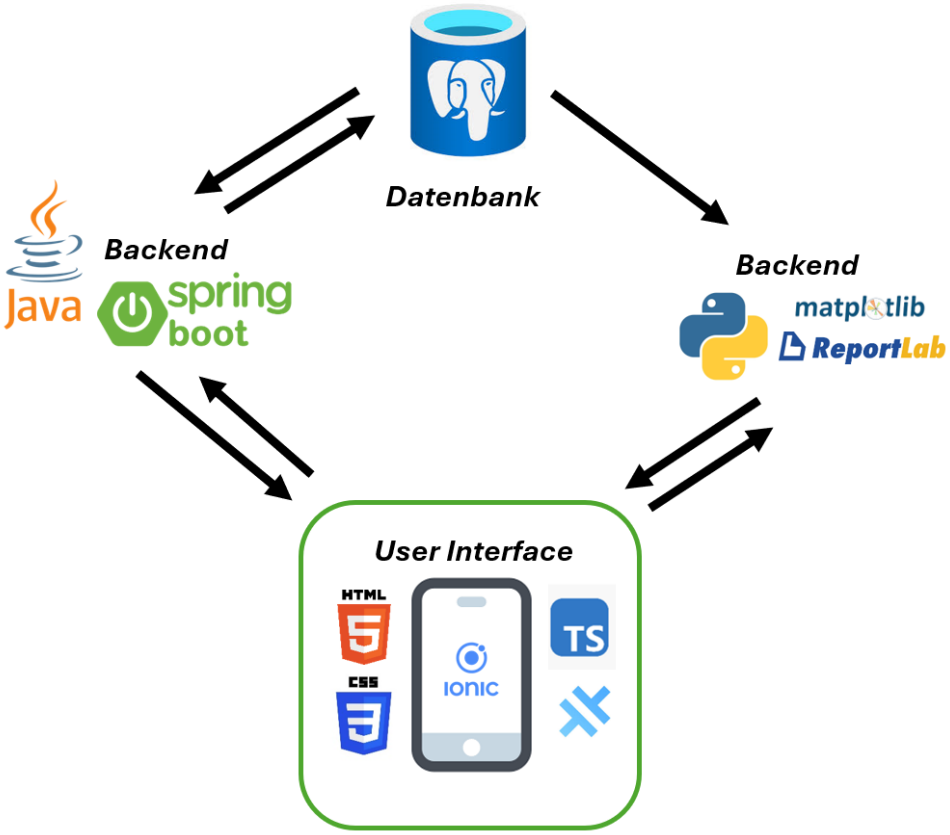


Abbildung 7: Architektur - App Anwendung

4.2.2 Struktur

Im Frontend wurde eine klare Trennung zwischen UI, Geschäftslogik und Datenverwaltung eingehalten. Dabei wurde das Projekt in Komponenten, Modelle und Services gegliedert. Somit bleibt die UI unabhängig von der Datenlogik, während die Services die Daten bereitstellen. In den Modellen werden alle Datenobjekte festgelegt und bilden somit die Grundlage.
Zusätzlich gibt es einerseits die Router-Mechanismen, welche die Navigation ermöglichen. Andererseits werden WebSockets implementiert, um eine Echtzeitübertragung zu gewährleisten.

Komponenten

Die Benutzeroberfläche unterteilt sich in mehrere, unterschiedliche Bereiche, welche den Komponenten entsprechen. Die Gliederung der Komponenten ist in der Tabelle 4 angeführt.

Komponente	**Beschreibung**
HomePageComponent	Startseite, auf welcher der Nutzer einen Raum erstellen oder beitreten kann
RoomPageComponent	Hauptansicht, in der das ScrumPoker-Spiel stattfindet
PlayGroundComponent	Bereich, in dem Spieler ihre Karten auswählen
MenuComponent	Steuerung der Raumeinstellungen und Verwaltung der Nutzer
ConfirmDialogComponent	Anzeigen eines Dialogs als Bestätigung für eine Handlung

Tabelle 4: Komponentenbeschreibung

Jede Komponente folgt dabei dem Standardaufbau mit einer HTML-Ansicht, einer CSS-Datei für das Styling und einer TypeScript-Datei für die Logik und Funktionalität. Besonderheiten werden in weiterer Folge genauer erklärt.
Die Kommunikation zwischen den Komponenten findet mittels '@Input' und '@Output'-Parametern oder durch Services statt. Bei Komponenten, die in einer Eltern-Kind-Beziehung stehen, wie die 'MenuComponent' und 'RoomPageComponent', werden die Daten mittels der '@Input' und '@Output'-Parametern übermittelt. Sollte keine direkte Verbindung bestehen, werden die Services genutzt, um einen Datenaustausch zu ermöglichen.

Routing und Navigation

Um zwischen den Ansichten navigieren zu können, wurde für jede Seite ein eindeutiger Pfad festgelegt. Dies erfolgt im RouterModule, welches in Listing 1 angeführt ist.

Listing 1: Routing

```
const routes: Routes = [
  {
    path: 'home',
    loadChildren: () => import('./home/home.module').then(m => m.HomePageModule)
  },
  {
    path: 'create-room',
    loadChildren: () => import('./home/play-ground/play-ground.module').then(m =>
        m.PlayGroundModule)
  },
  {
    path: '',
    redirectTo: 'home',
    pathMatch: 'full'
  },
];
```

Durch diese Routes ist es möglich, einfach mittels 'routes.navigate('/Path') zu den jeweiligen Seiten zu navigieren.

Services für die Datenverarbeitung

Die Services ermöglichen die Trennung der Komponenten von der Datenverarbeitung. Dabei kommunizieren sie mit dem Backend und übernehmen die Aufbereitung der Daten. Die Logik teilt sich auf die in der Tabelle 5 befindlichen Dienste auf.

Service	**Aufgabe**
RoomService	Erstellt und verwaltet Räume, verwaltet Teilnehmer
StatisticService	Berechnet und speichert die Abstimmungsergebnisse

Tabelle 5: Service und Aufgaben

Die Services sind somit die Vermittler zwischen Frontend und Backend. Hierbei senden sie HTTP-Anfragen an die REST-API und nehmen deren Antworten entgegen. Diese Daten werden in weiterer Folge verarbeitet und an die Komponenten weitergeleitet. Daten, welche vermehrt benötigt werden und sich nicht ändern, werden in den Services zwischengespeichert, um unnötige API-Aufrufe zu verringern.

WebSockets

In der Anwendung werden WebSockets verwendet, um Teilnehmer über Änderungen in Echtzeit informieren zu können. Dabei dient der eine dazu, Teilnehmer über User-Änderungen zu informieren. Der andere ermöglicht es, Benachrichtigungen über neue Fragestellungen zu senden. Die Aufgaben der WebSockets sind in der Tabelle 6 erneut aufgelistet.

WebSocket	**Aufgabe**
WebSocketUserService	Sorgt für Echtzeitkommunikation zwischen Teilnehmern
WebSocketQuestionService	Überträgt Fragen und Abstimmungsergebnisse in Echtzeit

Tabelle 6: WebSocket und Aufgaben

UI und Styling mit Ionic

Die Benutzeroberfläche wird, wie bereits erwähnt, mit Ionic gestaltet. Dies bietet mehrere Vorteile, da Ionic eine Vielzahl von vorgefertigten UI-Komponenten zur Verfügung stellt. Dies ermöglicht eine effizientere Entwicklung. Darüber hinaus wird in der App ein responsives Design verwendet. Dies bedeutet, dass die Elemente der App automatisch an die Bildschirmgröße angepasst werden. Somit kann die App beliebig auf Tablets und Smartphones angewendet werden.

Beim Designen wurde auch auf ausreichend Kontrast geachtet, damit eine gute Lesbarkeit gegeben ist. Dennoch wurde die App im selben Farbschema, wie das bereits zu Beginn angefertigte Logo, in Abbildung 8 ersichtlich, gestaltet. Das verwendete Farbschema ist in Abbildung 9 nochmals genau dargestellt. Mithilfe dieser Vorlage konnte ein einheitliches Bild erzeugt werden.
Beim Anordnen der Elemente wurde auch auf eine klare Struktur geachtet, damit sich die Nutzer besser zurechtfinden.

Abbildung 8: Logo der App

Abbildung 9: Farbpalette

Benutzerinteraktion und Benutzerfreundlichkeit

Ein wesentlicher Aspekt für die Nutzerfreundlichkeit ist ein Feedback nach einer Interaktion. Deswegen folgt in der Anwendung nach jeder Bedienung eine visuelle Änderung oder eine Benachrichtigung. Die wichtigsten Benachrichtigungen sind in der Tabelle 7 angegeben.

Aktion	**Visuelles Feedback**
Karte gewählt	Karte wird hervorgehoben
Raum erfolgreich betreten	Bestätigungsmeldung erscheint
Falsche Raum-ID eingegeben	Fehlermeldung wird rot dargestellt
Scrum Master hat Runde gestartet	Animation/Übergang zur Abstimmung

Tabelle 7: Aktionen und Visuelles Feedback

Damit Nutzer über einen Fehler ausreichend informiert werden, gibt es verschiedene Validierungen und Fehlermeldungen. Um diese einheitlich darstellen zu können, wurde eine Toast-Benachrichtigung, wie im Listing 2 dargestellt, angefertigt. Diese Methode kann für alle Nachrichten einheitlich verwendet werden. Um Eigenschaften ändern zu können, bietet die Methode die Möglichkeit, mittels Übergabeparametern die Nachricht, Anzeigedauer und Farbe zu übergeben. Am Ende wird dies als Pop-up-Benachrichtigung am Bildschirm angezeigt.

Listing 2: Toast-Methode

```
1    async presentToast(message: string, duration: number = 3000, color: string = 'danger'){
2      const toast = await this.toastController.create({
3        message:massge,
4        duration:duration,
5        position: 'top',
6        color:color
7      });
8      toast.present();
9    }
```

4.2.3 Login / Authentifizierung

Bei unserer Anwendung wurde sich bewusst gegen die Verwendung eines klassischen Login-Systems entschieden. Dies basiert darauf, dass die Porsche Informatik den Beitritt in einen Raum so schnell und einfach wie möglich wollte. Im Fokus stand dabei das Prinzip 'Keep it simple'.
Ein Login-System hätte zusätzlich Komplexität miteingebracht und den Prozess des Beitretens verlängert. Deswegen war es dem Auftraggeber wichtiger, keine Authentifizierung einzubinden. Anstatt des Logins wird nun die Möglichkeit geboten, innerhalb eines Raumes seinen Namen zu ändern. Dadurch können sich die Teammitglieder identifizieren und somit wird zusätzlich Zeit eingespart.
Die technische Umsetzung hierbei erfolgt dadurch, dass der Name mit der Raum-ID als JSON-Objekt mittels des Backends abgespeichert wird. Jeder Teilnehmer hat dabei die im Listing 3 aufgelisteten Attribute:

Listing 3: Interface für Player

```
export interface Player {
  id: number,
  roomId: string,
  name: string,
  isScrumMaster: boolean,
  selectedCard: string,
  color: string
}
```

Änderungen des Namens werden über die Benutzeroberfläche an das Backend übermittelt. Dort werden die Daten persistent in der Datenbank abgespeichert.

4.2.4 Raum erstellen / beitreten

In unserer Applikation gibt es die Möglichkeit, einen Raum zu erstellen oder einem Raum beizutreten. In beiden Fällen übernehmen WebSockets die Echtzeit-Übermittlung von Änderungen.

Raum erstellen

Wird mittels des Buttons 'Raum erstellen' ein Raum angelegt, erhält der Nutzer automatisch die Rolle des Scrum Masters. Dieser besitzt dabei besondere Rechte, darunter die Möglichkeit, Fragen zu stellen und eine Runde zu starten oder zu beenden.
Wird der 'Create Room' Button geklickt, wird die in Listing 4 angeführte Methode aufgerufen und ein Request gesendet. Dadurch wird im Backend eine eindeutige ID generiert. Diese wird als Response zurückgegeben und dient als Identifikation für den Raum.

Listing 4: Create a room

```
createRoom() {
  this.isLoading = true;

  console.log('Creating room');
  this.roomService.createRoom().subscribe(response => {
    console.log('RoomID: ', response.room_id);
    this.roomID = response.room_id;
    this.roomLink = this.roomID;

    setTimeout(() => {
      this.connectWebSocket();
      this.connectWebSocketQuestion();

      this.isLoading = false;

    }, 2000);
  }, error => {
    console.error('Error: try again later!', error);
    console.log('error');
    this.router.navigate(['/'], {state: {createError: true}});
    this.isLoading = false;
  });
}
```

Nachdem ein Raum erfolgreich erstellt wurde, wird mittels des Routers automatisch zur Raum-Seite gewechselt. Das Navigieren ist möglich, da im Routes-Array der Pfad 'create-room' definiert ist, wie im Listing 1 zu sehen. Im darauffolgenden Listing 5 wird gezeigt, dass beim Wechseln der Ansicht zusätzlich der Parameter 'isScrumMaster' = true übergeben wird. Hiermit kann sichergestellt werden, dass es sich bei dem User um einen Scrum Master handelt. Dies wird in weiterer Folge beim User hinterlegt.

Listing 5: Navigate to room

```
goToCreateRoom() {
  this.router.navigate(['/create-room'], { state: { isScrumMaster: true } });
}
```

Raum beitreten

Die Anwendung verfügt, wie in derAbbildung 10 zu sehen ist, über die Möglichkeit, einem Raum beizutreten. Dafür muss der Nutzer die Raum-ID, welche er vom Scrum Master erhalten hat, eingeben. Wenn die Eingabe erfolgt, wird eine Validierung durchgeführt. Ist diese erfolgreich, wird eine Anfrage an die API gestellt, ob der Raum existiert. Sobald dies bestätigt wurde, wird der Nutzer mit dem Raum verbunden und alle restlichen Raummitglieder über WebSockets darüber informiert.

Abbildung 10: ScrumPoker-App - Home Screen

Echtzeit-Updates mit WebSockets

Wie bereits erwähnt, wird ein WebSocket genutzt, um alle Teilnehmer über Änderungen zu informieren. Dies umfasst Änderungen der Mitglieder, der Fragen und der ausgewählten Karten. Damit dies möglich ist, wird ein WebSocket-Service benötigt. Darin werden, wie im Listing 6 zu sehen, die Methoden des WebSockets definiert.

Ein WebSocket ermöglicht es, eine ständige Verbindung zwischen einer Applikation und einem Server herzustellen. Dadurch können Nachrichten an den Server gesendet werden und man erhält eine Antwort, ohne diese vom Server abgefragt haben zu müssen. Dadurch erhält unsere Anwendung die Möglichkeit, über Änderungen informiert zu werden, ohne explizit danach zu fragen. [24]

Jeder Nutzer öffnet beim Betreten eines Raumes die WebSocket-Verbindung. Diese besteht aus Raum-ID und Benutzer-ID. Die Verbindung wird vom WebSocket verwaltet, und somit werden die Nachrichten an die richtigen Nutzer verteilt.

Listing 6: WebSocket Verbindung

```
connect(roomId: string, userId: number): void {
  this.socket = new
      WebSocket(`ws://localhost:8080/ws/users?roomId=${roomId}&userId=${userId}`);
  this.socket.onmessage = (event) => this.messagesSubject.next(JSON.parse(event.data));
}

getMessages(): Observable<any> {
  return this.messagesSubject.asObservable();
}
```

Damit die Komponenten die WebSocket-Updates empfangen können, müssen sie sich mit diesen verbinden und den Service abonnieren. Im Listing 7 wird die Verbindung zum WebSocket aus der RoomPage-Komponente aufgebaut.

Listing 7: WebSocket Verbindung herstellen

```
connectWebSocket() {
  if (this.roomID) {
    this.createPlayer(this.ownPlayer, this.roomID);
    this.webSocketService.connect(this.roomID, this.ownPlayer.id);
    console.log('WebSocket connected for room:', this.roomID);
    this.webSocketSubscription = this.webSocketService.getMessages().subscribe(
      data => {
        console.log('Received data:', data);
        this.players = data;
      },
      error => {
        console.error('Error receiving WebSocket message:', error);
      }
    );
  }
}
```

Durch das Abonnieren dieses Streams erhalten die Komponenten Benachrichtigungen und werden somit über neue Spieler oder aktualisierte Namen benachrichtigt. Wie im Code ersichtlich ist, verwendet das WebSocket-Service ein Subject. Dieses dient dazu, die Nachrichten zu verarbeiten. Damit andere Komponenten das Subject abonnieren können, ohne direkten Schreibzugriff darauf zu haben, wird die Methode asObservable() verwendet. Diese Methode stellt sicher, dass die Komponenten ausschließlich auf die Daten reagieren, ohne sie manipulieren zu können. [25]

4.2.5 Ablauf einer Runde

Um die Aufwandsschätzung in unserer Applikation zu ermöglichen, gibt es den Prozess 'Runde starten'.

Starten der Runde

Eine Runde beginnt, sobald der Scrum Master eine Frage formuliert. Diese kann er entweder schriftlich in das bereitgestellte Eingabefeld schreiben oder während eines Meetings mündlich

stellen. Idealerweise sollte die Frage jedoch verschriftlicht werden, damit sie für spätere Auswertungen protokolliert werden kann. Sobald der Scrum Master nun auf den 'Vote'-Button klickt, wird allen Teilnehmern im Raum die Frage angezeigt. Um auf die Benutzerinteraktionen reagieren zu können, wird ein click()-Event, welches im Listing 8 angeführt ist, verwendet. Durch dieses Event wird die Funktion 'beginVoting()' aufgerufen, die anschließend die Übermittlung der Frage übernimmt.

Listing 8: Click-Event()

```
<button class='btn btn-primary send-button'
    (click)='beginVoting()'
    *ngIf='!isVoting'>Vote
</button>
```

Das Ausführen der Funktion 'beginVoting()' führt außerdem dazu, dass alle Teilnehmer die Möglichkeit erhalten, eine Karte auszuwählen. Im Listing 9 ist ersichtlich, wie die 'SelectCard'-Methode aufgebaut ist. Dabei wird mittels des 'roomServices' das Backend über die neuen Zuweisungen der Karten informiert.

Listing 9: SelectCard Methode

```
  selectCard(card: number) {
    if (this.votingAllowed) {
      console.log(card + ' selected');
      this.ownPlayer.selectedCard = card.toString();
      this.roomService.setSelectCard(this.ownPlayer).subscribe(response => {
        this.ownPlayer.selectedCard = response.selectedCard;
        console.log('selectedCard updated: ' + this.ownPlayer.selectedCard);
      });
    }
  }
```

Fibonacci-Reihe

Die Karten basieren auf der Fibonacci-Reihe. Diese Folge wird verwendet, da sie viele Vorteile mit sich bringt. Die Fibonacci-Reihe hat, wie bereits erwähnt, zu Beginn sehr kleine Sprünge. Bei höheren Zahlen werden die Lücken jedoch immer größer. Dies eignet sich gut zum Abschätzen, denn auch kleinere Pakete können viel präziser geschätzt werden. Die großen Zahlen wie 40 und 100 sind lediglich dafür da, um mitzuteilen, dass der Aufwand zu groß ist und dass das Paket unterteilt werden sollte. Die Zahlen sind dabei keine exakten Einheiten wie Stunden oder Tage, sondern dienen als Maßstab für einen Schwierigkeitsgrad. Bezeichnet werden sie als Story Points.[26] In der Abbildung 11 ist die Fibonacci-Reihe nochmals grafisch visualisiert.

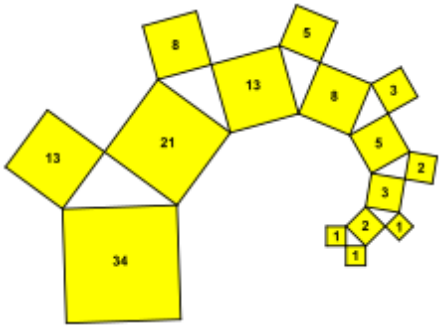


Abbildung 11: Fibonacci-Reihe
Quelle: https://de.wikipedia.org/wiki/Fibonacci-Folge

Beenden der Runde

Wenn der Voting-Prozess beendet werden soll, kann der Scrum Master mittels Klick auf 'End Vote' den Vorgang abschließen. Dadurch wird die Methode aus dem Listing 10 aufgerufen. Mittels dieser Funktion werden die Karten von allen Teilnehmern aufgedeckt und ausgewertet. Zusätzlich wird eine Anfrage an den Server gesendet, die den Durchschnittswert abruft. Dieser wird anhand der Karten der Teilnehmer berechnet. Sobald der Response wieder im Frontend eintrifft, wird der Durchschnitt übersichtlich für alle Teilnehmer dargestellt.
Auch wenn der Request, um den Durchschnitt abzufragen, nur vom Scrum Master abgesendet wird, erhalten dennoch alle Teilnehmer den Response mit dem Ergebnis. Der genaue Ablauf der Auswertung wird im Kapitel 4.2.6 erläutert.

Listing 10: EndVoting Methode

```
1   sendEndVoting() {
2     this.roomService.endVote(this.roomID).subscribe(response => {
3       this.averageNumber = Number(response);
4       console.log('Round ended');
5     });
6   }
```

Ebenso kann der Scrum Master entscheiden, ob eine neue Frage, oder ob dieselbe Frage erneut gestellt werden soll.

WebSocket

Für den Rundenablauf wird ebenso ein WebSocket verwendet. Dieser ermöglicht es, dass alle Teilnehmer in Echtzeit sehen, wann eine neue Frage gestellt oder das Voting beendet wird.
Zum Verbinden dieses WebSockets wird nur die Raum-ID benötigt, da jedes Mitglied eines Raumes die Benachrichtigungen erhalten soll. Die Attribute, welche zum Verbindungsaufbau benötigt werden, werden im WebSocket-Question Service definiert. Dies ist im Listing 11 genau angeführt.

Listing 11: WebSocket-Question

```
1   connect(roomId: string): void {
2     this.socket = new WebSocket(`ws://localhost:8080/ws/questions?roomId=${roomId}`);
3     this.socket.onmessage = (event) => this.messagesSubject.next(JSON.parse(event.data));
4   }
```

Damit neue Fragen direkt in der UI angezeigt werden, muss die Room-Komponente den WebSocket abonnieren.
Durch die Connect-Methode im Listing 12 wird die Verbindung zum Raum hergestellt und der Service kann abonniert werden. Dadurch wird bei Änderungen immer das 'data'-Objekt übermittelt. Mittels dessen kann festgestellt werden, welche neue Frage gespielt wird oder ob

die Runde beendet wurde und die Ergebnisse angezeigt werden sollen. Zudem wird das Attribut 'active' übergeben, welches angibt, ob zum Zeitpunkt des Beitritts bereits eine Runde gespielt wird. Ist dies der Fall, werden die Frage und die ausgewählten Karten angezeigt.

Listing 12: WebSocket-Question

```
connectWebSocketQuestion(){
  console.log('WEBSOCKET QUESTION')
  this.webSocketQuestionService.connect(this.roomID);
  this.webSocketSubscription =
      this.webSocketQuestionService.getMessages().subscribe(data => {
    if(data.active){
      this.showCards = true;
      this.showSelectedNumbers = false;
      this.votingAllowed = true;
      this.questionText = data.text;
    }
    else{
      this.showCards = false;
      this.showSelectedNumbers = true;
      this.votingAllowed = false;
      this.averageNumber = data.average_vote;
    }
  });
}
```

4.2.6 Auswertung

Die Auswertung des Votings ist ein wichtiger Prozess der Anwendung. Sobald alle Teilnehmer eine Karte ausgewählt haben oder der Scrum Master die Runde beendet hat, wird die Auswertung durchgeführt. Dabei wird der Durchschnitt aller Karten berechnet und angezeigt. Das Ergebnis dient in weiterer Folge als Grundlage für weitere Diskussionen und Planungen.

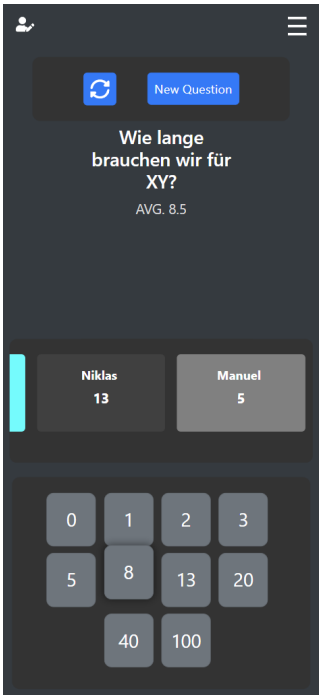


Abbildung 12: App - Auswertung

Ablauf der Auswertung

Beendet der Scrum Master die Runde durch Drücken des 'End Vote'-Buttons, wird die Kartenauswahl aller Teilnehmer deaktiviert und ein Request an das Backend gesendet. Dabei wird der Durchschnitt aller Karten berechnet und zurückgegeben. Die Ergebnisse werden danach dynamisch bei jeden User, wie in der Abbildung 12 angezeigt. Mithilfe des Operators '*ngFor' werden die Daten für jeden Spieler in der Liste abgefragt und dargestellt. Die konkrete Implementierung dieser Darstellung ist im Listing 13 ersichtlich.

Listing 13: Ergebnis anzeigen

```
<div *ngFor ='let user of players'>
  <p>{{ user.name }}</p>
  <div class='card-display'>{{ user.selectedCard}}</div>
</div>
<h2 *ngIf='showSelectedNumbers'>Durchschnitt: {{ averageNumber }}</h2>
```

4.2.7 Menüleiste

Die Menüleiste ermöglicht die Steuerung und Verwaltung von Teilnehmern. Der Scrum Master hat somit die Möglichkeit, wichtige Aktionen auszuführen, ohne das Spiel unterbrechen zu müssen. Diese Funktionalitäten umfassen:

Spieler verwalten

Der Scrum Master kann Teilnehmer promoten. Dabei wird ein anderer Teilnehmer zum Scrum Master ernannt. Dies ist vor allem dann wichtig, wenn der Scrum Master den Raum verlassen möchte und einer bestimmten Person die Rechte übergeben will.
Wird auf den 'Promoten'-Button geklickt, öffnet sich zunächst ein Bestätigungsdialog, in dem der Scrum Master gefragt wird, ob er die Rechte endgültig übertragen möchte. Die Implementierung dieses Dialogs ist im Listing 14 dargestellt.

Listing 14: Promote User

```
1    promoteUser(user: Player) {
2      const dialogRef = this.dialog.open(ConfirmDialogComponent, {
3        width: '400px',
4        data: { user },
5        disableClose: true,
6      });
```

Erst nach der Bestätigung wird die Promotion-Funktion ausgeführt, und der ausgewählte Benutzer wird zum neuen Scrum Master ernannt.
Wird ein Spieler während einer Runde inaktiv, kann dieser ebenso durch die Kick-Funktion entfernt werden. Dabei wird seine Verbindung getrennt und die Benachrichtigung 'Sie wurden vom Scrum Master gekickt!' wird angezeigt.

Raum Verlassen

In der Menüleiste hat jeder Teilnehmer die Option, mittels Klick auf den 'Leave'-Button den Raum zu verlassen. Beim Verlassen müssen die WebSocket-Verbindungen wieder geschlossen werden. Dies ist im Listing 15 abgebildet. Dabei wird zuerst ein Request an das Backend gesendet, der diese Informationen übermittelt.
Ist die WebSocket-Verbindung geschlossen, können keine weiteren Nachrichten mehr empfangen werden. Die anderen Teilnehmer werden über diese Änderung informiert.

Listing 15: Close WebSockets

```
closeWebSocketQuestion(){
  this.webSocketQuestionService.close();
}

closeWebSocket(){
  this.webSocketService.close();
}
```

Falls der Scrum Master den Raum verlässt, ohne zuvor einen Spieler zu promoten, wird der zuerst beigetretene Spieler automatisch zum neuen Scrum Master ernannt.

Zugangscode teilen

Jeder Raum verfügt über eine eindeutige ID, die auch für den Beitritt verwendet wird. Der Zugangscode wird in der Menüleiste angezeigt und kann von allen Teilnehmern kopiert werden. Beim Verwenden des Kopieren-Buttons wird dieser automatisch in die Zwischenablage übertragen. Die Logik für das Kopieren dieses Codes ist im Listing 16 dargestellt.

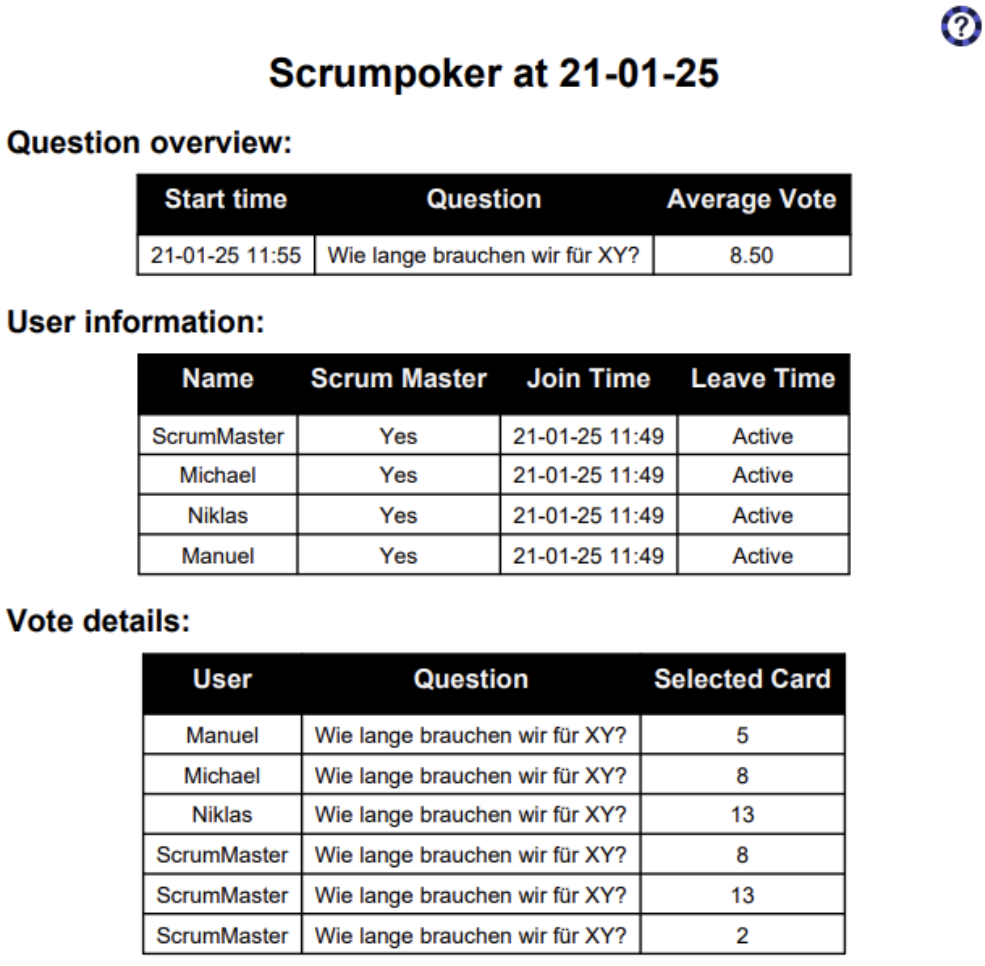
Listing 16: Room-Link kopieren

```
copyRoomLink() {
  navigator.clipboard.writeText(this.roomLink).then(() => {
    alert('Room link copied to clipboard!');
  }, () => {
    alert('Failed to copy room link.');
  });
}
```

Der Zugangscode kann den Teilnehmern übermittelt werden. Sie können diesen dann in das Eingabefeld der HomePage einfügen, um dem Raum direkt beizutreten.

PDF-Auswertung

Während eines Spiels werden die Daten und Ergebnisse laufend protokolliert und für weitere Auswertungen bereitgestellt. Zum Abrufen der Daten steht die Funktion 'Export als PDF' zur Verfügung. Dabei werden die Daten in einer PDF-Datei zusammengefasst und heruntergeladen. Der Inhalt umfasst die Teilnehmer, die gestellten Fragen, die ausgewählten Karten sowie den Durchschnitt jeder Runde. Ein Beispiel für die generierte PDF-Datei ist in Abbildung 13 zu sehen. Diese Datei ist nützlich für die Dokumentation von Sprint-Meetings und kann die weitere Planung erleichtern.
Erstellt wird dieses PDF vom Python-Backend und wird über einen API-Endpunkt zur Verfügung gestellt. Somit wird im Frontend mit dem Klick auf den Button das PDF abgefragt und der Download ausgeführt.



Scrumpoker at 21-01-25

Question overview:

Start time	Question	Average Vote
21-01-25 11:55	Wie lange brauchen wir für XY?	8.50

User information:

Name	Scrum Master	Join Time	Leave Time
ScrumMaster	Yes	21-01-25 11:49	Active
Michael	Yes	21-01-25 11:49	Active
Niklas	Yes	21-01-25 11:49	Active
Manuel	Yes	21-01-25 11:49	Active

Vote details:

User	Question	Selected Card
Manuel	Wie lange brauchen wir für XY?	5
Michael	Wie lange brauchen wir für XY?	8
Niklas	Wie lange brauchen wir für XY?	13
ScrumMaster	Wie lange brauchen wir für XY?	8
ScrumMaster	Wie lange brauchen wir für XY?	13
ScrumMaster	Wie lange brauchen wir für XY?	2

Abbildung 13: App - PDF Download

4.2.8 Capacitor

Bei der Entwicklung der App wurde zusätzlich Capacitor verwendet. Dabei handelt es sich um ein plattformübergreifendes Tool, welches es ermöglicht, native Funktionen eines mobilen Geräts zu nutzen. Darüber hinaus bietet es, ähnlich wie Ionic, eine hybride Lösung und ist somit auf allen Smartphones lauffähig. Es unterstützt also iOS, Android und Web. Dennoch ist es mit Capacitor möglich, zwischen den verschiedenen Plattformen zu unterscheiden und somit individuelle Anpassungen durchzuführen. [27]
In der ScrumPoker-App wird Capacitor, insbesondere für den Zugriff auf das Dateisystem und zum Speichern der PDF-Dateien, verwendet.

PDF Speicherung

Die Capacitor Filesystem API ermöglicht das Konvertieren und Speichern von PDF-Dateien. Bevor die Datei im Dokumentenverzeichnis des Geräts gespeichert werden kann, muss sie in Base64 umgewandelt werden. Base64 besteht aus 64 Zeichen und lässt sich mit jedem lateinischen Zeichensatz darstellen. Es kodiert binäre Dateien, wie die PDF-Datei, in eine Zeichenkette, die nur aus ASCII-Zeichen besteht. Da der Zugriff auf das Dateisystem eingeschränkt ist, wird die Base64-Kodierung benötigt, um die Datei in Textform abzuspeichern. Zudem ist Base64 plattformübergreifend kompatibel. Um die Datei zu konvertieren, werden nun wie im Listing 17 ersichtlich die Funktionen der Capacitor Filesystem API verwendet. [28]

Listing 17: Konvertieren zu Base64

```
1   private convertBlobToBase64 = (blob: Blob) => new Promise((resolve, reject) : void => {
2     const reader :FileReader = new FileReader(
3     reader.onerror = reject;
4     reader.onload = () => {
5       resolve(reader.result);
6     };
7     reader.readAsDataURL(blob);
8   });
```

War die Umwandlung erfolgreich, kann die Datei im Dateisystem abgelegt werden. Dabei kann man mittels Capacitor einfach auf das Filesystem-Directory zugreifen. Die Capacitor Filesystem API bietet dabei Zugriff auf verschiedene Speicherorte. Da gewisse Speicherorte vom Betriebssystem abhängig sind, wurde ein Verzeichnis verwendet, auf das man betriebssystemunabhängig zugreifen kann. Auf Grundlage dessen wurde das Standard-Dokumentenverzeichnis 'Directory.Documents' verwendet. Wie im Listing 18 ersichtlich, wurde mittels der Capacitor Filesystem API der Zielort auf '.Documents' gesetzt.

Listing 18: PDF abspeichern

```
async savePdfFile(blob: Blob, fileName: string) {
  const base64Data = await this.convertBlobToBase64(blob) as string;
  const saveFile = await this._Filesystem.writeFile({
    path: fileName,
    data: base64Data,
    directory: this.FilesystenDirectory.Docunents,
  });

  console.log('Datei gespeichert unter:', savedFile.uri);
  return saveFile.uri;
}
```

Zugriff auf das Dateisystem

Damit Capacitor Dateien speichern kann, müssen im AndroidManifest Berechtigungen gesetzt werden. Dabei wird die Berechtigung 'WRITE_EXTERNAL_STORAGE' für das Speichern von Daten und 'READ_EXTERNAL_STORAGE' für das Lesen dieser benötigt. Zusätzlich wird die 'INTERNET'-Berechtigung verwendet, um die API-Aufrufe zu laden. Die genaue Konfiguration der Berechtigungen im AndroidManifest ist im Listing 19 zu sehen.

Listing 19: Permissions

```
<!-- Permissions -->
  <uses-pernission android:name='android. permission.INTERNET' />
  <uses-pernission android:name='android.permission.WRITE_EXTERNAL_STORAGE' />
  <uses-permission android:name='android.permission.READ_EXTERNAL_STORAGE'/>
```

4.3 Webapplikation

4.3.1 Einleitung

Für den grafischen Teil der Diplomarbeit wurde ein Webfrontend mit Angular entwickelt. Die Website zeichnet sich durch hohe Benutzerfreundlichkeit und einfache Bedienung aus. Die Webapplikation verfügt über eine Startseite und das eigentliche Spielfeld. Auf der Startseite kann man einen Raum erstellen oder beitreten. Das Spielfeld dient dann dazu, dass man die Fragen vom Scrum Master bewertet und im Anschluss darüber diskutiert. Der Scrum Master hat neben der Berechtigung, Fragen zu stellen, auch noch die Kontrolle über die anderen Mitglieder. Er kann sie kicken oder er kann einen neuen Scrum Master bestimmen. Am Ende eines erfolgreichen ScrumPoker-Spiels kann man sich alle Fragen inklusive Antworten herunterladen. Die Architektur wird in der Abbildung 14 veranschaulicht.

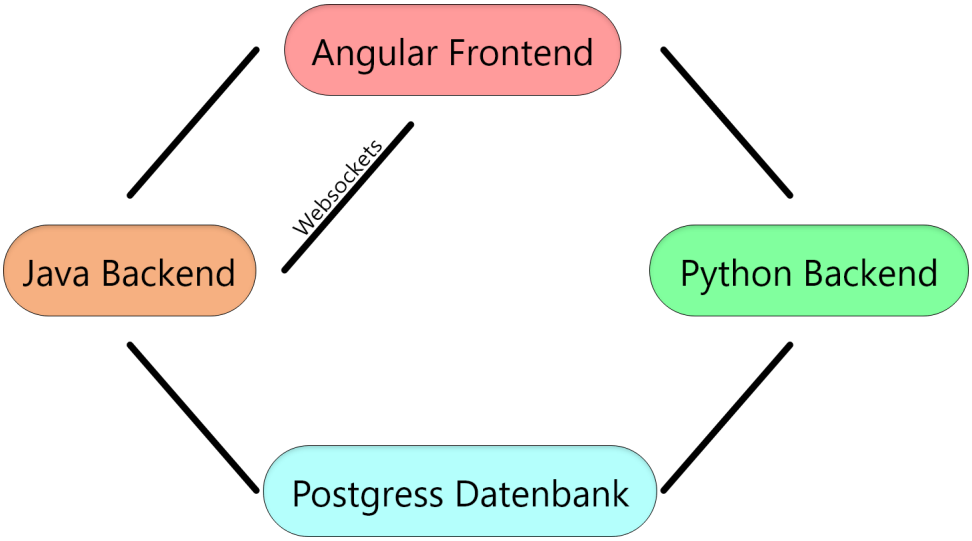


Abbildung 14: Architektur - Web Anwendung

4.3.2 Aufbau

Die wesentlichen Teile des Web-Frontends sind:

- Startseite
- Spieloberfläche
- Seitenleiste

Die einzelnen Seiten entsprechen genau den Komponenten des Angular-Projekts. Die Komponenten lassen sich dann mittels Routing aufrufen.

Für die Kommunikation mit der API des Backends wurde ein eigener Service erstellt. In den Komponenten wurde dann auf den Service zugegriffen. Für die WebSockets wurden ebenfalls Service-Klassen generiert.

4.3.3 Startseite

Um einen der Kernpunkte, nämlich 'schnell und einfach' erfolgreich zu erreichen, wurde eine sehr einfache und übersichtliche Startseite gewählt. Sie besteht aus einer Überschrift, Text und zwei Buttons. Um dem Eindruck von Leere und Farblosigkeit entgegenzuwirken, wurde zusätzlich eine Illustration eingebunden, die über einen Hover-Effekt verfügt. In der Abbildung 15 wird die Startseite abgebildet.

Abbildung 15: Startseite - Web Anwendung

Raum erstellen

Mit dem 'Create Game'-Button wird mithilfe der API ein Raum generiert. Danach wird man auf die Spielfeldseite als Scrum Master weitergeleitet. Man ist nun auch in der Lage, andere Spiele einzuladen. Man kann entweder einfach den Link über die Suchleiste kopieren und teilen oder wenn man auf die Seitenleiste geht, kann man mittels eines Buttons auch kopieren.

Raum beitreten

Wenn man einem bereits bestehenden Spiel/Raum beitreten möchte, kann man dies mittels des 'Join Game'-Buttons machen. Wenn man diesen drückt, erscheint ein Eingabefeld mit einer Validierung, dass man mindestens sechs Buchstaben oder Zahlen eingeben muss. Mittels Kommunikation an das Backend, wird überprüft, ob der Raum existiert. Falls ja, wird man auf die Spielfeldseite weitergeleitet und kann mitspielen. Falls dies fehlschlägt, wird der Benutzer durch eine Fehlermeldung darüber informiert, dass dieser Raum nicht existiert.

Spiel beitreten

Wenn der Benutzer den 'Join Game'-Button drückt, wird er zu einem Eingabefeld weitergeleitet. Die Eingabe wird anschließend durch das Backend überprüft, um sicherzustellen, dass sie korrekt ist und die Anforderungen erfüllt. Bei erfolgreicher Validierung kann der Benutzer dem Spiel beitreten. Im Listing 20 wird der Code dazu gezeigt.

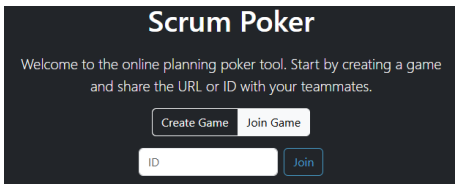


Abbildung 16: Spiel beitreten - Web Anwendung

Listing 20: Validierung und Beitritt zu einem Raum

```
1   async joinGame() {
2     // Validate
3     if (this.inputID.length !== 6) {
4       this.errorMsg = 'ID has to be 6 figures long.'
5       this.showAlert();
6       return;
7     }
8     // Connect to Room
9     let failed: boolean = false;
10    await this.api.getAllPlayers(this.inputID)
11      .toPromise()
12      .then((data) => console.log(data))
13      .catch((error) => failed = true);
14    if (failed) {
15      this.errorMsg = 'Room doesn't exists.';
16      this.showAlert();
17      return;
18    } else {
19      this.router.navigate(['/room/' + this.inputID]);
20    }
21  }
```

4.3.4 Spieloberfläche

Auf der Spieloberfläche ist die Übersichtlichkeit das wihtigste. Deshalb wurde alles sehr minimalistisch gehalten. In dem oberen Segment befindet sich die Benutzerliste, ein Button zum Öffnen der Seitenleiste und ein Eingabefeld zur Namensänderung, weil man standardmäßig als GuestXX in den Raum geladen wird. Das Herz der Anwendung ist der Pokertisch in der Mitte. Hier kann man den Status des Spiels erkennen. In der Mitte des Tisches sieht man dann die Frage des Scrum Masters. Weiter unten kann man seinen eigenen Spieler sehen. Das Kartendeck, das zum Spielen verwendet wird, befindet sich unten in der Mitte. In der Abbildung 17 ist die Spieloberfläche abgebildet.

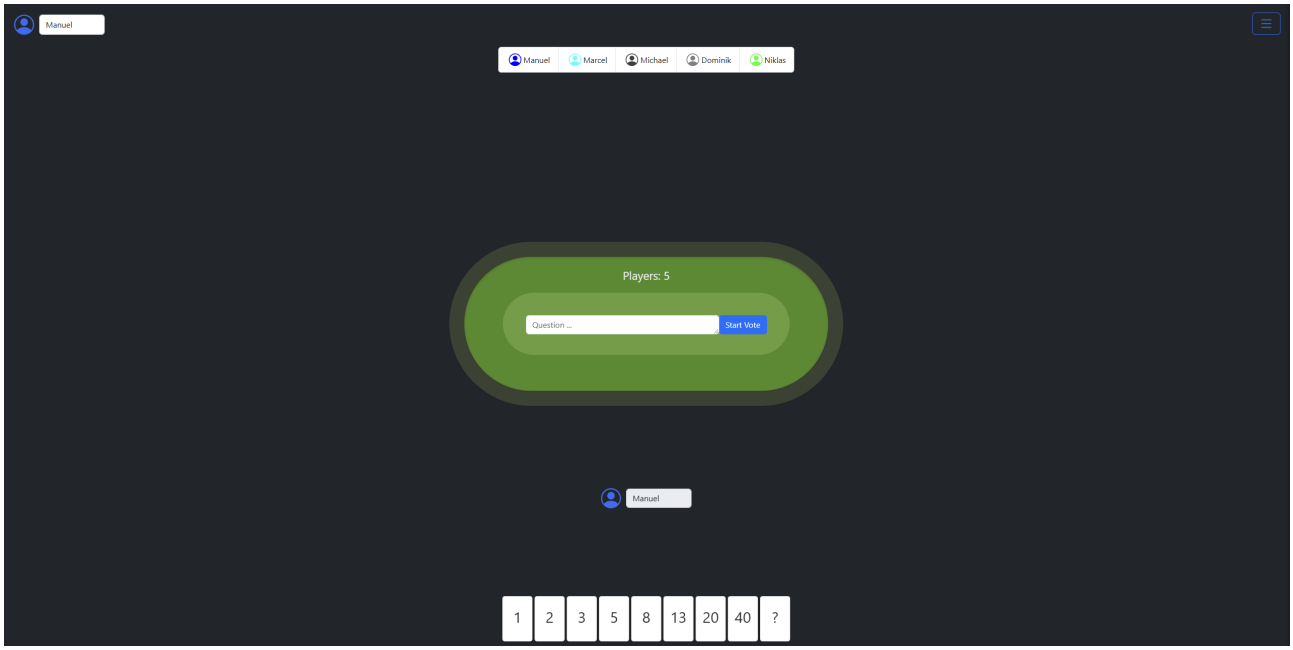


Abbildung 17: Spielfeld

Benutzerliste

In der Benutzerliste sieht man alle aktiven Spieler im Raum. Diese werden mittels Socketverbindungen auf dem Laufenden gehalten. Die Benutzerliste verfügt über drei verschiedene Ansichten:

- Benutzerliste - wird angezeigt, wenn gerade nicht gespielt wird. Abbildung 18
- Benutzerliste inklusive Stimmenabgabe (zugedeckt). Abbildung 19
- Benutzerliste inklusive Stimmenabgabe (aufgedeckt). Abbildung 20

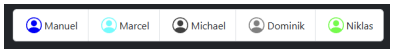


Abbildung 18: Benutzerliste

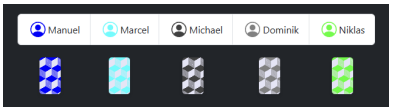


Abbildung 19: Benutzerliste - zugedeckt

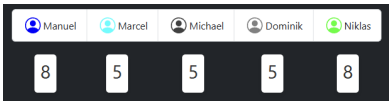


Abbildung 20: Benutzerliste - aufgedeckt

ScrumPoker Tisch

Beim ScrumPoker-Tisch kann man drei Komponenten erkennen. Das **Info Label**, zeigt die Spieleranzahl, wenn gerade auf den Scrum Master mit seiner Fragestellung gewartet wird. Wenn der Scrum Master das Abstimmen zur Frage startet, sieht man hier die Anzahl an Spielern, die noch nicht abgestimmt haben, und wenn man fertig mit dem Abstimmen ist, sieht man die durchschnittliche Schätzzahl. Das **Eingabefeld** ist für die Frage. Hier muss man unterscheiden zwischen Scrum Master oder nicht, da für einen Scrum Master das Eingabefeld aktiviert ist und für die anderen nicht. Neben dem Eingabefeld hat der Scrum Master noch Buttons, mit denen er die Runde steuern kann. Unter dem Eingabefeld erscheint auch eine Karte, wenn man abgestimmt hat. Dies dient nochmal zur Bestätigung, dass der Spieler besser erkennen kann, dass er schon abgestimmt hat. In den Abbildungen 21, 22 und 23 wird der Workflow abgebildet.

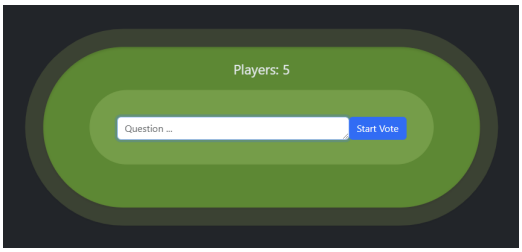


Abbildung 21: Poker Tisch – Spieler warten

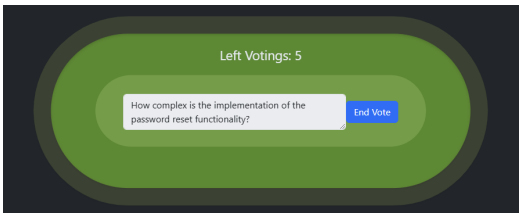


Abbildung 22: Poker Tisch – Spieler stimmen ab

Abbildung 23: Poker Tisch – Spieler fertig mit dem Abstimmen

Karten

Das Kartendeck besteht aus abgewandelten Fibonaccizahlen und einem Fragezeichen, wie man in der Abbildung 24 erkennen kann. Dieses dient dazu, falls jemand gerade etwas nicht versteht oder er bei dieser Frage aussetzen möchte. Das Kartendeck wurde als eigene Komponente erstellt. Dies bietet den Vorteil, dass unser Kunde die Karten schnell und einfach bearbeiten kann, falls sie andere Zahlen nehmen möchten. Man kann nur eine Karte auswählen, wenn der Scrum Master das Voting gestartet hat. Wenn man eine Karte dann auswählt, wird sie mit einer Animation nach oben geschoben. Die anderen Spieler erkennen dann an der Benutzerliste, ob man schon abgestimmt hat oder nicht. Der Nutzer ist in der Lage, solange seine ausgewählte Karte zu ändern, bis der Scrum Master die Abstimmung stoppt. Mittels eines WebSockets werden die Daten bei jedem Nutzer aktuell gehalten.

Abbildung 24: Karten

Design der Karten

Das Besondere an diesen Karten ist, dass es sich hierbei um keine Bilder handelt, sondern um SVGs (Scaleable Vector Graphics). Dies hat den großen Vorteil, dass sie leicht skalierbar, also in der Größe anpassbar, sind. Mithilfe eines '*ngfor' Elements ist es ebenso den Kunden gestattet, beliebig viele Karten hinzuzufügen. In den Listenings 21 und 22 ist der HTML-Code mit dem entsprechenden konfigurierbaren Array abgebildet. Im Listing 21 ist ebenfalls die Logik abgebildet, wie man Karten auswählen kann. Der optische Teil wird über '[class.selected]' verarbeitet. Hierbei wird kontrolliert, ob eine Karte selektiert ist, wenn ja, bekommt sie die entsprechende SCSS-Klasse, ansonsten nicht. Das Selektieren einer Karte wird über einen Mausklick gehandhabt. Wenn der Benutzer eine Karte anklickt, wird eine 'toggleSelected'-Methode aufgerufen. Diese leitet dann die Änderung der Daten an den anderen Komponenten weiter.

Listing 21: Cards-HTML

```
<div class="d-flex justify-content-center" style="position: fixed; bottom: 0; left: 0;
    right: 0;">
  <div *ngFor="let card of cards" class="card" [class.selected]="card.selected"
      (click)="toggleSelected(card)">
    <svg width="100%" height="100%">
      <rect width="100%" height="100%" fill="none" />
      <text x="50%" y="50%" text-anchor="middle" dominant-baseline="middle" fill="#333">{{
          card.value }}</text>
    </svg>
  </div>
</div>
```

Listing 22: Cards-Array

```
  cards = [
    { value: 1, selected: false },
    { value: 2, selected: false },
    { value: 3, selected: false },
    { value: 5, selected: false },
    { value: 8, selected: false },
    { value: 13, selected: false },
    { value: 20, selected: false },
    { value: 40, selected: false },
    { value: "?", selected: false },
  ];
```

Im Listing 23 sind drei SCSS-Klassen abgebildet. In der '.card' wird das grundlegende Design des SVGs festgelegt. Das Besondere ist, dass auch eine Transition verwendet wird. Dies ist ein Attribut, welches die Art der Animation und die Dauer der Animation festlegen kann. Wenn nun eine Karte selektiert wird, verfügt sie wegen der '[class.selected]' über eine zweite Klasse namens '.selected'. Diese Klasse hat ein Attribut 'transform', welches dann dafür sorgt, dass die Animation ausgeführt wird. In diesem Fall wird die Karte dann in 0.4 Sekunden mit einer 'ease'-Animation um 40 Pixel nach oben geschoben. Im Listing 23 ist noch eine '.card:hover'-Klasse. Mit dieser Klasse verfügt dann jede Karte über einen Hover-Effekt. Das heißt, sobald man mit dem Mauszeiger über eine Karte schwebt, wird diese wieder mit den Animations-Einstellungen der '.card'-Klasse, transformiert. Jedoch diesmal wird sie nicht verschoben, sondern um den Faktor 1.2 skaliert.

Listing 23: Cards-CSS

```css
.card {
  width: 60px;
  height: 90px;
  margin: 2px 2px 10px;
  display: inline-block;
  position: relative;
  border-radius: 5px;
  overflow: hidden;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  transition: transform 0.4s ease;
}

.card:hover {
  transform: scale(1.2);
  z-index: 1;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
  cursor: pointer;
}

.card.selected {
  transform: translateY(-40px);
  box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
  z-index: 2;
}
```

Namen ändern

In der linken oberen Ecke kann man dann seinen Namen ändern/eingeben. Dieser muss mindestens vier Buchstaben lang sein. Sobald dies erreicht ist, wird eine Nachricht an die API geschickt und alle anderen Spieler bekommen dann eine neue Benutzerliste, sodass sie den neuen Namen direkt sehen.

Listing 24: Name ändern

```typescript
  async changeName() {
    if (this.currentPlayer) {
      // Backend changeName
      if (this.playerName.length >= 4) {
        // console.log('test')
        this.currentPlayer.name = this.playerName;
        await this.api.changeName(this.currentPlayer)
          .toPromise()
          .then((data) => console.log(data))
          .catch((error) => console.log(error));
      }
    }
  }
```

Das Listing 24 zeigt, wie die Änderung des Namens implementiert wurde. Die Mindestlänge wird überprüft, und eine Nachricht wird an die API gesendet.

Listing 25: WebSocket

```typescript
  this.WebSocketSubscription = this.WebSocketService.getMessages().subscribe(data => {
    this.players = data;
    this.sortPlayers();
    this.updatedCurrentPlayer();
  });
```

In der Listing 25 sieht man die WebSocket-Kommunikation im Client.

Listing 26: WebSocket Service

```
1    connect(roomId: string, userId: number): void {
2      this.socket = new
           WebSocket(`ws://192.168.8.121:8080/ws/users?roomId=${roomId}&userId=${userId}`);
3      this.socket.onmessage = (event) => this.messagesSubject.next(JSON.parse(event.data));
4    }
5
6    getMessages(): Observable<any> {
7      return this.messagesSubject.asObservable();
8    }
```

Das WebSocket-Service-Codebeispiel Listing 26 zeigt, wie Nachrichten an alle verbundenen Benutzer gesendet werden.

4.3.5 Seitenleiste

Um die Seitenleiste zu aktivieren oder zu deaktivieren, verfügt die Seite über ein Hamburger-Icon(Menü-Icon) in der rechten oberen Ecke, das ebenso als Button fungiert. Hinterlegt ist hierbei eine 'toggleSidebar()' Methode, die den Wert einer Boolean-Variable verändert. Wenn der Wert dann 'True' gesetzt wird, von rechts eine Seitenleiste eingeschoben. Die Seitenleiste verfügt über vier Sektionen. In der Abbildung 25 ist die Seitenleiste abgebildet.

Abbildung 25: Seitenleiste

Spieler Verwaltung

In der ersten Sektion wird eine Liste der Spieler angezeigt. Der Scrum Master verfügt ebenso über die Funktion, Benutzer zu befördern oder zu kicken. Wenn er einen anderen Benutzer befördert, verliert der Scrum Master seine Rolle als Scrum Master und der andere Benutzer wird zum Scrum Master. In der Abbildung 26 wird dies abgebildet.

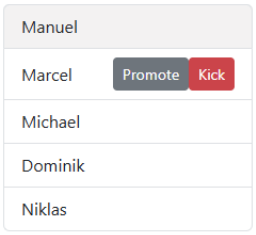


Abbildung 26: Spieler Verwaltung

Teilen des Links

Das zweite Element dient zum Teilen des Links. Hier kann man die URL sehen und mithilfe eines Buttons in die Zwischenablage kopieren.

Daten exportieren

Darunter befindet sich ein weiterer Knopf, mit dem man die Daten exportieren kann. In diesem Segment kommt unsere zweite API, die Python REST-API, ins Spiel. Mithilfe dieser API kann man einen Aufruf starten. Als Übergabeparameter benötigt dieser die RaumID. Zurück bekommt man eine PDF-Datei. Der Inhalt ist der Verlauf des Spiels. Es dient als Protokoll für das Spiel. Der Spieler, wie auch der Scrum Master, kann sich das Protokoll herunterladen.

Raum verlassen

Zuletzt gibt es noch einen Button zum Verlassen des Raumes. Hierbei wird der Benutzer zur Startseite zurückgeleitet.

4.3.6 Service

In Angular ist ein Service eine zentrale Anlaufstelle, um wiederverwendbare Logik aus verschiedenen Komponenten auszulagern und sauber zu strukturieren. Die Idee dahinter ist, dass bestimmte Funktionalitäten, wie zum Beispiel der Zugriff auf externe Datenquellen, die Verwaltung von Zuständen oder komplexe Berechnungen, nicht direkt in einer Komponente implementiert werden sollten, sondern in eigene abgegrenzte Klassen ausgelagert werden. Ein Service lässt sich dann in beliebig vielen Komponenten einbinden. Dies verbessert die Wartung einer Anwendung enorm.

Technisch gesehen ist ein Service in Angular nichts anderes als eine Klasse, die mit dem @Injectable()-Decorator versehen ist. Dieser gibt an, dass Angular die Klasse über sein Dependency-Injection System verwalten kann. Dependency Injection bedeutet, dass Abhängigkeiten – also zum Beispiel Services – automatisch dort bereitgestellt werden, wo sie benötigt werden, ohne dass sie manuell instanziiert werden müssen. Das sorgt für eine gute Kopplung zwischen Komponenten und den Services, die sie verwenden, und ist ein zentrales Konzept im Angular-Framework. In der Abbildung 27 wird die Kommunikation dargestellt.

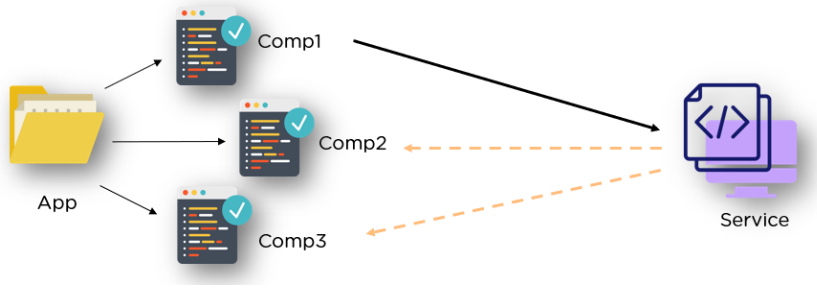


Abbildung 27: WebSocket
Quelle: https://www.simplilearn.com/tutorials/angular-tutorial/angular-service

4.3.7 WebSockets

WebSockets ermöglichen eine bidirektionale Verbindung zwischen dem Server und dem Client. Im Gegensatz zu HTTP-Anfragen ermöglichen Sockets eine dauerhafte Kommunikation und der Client sowie der Server können Daten austauschen. Dies ist von großem Vorteil bei Echtzeitanwendungen wie Computerspielen oder Aktien-Apps.

Ein WebSocket wird über einem sogenannten 'Handshake'-HTTP-Anfrage initialisiert. Wenn der Handshake erfolgreich ist, wird die Verbindung auf das WebSocket-Protokoll (ws:// oder wss:// für verschlüsselte Verbindungen) umgestellt. Ab diesem Moment bleibt die Verbindung offen, bis die Verbindung explizit geschlossen wird oder unterbrochen wird. [29] In der Abbildung 28 wird der Handshake veranschaulicht.

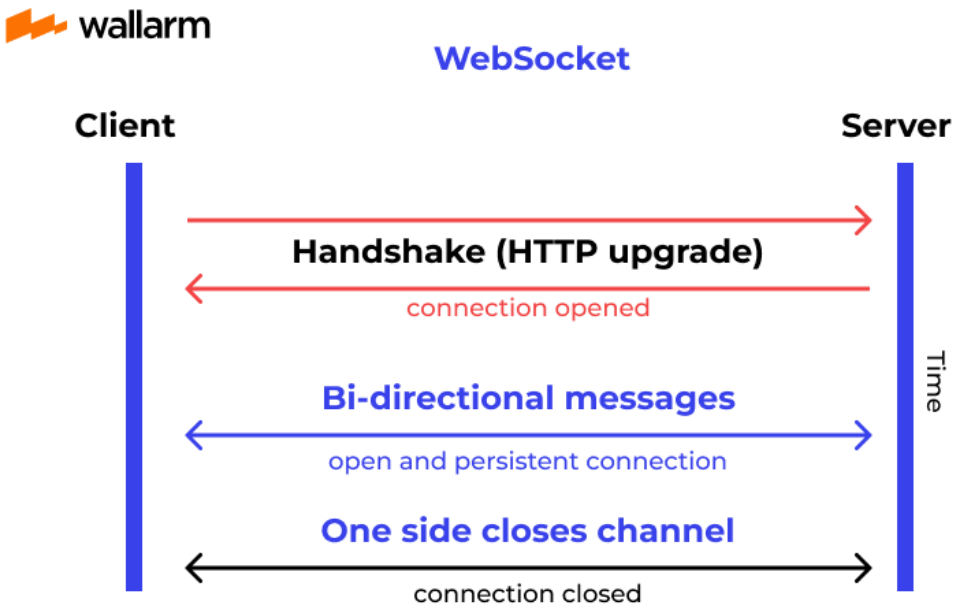


Abbildung 28: WebSocket Handshake
Quelle: https://www.wallarm.com/what/a-simple-explanation-of-what-a-WebSocket-is

Implementierung des WebSockets

Im Frontend befinden sich zwei verschiedene WebSockets. Ein Socket sorgt für den Datenaustausch von den Benutzern. Das heißt, es werden immer die aktuellen Benutzer angezeigt. Das Backend gibt mir Bescheid, wenn sich etwas mit der Benutzerliste ändert (Namen oder neuer Benutzer). Der andere Socket sorgt für die Aktualität der Frage. Beide Sockets sind ähnlich aufgebaut und verfügen über drei Methoden. Die 'connect'-Methode sorgt für den Aufbau des WebSockets und die 'getMessages' sorgt dann für die Echtzeiterhaltung der Daten. Vor der URL steht 'ws://'. Dies ist das Protokoll für die WebSockets. Im Listing 27

Listing 27: WebSocket

```
1   connect(roomId: string, userId: number): void {
2     this.socket = new
          WebSocket(`ws://192.168.8.121:8080/ws/users?roomId=${roomId}&userId=${userId}`);
3     this.socket.onmessage = (event) => this.messagesSubject.next(JSON.parse(event.data));
4   }
5   getMessages(): Observable<any> {
6     return this.messagesSubject.asObservable();
7   }
8
9   close(): void {
10    if (this.socket) {
11      this.socket.close();
12      this.socket = null;
13    }
14  }
```

4.3.8 Design

Das Wichtigste beim Design ist, dass es einfach und sauber aussieht. Um dies zu erreichen, muss man sinnvolle Fehlerbehandlung machen. Weiters gehören dazu verschiedene Animationen, damit der Benutzer immer weiß, was gerade am Bildschirm passiert. Weiters ist es unserem Auftraggeber wichtig, dass das Programm auch auf verschiedene Bildschirmgrößen problemlos bedienbar ist.

Lade Animation

Damit der Benutzer sich auskennt, verfügt der 'create Game' Button über eine Ladeanimation, das heißt, solange der Raum noch nicht fertig generiert ist, bekommt der Benutzer eine Ladeanimation, wo steht 'Create Room'. Somit ist sich der Benutzer im Klaren, dass der Raum generiert wird. In der Abbildung 29 ist diese Animation ersichtlich.

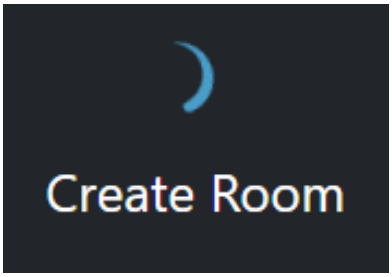


Abbildung 29: Loading Screen

Alerts

Für Fehlermeldungen oder andere wichtige Informationen für den Benutzer werden Alerts verwendet. Mithilfe dieser Alerts lässt sich schnell und einfach dem Benutzer mitteilen, ob etwas nicht korrekt ist. In der Abbildung 30 ist die Fehlermeldung abgebildet, wenn ein Benutzer eine RaumID eingibt, bei der kein Raum existiert. Der Alert erscheint dann für eine gewisse Zeit rechts oben. Ein Beispiel-Alert wird in der Abbildung 30 gezeigt.

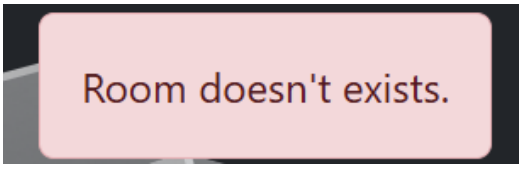


Abbildung 30: Alert

Responsive Design

Der Auftraggeber wollte ebenfalls, dass man es überall spielen kann. Somit muss die Applikation auch auf anderen Bildschirmgrößen spielbar sein. Um dies zu ermöglichen, wurde mittels SCSS-Klassen und einigen Transformationsattributen eine responsive Spieloberfläche geschaffen. Diese verfügt über alle Funktionen, die man sonst auch auf einem größeren Bildschirm hat. Die Abbildung 31 zeigt, dass die Applikation auch im Hochformat funktioniert.

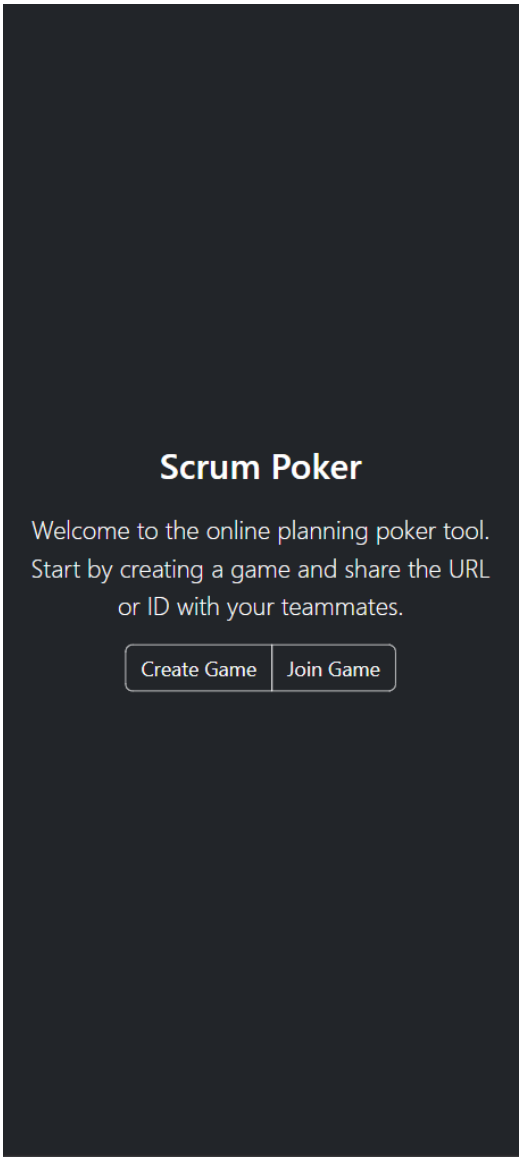


Abbildung 31: Responsive Design

4.4 REST-API

4.4.1 Einleitung

Die REST-API ist die zentrale Komponente unseres Projekts und stellt die Logik hinter dem Frontend zur Verfügung, speichert die Daten in der Datenbank und führt die Berechnung des Durchschnitts einer Runde durch oder generiert eine einzigartige ID für jeden Raum. Bei jeder Aktion, die gemacht wird, führt die REST-API den logischen Teil aus. Es wurde sich für eine RESTful-API entschieden, da diese schon aus dem Schulalltag und aus vorherigen Projekten bekannt war und da unser Auftraggeber auch hauptsächlich mit REST-APIs, geschrieben in Java, arbeitet.

4.4.2 Aufbau

Das Backend trennt die Verantwortlichkeiten, dies ist wichtig für die Wiederverwendbarkeit und Wartbarkeit des Codes. Daher wurde ein klassisches API-Design gewählt, welches einen Controller, eine Service-, eine Modell- und eine Datenbankschicht hat. Es gibt unterschiedliche Schichten: In der Controller-Schicht werden HTTP-Anfragen verarbeitet und Antworten gesendet. In der Service-Schicht befindet sich die Businesslogik unseres Projekts. Die Datenbank- oder Persistenzschicht ist verantwortlich für alle Datenbankinteraktionen. Die Model-Schicht definiert die unterschiedlichen benötigten Datenmodelle.

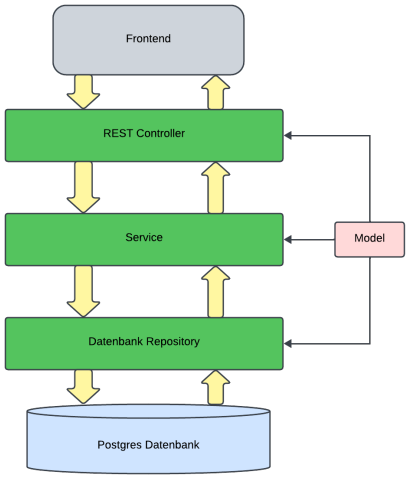


Abbildung 32: Aufbau des Backends

4.4.3 Controller

Die unterschiedlichen Controller verwalten HTTP-Anfragen und geben Antworten beziehungsweise Statuscodes zurück, je nachdem, ob die Anfrage erfolgreich war oder nicht. Es gibt drei unterschiedliche Controller, jeweils für Question, Room und den User. Darin befinden sich die verfügbaren Methoden für jede dieser Entitäten. Für diese Gliederung wurde sich aufgrund unserer Datenbankentitäten entschieden, da unser Backend nach dem 'Database-First'-Ansatz entwickelt wurde. Im User Controller werden die Methoden zum Erstellen, Löschen, Abfragen, Befördern zum Scrum Master oder das Auswählen einer Karte bereitgestellt und an den jeweiligen Service delegiert, da sich im Controller keinerlei Businesslogik befindet.

Listing 28: Endpunkt im 'UserController' zum Erstellen eines Users

```
1       @PostMapping("/create/user/{room_id}")
2       public ResponseEntity<?> createUser(@PathVariable String room_id, @RequestBody
            Map<String, Object> requestBody) {
3           if (requestBody == null) {
4               return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Request body is
                    null");
5           }
6
7           try {
8               boolean isScrumMaster =
                    Boolean.parseBoolean(requestBody.get("is_scrum_master").toString());
9               User createdUser = userService.createUser(room_id, isScrumMaster);
10              return ResponseEntity.status(HttpStatus.CREATED).body(createdUser);
11          } catch (Exception e) {
12              return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(e.getMessage());
13          }
14      }
```

Im Codebeispiel in Listing 28 erkennt man, dass der Controller einen API-Endpunkt unter der URL '/create/user/room-id' als POST-Request bereitstellt. Ein Controller hat die Aufgabe, einen API-Endpunkt bereitzustellen und mit einem entsprechenden HTTP-Statuscode zu antworten und zusätzlich, wenn benötigt, auch noch mit einem Response-Body. Hier ruft die Methode im Controller dann die Benutzer-erstellen-Methode im User-Service auf, welche dann dafür verantwortlich ist, dass der User mit den Parametern richtig erstellt wird und anschließend die Repository-Schicht aufruft, in der die Daten persistent gespeichert werden.

4.4.4 Services

Die Services sind die Schnittstellen zwischen Controller und Repository, hier wird außerdem die gesamte Businesslogik implementiert. Es gibt für jede Entität einen eigenen Service, dies fördert die Lesbarkeit des Codes. Da die Methoden für jede Entität eine eigene Klasse haben, sind sie leichter zu finden und zu ändern. Eine Aufgabe des Room-Services ist beispielsweise die Erstellung eines Raums und die Generierung einer einzigartigen Raum-ID, welche aus sechs Zeichen besteht. Diese Zeichen sind entweder Großbuchstaben oder Zahlen, das sind genug Zeichen, um eine einzigartige ID für jeden Raum zu generieren, da die Räume nach einer Woche gelöscht werden und es 2.176.782.336 unterschiedliche Möglichkeiten für die Raum-ID gibt. Jedoch ist es trotzdem denkbar, jemandem einfach die Raum-ID anzusagen, sodass dieser dem Raum beitreten und mitspielen kann, das heißt, man muss jemandem nicht unbedingt den Link zum Raum senden. Dies macht das Spielerlebnis für ein Team einfacher. Im Codebeispiel in Listing 29 sieht man, wie die ID generiert und die Methode im Service aufgerufen wird, diese ist dann für den Aufruf der Methode im Repository, mit dem fertigen Raum-Objekt, verantwortlich.

Listing 29: Raum erstellen im UserService

```
1       public String createRoom() {
2           List<Room> rooms = getAllRooms();
3           String randomID = generateID();
4           rooms.add(new Room(randomID));
5           try {
6               createRoom(randomID);
7           } catch (SQLException e) {
8               System.err.println(e.getMessage());
9           }
10          return randomID;
11      }
```

Die Raum-ID wird in der 'generateID'-Methode in Listing 30 generiert. Zum Erstellen der zufälligen ID wird die 'StringBuilder'-Klasse sowie die Klasse 'SecureRandom' verwendet. Der StringBuilder wird hierbei mit der Länge der ID initialisiert und danach wird eine Schleife durchlaufen, bei der so oft wie die gewünschte Länge der ID ein zufälliger Charakter ausgewählt wird. Der String 'CHARACTERS' beinhaltet alle Großbuchstaben des Alphabets sowie die Ziffern von null bis neun. Also wird zunächst eine zufällige Zahl von null bis zur Anzahl der gewünschten Charaktere ausgewählt. Als Nächstes wird dieser Charakter an den StringBuilder hinten angehängt und, sobald die gewünschte Länge erreicht wurde, wird die ID des Raums zurückgegeben und kann so vom Controller per HTTP-Response an das Frontend zurückgegeben werden, und die ID wird in der Datenbank gespeichert.

Listing 30: Generieren einer neunen Raum-ID

```
1       public static String generateID() {
2           StringBuilder id = new StringBuilder(ID_LENGTH);
3           for (int i = 0; i < ID_LENGTH; i++) {
4               int index = random.nextInt(CHARACTERS.length());
5               id.append(CHARACTERS.charAt(index));
6           }
7           return id.toString();
8       }
```

4.4.5 Models

Die Entitäten sind von den Entitäten der Datenbank übernommen, um das Speichern in der Datenbank einfacher zu gestalten. Also wurden die Entitäten Question, Room und User erstellt. Jede dieser Entitäten stellt einen Konstruktor bereit, in diesem werden alle Variablen als Parameter übergeben und anschließend wird geprüft, ob ein gültiger Wert übergeben wird. Ist dies der Fall, wird die Variable gesetzt. Dann wird für jede Variable eine Getter- und eine Setter-Methode bereitgestellt, da nicht direkt auf die Variable zugegriffen werden darf. So sind alle Entitäten einheitlich und einfach zu verstehen. Auf die Entitäten kann von jeder Schicht im Backend zugegriffen werden. Zusätzlich wird noch eine individuelle 'toString'-Methode bereitgestellt, um die Standard-toString-Methode zu überschreiben. Das erhöht die Lesbarkeit, was für den Logging-Vorgang vorteilhaft ist. Folgend in Listing 31 wird als Beispiel die Room-

Entität gezeigt. Diese hat nur die Raum-ID gespeichert, welche dann in den anderen Entitäten als Fremdschlüssel genutzt wird.

Listing 31: Room Model

```
1       public class Room {
2           public String room_id;
3
4           public Room(String room_id) {
5               this.room_id = room_id;
6           }
7
8           public Room(String room_id, int scrum_master_id) {
9               this.room_id = room_id;
10          }
11
12          public Room() {
13          }
14
15          public String getRoom_id() {
16              return room_id;
17          }
18
19          public void setRoom_id(String room_id) {
20              this.room_id = room_id;
21          }
22
23          @Override
24          public String toString() {
25              return "Room{" +
26                      "room_id=" + room_id +
27                      '}';
28          }
29      }
```

4.4.6 Datenbankverbindung

Für die Datenbankverbindung wird die 'application.properties'-Datei, welche vom Spring Boot Framework bereitgestellt wird, genutzt. In dieser Datei werden alle benötigten Informationen für die Verbindung zur Datenbank festgelegt, wie die URL, der Name der Datenbank, das Passwort und der Benutzername. Diese Datei erleichtert das Entwickeln der Datenbanklogik, indem man hier zentral alle Verbindungskonfigurationen vornehmen kann. Spring Boot nutzt diese Datei, um automatisch eine DataSource zu konfigurieren, die für den Datenbankzugriff verwendet wird. Die '@PostConstruct'-Annotation sorgt dafür, dass die 'init()' Methode direkt nach der Initialisierung der Bean aufgerufen wird, so wird die Verbindung zur Datenbank direkt hergestellt. [30]

Listing 32: Verbindung zur Datenbank

```
@PostConstruct
public void init() {
    this.connectionString = env.getProperty("spring.datasource.url");
    this.user = env.getProperty("spring.datasource.username");
    this.password = env.getProperty("spring.datasource.password");

    if (connectionString == null || user == null || password == null) {
        throw new IllegalArgumentException("Datenbank-Verbindungsparameter sind nicht
            korrekt konfiguriert.");
    }
    connect();
}

public void connect() {
    try {
        Class.forName("org.postgresql.Driver");
        connection = DriverManager.getConnection(connectionString, user, password);
        System.out.println("Connection established");
    } catch (ClassNotFoundException | SQLException e) {
        throw new RuntimeException("Fehler beim Verbinden zur Datenbank: ", e);
    }
}
```

Im Codebeispiel, in Listing 32, wird gezeigt, wie die Verbindungsparameter zur Datenbank aus 'application.properties' zum Verbindungsaufbau genutzt werden.[31].

Die Persistenz-Klasse implementiert alle benötigten Datenbankoperationen. So ruft der Service die folgende Methode in Listing 33 auf, um den Namen eines Nutzers zu ändern und die Änderung direkt in der Datenbank zu speichern. Der aktualisierte Nutzer wird dann unmittelbar zurückgegeben. Nach diesem Konzept funktionieren alle Methoden in der Datenbank. Dies gewährleistet eine einfachere Wartung und eine bessere Übersichtlichkeit, da alle Funktionen sauber getrennt sind.

Listing 33: Ändern des Benutzernamens in der Persistenzschicht

```
public User changeUserNameDB(String roomId, int userId, String username) throws
    SQLException, Exception {
    String queryUpdate = "UPDATE \"User\" SET name = ? WHERE user_id = ? AND room_id =
        ?";

    // Benutzername aktualisieren
    try (PreparedStatement updateStatement = connection.prepareStatement(queryUpdate))
        {
        updateStatement.setString(1, username);
        updateStatement.setInt(2, userId);
        updateStatement.setString(3, roomId);
        int rowsAffected = updateStatement.executeUpdate();

        if (rowsAffected == 0) {
            throw new Exception("User not found or no changes detected.");
        }
    }
    return getUpdatedUser(userId, roomId);
}
```

Es wird JDBC (Java Database Connectivity) genutzt, um die Datenbankverbindung aufzubauen, ebenso werden alle Queries per PreparedStatement durchgeführt. Bei einem PreparedStatement wird eine SQL-Query vorbereitet mit '?' in der Abfrage, welche danach im Code gesetzt werden können, wie in Listing 33 zu sehen ist. In diesem Beispiel wird die Query zum Ändern des Benutzernamens eines Users vorbereitet und danach mit der 'updateStatement'-Methode die

Parameter Benutzername, Benutzer-ID und Raum-ID gesetzt und die Query wird auf der Datenbank ausgeführt. Es wurde sich für PreparedStatements entschieden, da die Nutzung von PreparedStatements Schutz vor SQL-Injection bietet, da die SQL-Query und die Daten getrennt werden und erst zur Laufzeit eingesetzt und geprüft werden. Zusätzlich wird eine Abfrage nur einmal geparsed und kompiliert, und danach wird diese Abfrage immer wieder mit den neuen Daten ausgeführt, dies macht die Abfragen effizienter. Ebenso wird eine bessere Wartbarkeit und Lesbarkeit durch die Trennung von SQL und Code erreicht, und es ist einfacher, Änderungen im Code vorzunehmen. [32] Es wird zuerst eine SQL-Query geschrieben, in dieser werden Parameter festgelegt und dann per 'Update-Statement' gesetzt. Danach wird die Datenbankoperation auf der verbundenen Datenbank ausgeführt und in diesem Beispiel dann der Name des entsprechenden Nutzers geändert. Dazu wird die Benutzer- und Raum-ID benötigt, da der Primärschlüssel der Benutzerentität in der Datenbank aus diesen beiden gebildet wird. Dieses Vorgehen wird in allen Methoden der Persistenzschicht angewandt. Alle diese Methoden sind im Allgemeinen gleich aufgebaut. Dies erleichtert die Wartbarkeit des Codes und macht diesen übersichtlich.

4.4.7 WebSockets

Das Backend ist die zentrale Komponente unserer Anwendung, es wird also auch dafür verwendet, die unterschiedlichen Clients zu synchronisieren. Dies wird mittels zwei Klassen, namentlich 'QuestionWebSocketHandler' und 'UserWebSocketHandler' ermöglicht. Zusätzlich wird auch noch die Klasse 'WebSocketConfig' benötigt, um die beiden WebSockethandler nach den individuellen Anforderungen zu konfigurieren. Ein WebSocket bietet die Möglichkeit, eine Website dynamisch und in Echtzeit abzurufen, wobei der Kommunikationskanal geöffnet bleibt. Dadurch ist es für unser Backend möglich, allen Clients eine Benachrichtigung zu senden, wenn ein neuer Benutzer beigetreten ist oder wenn vom Scrum Master eine Frage gestellt wurde. Dies ist für unser Projekt essenziell, da dies die Grundlage für unsere Spielmechanik ist. [24]
Es wurde sich für WebSockets entschieden, da es dank WebSockets möglich ist, Nachrichten zwischen Front- und Backend auszutauschen, ohne immer Requests und Responses und den damit verbundenen Aufwand für den Verbindungsaufbau senden zu müssen. Dies ist in der Abbildung 33 grafisch dargestellt. Es wird einmal beim Verbinden des Benutzers eine Session geöffnet. Dann kommunizieren Frontend und Backend bidirektional. Dies bietet auch eine geringere Latenz, da durch die persistente Verbindung der Overhead von HTTP-Anfragen, welche immer wieder gesendet werden, eingespart wird. Eine Alternative wäre gewesen, die Updates über Polling zu erhalten. Dabei sendet der Client dauerhaft einen Request an den Server, ob sich etwas geändert hat. Dies ist sehr ineffizient, da man natürlich die meiste Zeit

dieselbe Antwort vom Server zurückbekommt. Hier sind WebSockets bei Weitem die bessere Alternative, da hier einmal eine Verbindung aufgebaut wird und dann der Server, wenn eine Änderung passiert, dem Client eine Benachrichtigung senden kann. Außerdem sind WebSockets der Standard in der Entwicklung von Applikationen, welche Echtzeitkommunikation benötigen, und auch die bevorzugte Option unseres Auftraggebers. [33]

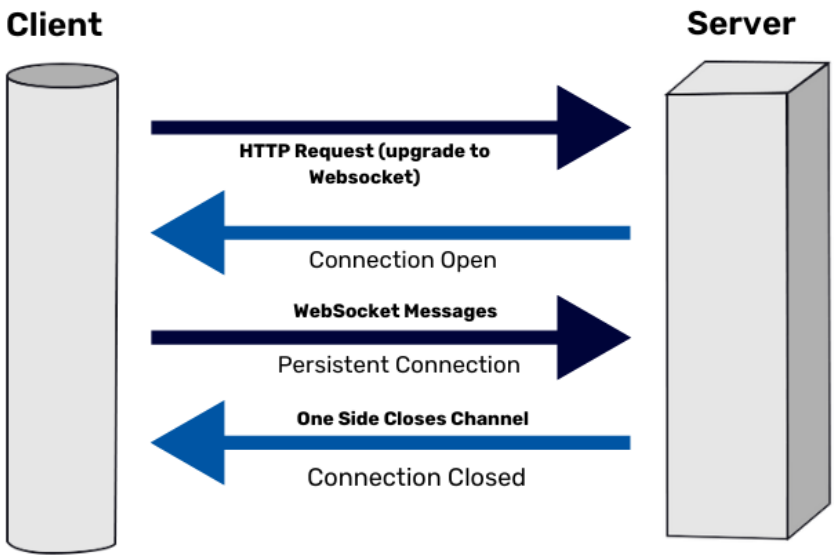


Abbildung 33: WebSocket Funktion
Quelle: https://www.mindbowser.com/springboot-websockets-real-time-guide/

WebSockets ermöglichen eine duplexe Kommunikation über eine einzelne TCP-Verbindung, dies ermöglicht einen Datenaustausch in Echtzeit, wo auch der Server den Client benachrichtigen kann, dass sich etwas geändert hat. Dafür muss der Client keine Anfrage senden, anders als bei standardmäßiger HTTP-Kommunikation. [34]

WebSocket und REST-API

Es wurde sich für eine Architektur mit sowohl einer Kommunikation zwischen Client und Server über eine REST-API als auch über WebSockets entschieden. Diese Architektur wurde gewählt, da beide Technologien Vorteile und Nachteile haben und sie beide in Kombination am effizientesten zum Einsatz kommen in unserer Anwendung. Durch die Nutzung beider Technologien müssen die Verantwortlichkeiten beider klar definiert sein, um die Kombination aus beiden effektiv nutzen zu können. Durch die Kombination dieser beiden Technologien ist unsere Anwendung auch skalierbarer und performanter. Es können also problemlos auch viele Benutzer gleichzeitig in unterschiedlichen Räumen ein Sprint-Planning-Meeting abhalten, ohne dass es zu Performanceeinbrüchen kommt. Die REST-API ist also verantwortlich für Anfragen

für persistente Daten, beispielsweise das Erstellen eines neuen Raums oder das Speichern einer Frage. Die WebSockets hingegen sind verantwortlich für Echtzeit-Benachrichtigungen des Servers an den Client. Ein Beispiel dafür wäre, das Senden einer Notifikation, wenn ein Spieler eine Karte auswählt.

UserWebSocket

Der UserWebSocket ist verantwortlich für die Synchronisierung der unterschiedlichen Benutzer in einem Raum. Wenn beispielsweise ein Spieler in den Raum beitritt, müssen die restlichen Spieler, die davor schon in diesem Raum waren, benachrichtigt werden, damit der neue Benutzer angezeigt werden kann. Zuerst müssen jedoch die Session-Attribute gesetzt werden, um die Funktionalität des WebSocketHandlers zu ermöglichen. Dies wird in der ‘afterConnectionEstablished‘-Methode gemacht. Diese Methode ist im Listing 35 abgebildet und wird ausgeführt, wenn die Verbindung zwischen Client und Server aufgebaut wurde, also wenn ein Benutzer einem Raum beigetreten ist. Anschließend werden die Identifikationsnummern des Raums und des Users in der Hash-Map ’sessions’ gespeichert. Diese ’sessions’ sind WebSocketSessions und werden in einer Map gespeichert. Über dieses Interface werden alle WebSocket-spezifischen Operationen ausgeführt. In der Methode ’getQueryParam’ werden die Query-Parameter der im Parameter mitgegebenen Session über die ’.getQuery()’-Methode ausgelesen. So kann man die Benutzer- und Raum-ID aufrufen und diese dann als Session-Attribute setzen. [35]

Listing 34: ’getQueryParam’-Methode im UserWebSocket

```
1  private String getQueryParam(WebSocketSession session, String paramName) {
2      String query = session.getUri().getQuery();
3      if (query != null) {
4          for (String param : query.split("&")) {
5              String[] keyValue = param.split("=");
6              if (keyValue.length == 2 && keyValue[0].equals(paramName)) {
7                  return keyValue[1];
8              }
9          }
10     }
11     return null;
12 }
```

In Listing 34 wird die oben beschriebene ’getQueryParam’-Methode aufgerufen und die ausgelesene Raum- und Benutzer-ID als Session-Attribute gesetzt.

Listing 35: Sessionkonfiguration

```
@Override
    public void afterConnectionEstablished(WebSocketSession session) throws IOException {
        String roomId = getQueryParam(session, "roomId");
        String userId = getQueryParam(session, "userId");

        if (roomId != null && userId != null) {
            sessions.put(session.getId(), session);
            session.getAttributes().put("roomId", roomId);
            session.getAttributes().put("userId", userId);
            System.out.println("Added Session: " + session.getId() + " for Room: " +
                roomId + " and User: " + userId + "\n" + "Sessions: " + sessions.size());
        } else {
            session.close(CloseStatus.BAD_DATA);
        }
    }
```

Um nun auch noch die Benutzer zu benachrichtigen, dass eine Änderung der Spieler aufgetreten ist, wird die Methode 'notifyUsers', welche in Listing 36 gezeigt wird, vom User Service aufgerufen, wenn ein Spieler mit der gleichen Raum-ID erstellt wird. Diese Methode sendet dann an alle Benutzer in diesem Raum eine Benachrichtigung. Nach dieser Benachrichtigung wird im Frontend die Liste der Benutzer automatisch aktualisiert. Diese Liste der Spieler wird per 'TextMessage' übergeben.

Listing 36: Benachrichtigen der User in einem Raum

```
public void notifyUsers(String roomId) throws Exception {
    System.out.println("Notify Users in Room: " + roomId);
    String payload = objectMapper.writeValueAsString(userService.getUsersInRoom(roomId));
    for (WebSocketSession session : sessions.values()) {
        String sessionRoomId = (String) session.getAttributes().get("roomId");
        if (sessionRoomId != null && sessionRoomId.equals(roomId) && session.isOpen()) {
            session.sendMessage(new TextMessage(payload));
        }
    }
}
```

Natürlich muss auch das Verlassen eines Users an alle anderen Clients gesendet werden. Zu diesem Zweck gibt es die 'afterConnectionClosed'-Methode in Listing 37, diese Methode wird aufgerufen, wenn die Verbindung, unabhängig von welcher Seite, abbricht. Es kann sein, dass die Verbindung noch geöffnet ist, jedoch ist es sehr unwahrscheinlich, dass eine gesendete Nachricht zu diesem Zeitpunkt noch beim Gegenüber ankommt. [36] Bei unserem UserWebSocketHandler wird beim Aufrufen der Methode 'afterConnectionClosed' die Session mit der jeweiligen ID gelöscht. Das bedeutet, wenn ein Benutzer nach dem Verlassen wieder dem Raum beitritt, hat er nicht mehr dieselbe Benutzer-ID. Danach wird der Benutzer vom User Service gelöscht und die Benutzer, die sich noch im Raum befinden, werden mit der 'notifyUsers'-Methode benachrichtigt, dass sich an den Benutzern im Raum etwas geändert hat. Aufgrund dieser Benachrichtigung kann die Liste der Spieler im Raum per HTTP-Request aktualisiert und anschließend angezeigt werden.

Listing 37: Sessionabbruch eines Benutzers

```
@Override
public void afterConnectionClosed(WebSocketSession session, CloseStatus status) throws
    IOException {
    sessions.remove(session.getId());
    String userId = (String) session.getAttributes().get("userId");
    String roomId = (String) session.getAttributes().get("roomId");
    if (userId != null) {
        try {
            userService.deleteUser(Integer.parseInt(userId), roomId);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    try {
        notifyUsers(roomId);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

QuestionWebSocket

Der zweite WebSocket in unserem Backend stellt die Funktionalität der Fragen bereit. Er arbeitet mit dem Question Service, welches die Methoden zum Stellen einer Frage und dem Antworten der Mitspieler bereitstellt. Sonst funktioniert dieser WebSocket sehr ähnlich zu dem User-WebSocket. So wird in Listing 38 beim Erstellen oder beim Beenden einer Frage eine Benachrichtigung an alle Benutzer gesendet, dass sich bei der Frage etwas geändert hat. Die Benutzer senden, soweit sie abgestimmt haben, ihre Antwort per HTTP-Request an den Server. Der zweite Parameter in der 'notifyUsers'-Methode ist ein Boolean, dieser sagt aus, ob der Status der Runde aktiv oder inaktiv ist. Wenn eine Frage gestellt wird, ist dieser Boolean 'true', wenn auf 'End Vote' gedrückt wird, ist dieser Boolean 'false'.

Listing 38: Benachrichtigen der Benutzer im QuestionWebSocket

```
public void notifyUsers(String roomId,boolean isActive) throws Exception {
    System.out.println("Notify Users in Room new Question: " + roomId);
    Question question = questionService.getQuestion(roomId);
    question.setActive(isActive);
    System.out.println(question.isActive());
    String payload = objectMapper.writeValueAsString(question);
    for (WebSocketSession session : sessions.values()) {
        String sessionRoomId = (String) session.getAttributes().get("roomId");
        if (sessionRoomId != null && sessionRoomId.equals(roomId) && session.isOpen()) {
            session.sendMessage(new TextMessage(payload));
        }
    }
}
```

4.4.8 Spiellogik

Die Spiellogik ist möglichst einfach und intuitiv gehalten, damit das Spiel einfach in ein Sprintplanungs-Meeting integriert werden kann und es zu keinen langen Wartezeiten kommt, in denen Teammitglieder der Runde beitreten. Wenn man die Seite aufruft, hat man die Wahl, ob

man einen Raum erstellen oder einem Raum beitreten möchte. Wenn man einen neuen Raum erstellt, wird vom Frontend ein Request an das Backend gesendet, dass ein neuer Raum benötigt wird. Daraufhin kommt per HTTP-Response die Raum-ID zurück, im Listing 39 wird gezeigt, wie der Endpunkt für die Erstellung eines Controllers realisiert wird. Zusätzlich wird für den Benutzer, der den Raum erstellt hat, ein neuer Benutzer angelegt, bei welchem das Scrum Master-Attribut auf "true"gesetzt ist. Wenn man einem Raum beitritt, wird vom Frontend ein HTTP-Request gesendet, um einen Benutzer zu erstellen, welcher kein Scrum Master ist.

Listing 39: Raum erstellen Endpunkt im RoomController

```
@GetMapping("/create/room")
public ResponseEntity<?> createRoom() {
    try {
        String roomId = roomService.createRoom();
        Map<String, String> response = new HashMap<>();
        response.put("room_id", roomId);
        return ResponseEntity.status(HttpStatus.CREATED).body(response);
    } catch (Exception e) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(e.getMessage());
    }
}
```

Wenn der Raum erstellt ist und die Raum-ID an die individuellen Teammitglieder weitergegeben wurde, können diese dem Raum beitreten. Damit dies bei allen Benutzern angezeigt wird, die bereits im Raum sind, ohne dass diese die Seite neu laden müssen, muss das Backend allen Benutzern im Raum eine Notifikation per WebSocket senden, damit diese alle Benutzer angezeigt bekommen. Wenn alle Teammitglieder im Raum sind, kann der Scrum Master beginnen, eine Frage zu stellen und die Abstimmung zu beginnen. Um die Abstimmung zu beginnen, gibt der Scrum Master die Frage im Fragefeld ein und drückt dann auf den Start-Voting-Button. Beim Klick auf den Button wird ein POST-Request an die API gesendet und die Raum-ID sowie der Text der Frage werden als Parameter übergeben. Dieser Request wird anschließend wieder vom Controller entgegengenommen. Der Controller ruft die Methode zum Hinzufügen einer Frage im Question-Service auf, diese Methode wird im Listing 40 gezeigt. In dieser Methode wird die Frage in der Datenbank gespeichert und benachrichtigt alle Benutzer in diesem Raum. Dafür wird die ‘notifyUsers‘-Methode im Question-WebSockethandler aufgerufen.

Listing 40: Frage hinzufügen im QuestionService

```
public Question addQuestion(String roomId, String text, boolean playAgain) throws
    Exception {
    try {
        Question question = database.addQuestionDB(roomId, text, playAgain);
        userWebSocketHandler.notifyUsers(roomId);
        questionWebSocketHandler.notifyUsers(roomId,true);
        return question;
    }catch (Exception e) {
        throw e;
    }
}
```

Wenn eine Frage gestellt und alle Benutzer benachrichtigt wurden, wählen die Benutzer eine Zahl aus und schätzen so den Aufwand, der für das Paket erwartet wird. Die User-Entität speichert die ausgewählte Karte als String, um auch zu ermöglichen, dass ein Benutzer sich der Abstimmung enthält und symbolisch ein Fragezeichen abstimmt. Um die momentan ausgewählte Karte zu speichern, stellt die REST-API den PUT-Endpunkt '/select/card' bereit. Der Aufruf dieses Endpunkts ruft das Userservice auf, welches wieder die Methode zur Speicherung der momentanen Kartenwahl in der Datenbank und die Benachrichtigung der Benutzer aufruft. Diese Methode wird im Listing 41 gezeigt. Diese Benachrichtigung der anderen Benutzer wird benötigt, da es allen Benutzern angezeigt wird, sobald man abgestimmt hat. Wenn man sich noch einmal anders entscheidet und eine andere Karte auswählen möchte, kann man einfach eine andere Karte auswählen. Ist dies der Fall, wird derselbe Request mit der neuen Karte noch einmal gemacht und durch den PUT-Request wird der Benutzer aktualisiert.

Listing 41: Karte auswählen

```
1  public User selectCard(int userId, String roomId, int cardId) throws Exception {
2      User selectCard = selectCardDB(roomId, userId, cardId);
3      userWebSocketHandler.notifyUsers(roomId);
4      return selectCard;
5  }
```

Wenn alle Spieler abgestimmt haben, drückt der Scrum Master den 'End Vote' Button. Dann wird ein POST-Request an den Endpunkt '/end/Vote' gesendet mit der Raum-ID im Request-Body. Diese Anfrage ruft die 'endVote' Methode im Questionservice auf, diese wird im Listing 42 dargestellt. Diese Methode speichert die beendete Runde in der Datenbank und ruft die Benachrichtigung der Benutzer im Question-WebSockethandler auf. Zusätzlich wird im Service noch die Berechnung des Durchschnitts der gewählten Karten durchgeführt und dann als HTTP-Response zurückgegeben.

Listing 42: Voting beenden

```
1  public double endVote(String roomId) throws Exception {
2      try {
3          double averageVote = calculateAverage(roomId);
4          questionWebSocketHandler.notifyUsers(roomId,false);
5          endVoteDB(roomId);
6          return averageVote;
7      } catch (Exception e) {
8          throw e;
9      }
10 }
```

Danach kann man entweder die Runde mit derselben Frage noch einmal spielen oder mit einer anderen Frage weitermachen. Um die Frage noch einmal zu stellen, drückt man auf den 'Again'-Button und es wird ein POST-Request an '/again' gesendet und die Runde wird mit derselben Frage noch einmal gestartet. In diesem Fall wird die 'addQuestion'-Methode mit dem Boolean 'playAgain' true aufgerufen. Wenn man sich jedoch einigen konnte, kann man über das

nächste Paket abstimmen. Dafür betätigt man den entsprechenden Button und es wird wieder ein Request an den POST-Endpunkt '/question' gesendet.

4.4.9 Benutzerverwaltung

Die Benutzerverwaltung wird über die Klassen 'UserController' und 'UserService' abgewickelt. Wie schon erwähnt, wird ein Benutzer beim Beitreten eines Raumes über den Endpunkt 'create/user/room-id' erstellt. Wenn ein Benutzer beitritt, wird die 'notifyUsers'-Methode im UserWebSocketHandler aufgerufen, um alle Benutzer im jeweiligen Raum zu benachrichtigen und das Frontend zu aktualisieren, um den neuen Benutzer anzuzeigen. Der Scrum Master hat dann die Möglichkeit, einen Benutzer zum Scrum Master zu befördern oder diesen aus dem Raum zu entfernen. Für diese beiden Möglichkeiten kann man im Raum die Seitenleiste auf der rechten Seite aufklappen. Dort werden alle Benutzer im Raum aufgelistet. Der Scrum Master des Raums kann einen Benutzer auswählen und kann anschließend den Benutzer befördern. Wenn ein Benutzer befördert wird, wird der PUT-Endpunkt der REST-API 'user/promote' aufgerufen. Dieser Endpunkt wird in der 'promoteUser'-Methode bereitgestellt, diese wird im Listing 43 gezeigt. Diese Methode behandelt jegliche Fehler im Request und ruft dann die 'promoteUser'-Methode im UserService auf. Diese ruft nur die 'setScrumMaster'-Methode auf und benachrichtigt die Benutzer im Raum. In der 'setScrumMaster'-Methode wird dem aktuellen Scrum Master des Raums seine Rolle entzogen, damit am Ende nicht zwei Scrum Master existieren können. Dann wird der Benutzer, dessen ID übergeben wurde, in der Datenbank aktualisiert. Dies wird durch ein PreparedStatement durchgeführt. Wenn der Scrum Master erfolgreich geändert wurde, wird der aktualisierte Benutzer noch an das Frontend zurückgegeben.

Listing 43: Endpunkt zum befördern eines Benutzers

```
1  @PutMapping("user/promote")
2  public ResponseEntity<?> promoteUser(@RequestBody Map<String, Object> requestBody) {
3      if (requestBody == null) {
4          return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Request body is null");
5      }
6      try {
7          int user_id = Integer.parseInt(requestBody.get("user_id").toString());
8          String room_id = requestBody.get("room_id").toString();
9          User promoted = userService.promoteUser(user_id, room_id);
10         return ResponseEntity.status(HttpStatus.CREATED).body(promoted);
11     } catch (Exception e) {
12         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(e.getMessage());
13     }
14 }
```

In der Methode 'setScrumMaster',im Listing 44, wird ein Benutzer mit der übergebenen ID im jeweiligen Raum zum Scrum Master befördert. Um den Spielfluss reibungslos weiterführen zu können, ist es wichtig, dass keine Probleme in dieser Methode auftreten. Daher werden mögliche Fehler dauerhaft abgefragt. Es ist auch wichtig, dass es durch einen Fehler nicht nach dem

Befördern zwei Scrum Masters gibt. Aus diesem Grund wird zuerst dem ursprünglichen Scrum Master durch die 'deleteScrumMaster'-Methode seine Rolle als Scrum Master entzogen. Diese hat als Parameter nur die Raum-ID, da es in jedem Raum nur einen Scrum Master gibt und dann dieser einfach degradiert wird. Wenn der ursprüngliche Scrum Master seine Rolle als Scrum Master nicht mehr hat, wird die SQL-Query zum Setzen des 'isscrummaster'-Attributs auf 'true' mittels PreparedStatement erstellt. Danach werden die Parameter Raum- und Benutzer-ID gesetzt und das PreparedStatement durchgeführt. Wenn das alles erfolgreich war, wird der aktualisierte Benutzer, der neue Scrum Master, retourniert. Folgend wird das Service den aktualisierten Benutzer an den Controller übergeben, dieser sendet den HTTP-Response.

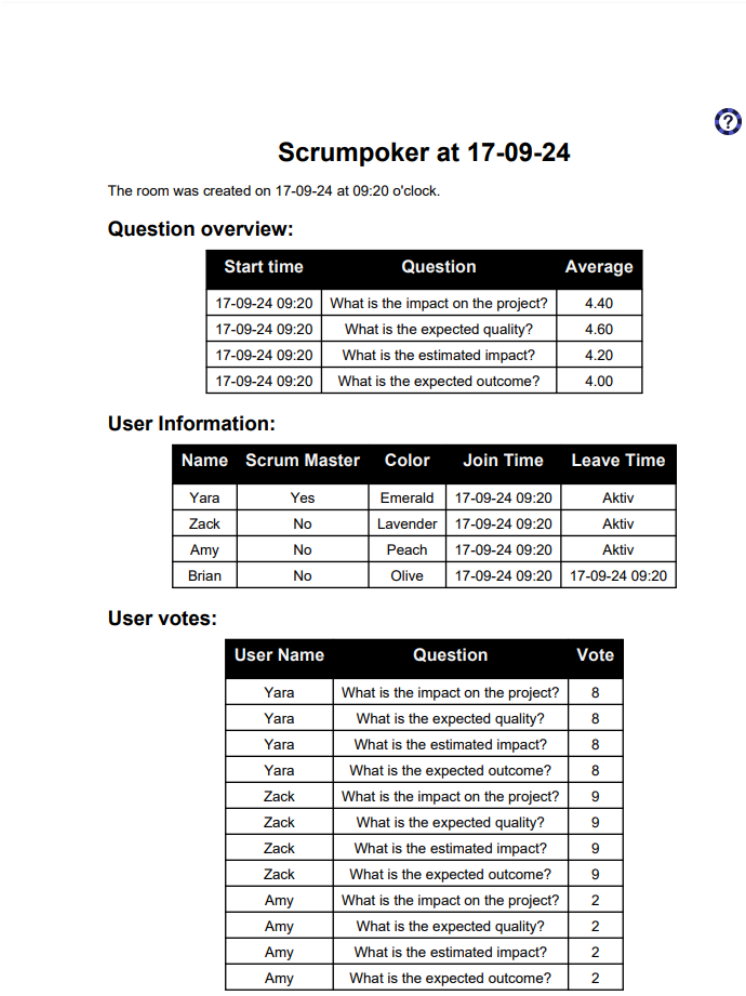
Listing 44: Scrum Master setzen

```
1  public User setScrumMaster(String roomId, int userId) throws Exception {
2          if (roomId == null || roomId.isEmpty()) {
3              throw new IllegalArgumentException("Room ID can't be null or empty.");
4          }
5
6          try {
7              deleteScrumMaster(roomId);
8          } catch (SQLException e) {
9              throw new Exception("Error while deleting Scrum Master: " + e.getMessage(), e);
10         }
11
12         String query = "UPDATE \"User\" SET isscrummaster = TRUE WHERE room_id = ? AND
               user_id = ?";
13
14         try (PreparedStatement statement = connection.prepareStatement(query)) {
15             statement.setString(1, roomId);
16             statement.setInt(2, userId);
17
18             int rowsAffected = statement.executeUpdate();
19             if (rowsAffected > 0) {
20                 System.out.println("Scrum Master added successfully.");
21             } else {
22                 throw new SQLException("Error: Scrum Master couldn't be added.");
23             }
24         } catch (SQLException e) {
25             throw e;
26         }
27
28         try {
29             return getUpdatedUser(userId, roomId);
30         } catch (Exception e) {
31             throw new Exception(e.getMessage(), e);
32         }
33     }
```

4.5 Python API

4.5.1 Einführung

In der Diplomarbeit wurde zusätzlich eine Python-API implementiert, die Analysen über die Daten durchführt, die während eines ScrumPoker-Spiels gespeichert werden. Außerdem protokolliert die Python-API, zu welchen Fragen abgestimmt wurde und welcher Benutzer was abgestimmt hat. Das PDF kann jederzeit von jedem heruntergeladen werden. Wenn man auf 'Download Data' klickt, erscheint ein Pop-up am rechten oberen Rand, und der Download des PDFs startet. Das fertige PDF sieht wie in Abbildung 34 dargestellt aus.



Scrumpoker at 17-09-24

The room was created on 17-09-24 at 09:20 o'clock.

Question overview:

Start time	Question	Average
17-09-24 09:20	What is the impact on the project?	4.40
17-09-24 09:20	What is the expected quality?	4.60
17-09-24 09:20	What is the estimated impact?	4.20
17-09-24 09:20	What is the expected outcome?	4.00

User Information:

Name	Scrum Master	Color	Join Time	Leave Time
Yara	Yes	Emerald	17-09-24 09:20	Aktiv
Zack	No	Lavender	17-09-24 09:20	Aktiv
Amy	No	Peach	17-09-24 09:20	Aktiv
Brian	No	Olive	17-09-24 09:20	17-09-24 09:20

User votes:

User Name	Question	Vote
Yara	What is the impact on the project?	8
Yara	What is the expected quality?	8
Yara	What is the estimated impact?	8
Yara	What is the expected outcome?	8
Zack	What is the impact on the project?	9
Zack	What is the expected quality?	9
Zack	What is the estimated impact?	9
Zack	What is the expected outcome?	9
Amy	What is the impact on the project?	2
Amy	What is the expected quality?	2
Amy	What is the estimated impact?	2
Amy	What is the expected outcome?	2

Abbildung 34: Python API PDF

4.5.2 Datenbank Verbindung

Damit die Anwendung auf die notwendigen Daten zugreifen kann, ist eine stabile Datenbankverbindung erforderlich. Die Verbindung ermöglicht es, Daten aus der Datenbank zu lesen. Sie dient als zentrale Schnittstelle zwischen der Anwendung und der Datenbank, in der Informationen wie Benutzeraktivitäten, Fragen oder Abstimmungsergebnisse gespeichert sind.

In unserer Anwendung wird PostgreSQL als Datenbank verwendet, da sie sich durch hohe Zuverlässigkeit und umfassende Funktionalität auszeichnet. Die Verbindung wird über die psycopg2-Bibliothek hergestellt, die leistungsstarke Tools für die Interaktion mit PostgreSQL bereitstellt. Eine Verbindungspool-Architektur sorgt dafür, dass die Ressourcen effizient genutzt werden.

Die folgende Funktion stellt eine Datenbankverbindung bereit, indem sie einen Verbindungspool nutzt, wie in dem Listing 45 ersichtlich.

Listing 45: Datenbank Verbindung

```
1
2  connection_pool = pool.SimpleConnectionPool(
3      minconn=1,
4      maxconn=10,
5      host="schaetzpoker-db",
6      database="schaetzpoker",
7      user=os.getenv("DB_USER", "postgres"),
8      password=os.getenv("DB_PASSWORD", "***********"),
9      port=5432
10 )
11
12 status = {"generating": False}
13
14     def get_db_connection():
15     """Get a connection from the connection pool."""
16     try:
17         return connection_pool.getconn()
18     except psycopg2.Error as e:
19         logger.error(f"Database connection error: {e}")
20         abort(500, description="Database connection error")
```

4.5.3 Ablauf Protokollieren

Für die Protokollierung gibt es in der Datenbank eigene Log-Tabellen, die Timestamps speichern. Hier ein kleines Beispiel mit Dummy-Daten, um die Python-API besser zu verstehen. Unsere Python-API protokolliert, was bei einem ScrumPoker-Spiel abgespielt wurde. Dabei wird das PDF in vier Teile unterteilt:

- Datum und Uhrzeit
  - Zeigt an, wann gespielt worden ist mit Datum und Uhrzeit.
- Frage Übersicht (Question overview:)
  - Zeigt, was alles für Fragen gestellt worden sind, wann sie gestellt worden sind und was für ein Durchschnitt raus gekommen ist.
- Teilnehmerübersicht (User Information:)
  - Zeigt alle Teilnehmer mit ihren Namen, ob sie noch im Raum sind, wer der Scrum Master in dem Raum ist, welche Farbe sie haben und wann sie dem Raum beigetreten sind.
  - Wenn sie noch im Raum sind steht bei Leave Time 'Aktiv' und wenn sie bereits den Raum verlassen haben sieht man die Uhrzeit, wann sie den Raum verlassen haben.
- Abstimmungen (User votes:)
  - Zeigt den Teilnehmername, die Frage und was sie für die jeweilige Frage abgestimmt haben.

4.5.4 Die Vier Teilbereiche des PDF's

Nun noch zu den vier Teilbereichen noch etwas genauer. Wie die Daten aus der Datenbank geladen werden und was genau in jedem Teilbereich ausgegeben wird.

Datum und Uhrzeit

In einem PDF-Dokument kann das aktuelle Datum und die Uhrzeit mithilfe von Python generiert und formatiert werden. Dabei wird das datetime-Modul verwendet, um die gewünschte Ausgabe in einem ansprechenden Format darzustellen. Der folgende Python-Code zeigt, wie der Titel mit Datum und Uhrzeit erstellt wird Listing 46:

Listing 46: Ausgabe von Datum und Uhrzeit

```
1
2  title_with_date = f"<b>Scrumpoker</b> at {datetime.datetime.now().strftime('%d-%m-%y')}"
3      elements.append(Paragraph(title_with_date, title_style))
```

Hier wird das Format `%d-%m-%y` verwendet, das man wie folgt verstehen kann:

- `%d`: Der Tag des Monats (zweistellig, z. B. 27 für den 27. Januar).
- `%m`: Der Monat (zweistellig, zb. 01 für Januar).
- `%y`: Das Jahr (zweistellig, zb. 25 für 2025).

Ein Beispiel für das Format ist die Ausgabe `27-01-25`. Dabei werden das Datum und die Uhrzeit dynamisch generiert und passen sich stets dem aktuellen Zeitpunkt an.

Frageübersicht (Question Overview):

Die Frageübersicht bietet eine tabellarische Darstellung aller gestellten Fragen in einem Raum. Dort werden Daten wie Startzeit und Durchschnitt der abgegebenen Bewertungen erfasst. Anders als bei Datum und Uhrzeit wird bei der Frageübersicht auf die Datenbank zugegriffen, und zwar auf die Tabelle QuestionLog. Von dieser Tabelle holt man sich nun: creatin_time, Text und den Durchschnitt der jeweiligen Frage. Wie man im Listing 47 sieht.

Listing 47: Log Data Query from QuestionLog

```
1
2      cur.execute('SELECT creation_time, text, average_vote FROM "QuestionLog" WHERE room_id
           = %s;', (room_id,))
3      log_data = cur.fetchall()
```

Diese Query wird ausgeführt und holt die Daten aus der Datenbank.

Damit eine Tabelle ausgegeben wird, erstellt die Python-API zunächst eine leere Tabelle mit den Tabellenüberschriften Start time, Question und Average Vote. Anschließend werden die Daten aus der Tabelle QuestionLog in die neu erstellte Tabelle eingefügt, dies ist ersichtlich in Listing 48.

Listing 48: Tabelle Frageübersicht

```
1
2       elements.append(Paragraph("Question overview:", styles['Heading2']))
3       question_data = [["Start time", "Question", "Average Vote"]]
4       for entry in log_data:
5           creation_time, question_text, average_vote = entry
6           formatted_time = creation_time.strftime('%d-%m-%y %H:%M')
7           question_data.append([
8               formatted_time,
9               question_text,
10              f"{average_vote:.2f}" if average_vote is not None else "N/A"
11          ])
12      question_table = Table(question_data)
13      question_table.setStyle(TableStyle([
14          ('BACKGROUND', (0, 0), (-1, 0), colors.black),
15          ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
16          ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
17          ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
18          ('FONTSIZE', (0, 0), (-1, 0), 12),
19          ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
20          ('BACKGROUND', (0, 1), (-1, -1), colors.white),
21          ('GRID', (0, 0), (-1, -1), 1, colors.black),
22      ]))
23      elements.append(question_table)
```

Hier wird das Format `%d-%m-%y %H:%M` verwendet, das man wie folgt verstehen kann:

- `%d`: Der Tag des Monats.
- `%m`: Der Monat (zweistellig, z. B. 01 für Januar).
- `%y`: Das Jahr (zweistellig, z. B. 25 für 2025).
- `%H`: Die Stunde im 24-Stunden-Format (zweistellig, z. B. 14 für 14 Uhr).
- `%M`: Die Minute (zweistellig, z. B. 05 für fünf Minuten nach der vollen Stunde).

Teilnehmerübersicht (User Information):

Die Teilnehmerübersicht zeigt alle Nutzer eines Raums in einer übersichtlichen Tabelle. Angezeigt werden der Name des Benutzers, seine zugewiesene Farbe sowie seine Eintritts- und Austrittszeit. Ist ein Benutzer noch im Raum, erscheint im Feld „Leave Time“ statt einer Uhrzeit der Status „Aktiv“. Das Zeitformat entspricht dem der Frageübersicht. Zudem gibt es ein Feld „Scrum Master“, das anzeigt, wer diese Rolle übernimmt. Alle Daten stammen aus der Datenbank, genauer aus der Tabelle 'UserLog'. Wie in dem Listing 49 ersichtlich.

Listing 49: Log Data Query from UserLog

```
1
2       cur.execute('SELECT name, isScrumMaster, color, action_time, end_time FROM "UserLog"
            WHERE room_id = %s;', (room_id,))
3           user_data = cur.fetchall()
```

Mit dieser Query werden die Daten aus der Tabelle 'UserLog' geholt, für den benötigten Raum.

Wie Listing 50 zeigt, erfolgt der Aufbau der Tabelle nach dem gleichen Prinzip wie bei der Fragenübersicht.

Listing 50: Tabelle Teilnehmerübersicht

```
1
2       elements.append(Paragraph("User information:", styles['Heading2']))
3       user_table_data = [["Name", "Scrum Master", "Join Time", "Leave Time"]]
4       added_users = set()
5       for user in user_data:
6           name, isScrumMaster, color, action_time, end_time = user
7           if name not in added_users:
8               user_table_data.append([
9                   name,
10                  "Yes" if isScrumMaster else "No",
11                  action_time.strftime('%d-%m-%y %H:%M') if action_time else "N/A",
12                  end_time.strftime('%d-%m-%y %H:%M') if end_time else "Active"
13              ])
14              added_users.add(name)
15      user_table = Table(user_table_data)
16      user_table.setStyle(TableStyle([
17          ('BACKGROUND', (0, 0), (-1, 0), colors.black),
18          ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
19          ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
20          ('GRID', (0, 0), (-1, -1), 1, colors.black),
21      ]))
22      elements.append(user_table)
```

Abstimmungen (User votes):

Listing 51 zeigt die Abfrage nach einer ausgewählten Karte eines Benutzers.

Listing 51: Log Data Query from SelectedCardChangeLog

```
1
2           cur.execute('''SELECT user_name, question_text, selected_card, room_id FROM
                "SelectedCardChangeLog"
3                           WHERE action_time IN (SELECT action_time FROM
                                "SelectedCardChangeLog" WHERE user_name IS NOT NULL AND
                                question_text IS NOT NULL AND room_id = %s)
4                           ORDER BY user_name, question_text, action_time DESC;''', (room_id,))
5           vote_data = cur.fetchall()
```

Mit dieser Query werden die Daten aus der Tabelle 'SelectedCardChangeLog' geholt, für den benötigten Raum. Dabei werden nur Einträge berücksichtigt, bei denen user_name und question_text nicht NULL sind. Die Ergebnisse werden nach Benutzername, Frage und der neuesten Aktion sortiert.

Die Struktur der Tabelle folgt dem gleichen Muster wie bei der Teilnehmer- oder Fragenübersicht: Eine leere Tabelle wird erzeugt und anschließend mit Daten befüllt wie in dem Listing 52 ersichtlich.

Listing 52: Tabelle Abstimmung

```
1
2       elements.append(Paragraph("Vote details:", styles['Heading2']))
3       vote_table_data = [["User", "Question", "Selected Card"]]
4       for row in vote_data:
5           if len(row) == 4:
6               user_name, question_text, selected_card, room_id = row
7               vote_table_data.append([user_name, question_text, selected_card])
8       vote_table = Table(vote_table_data)
9       vote_table.setStyle(TableStyle([
10          ('BACKGROUND', (0, 0), (-1, 0), colors.black),
11          ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
12          ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
13          ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
14          ('FONTSIZE', (0, 0), (-1, 0), 12),
15          ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
16          ('BACKGROUND', (0, 1), (-1, -1), colors.white),
17          ('GRID', (0, 0), (-1, -1), 1, colors.black),
18      ]))
19      elements.append(vote_table)
```

4.5.5 API Endpunkt: `/Download/_room_id`

Der Endpunkt `/Download/_room_id` dient zur Generierung und Erstellung eines detaillierten PDF-Berichts zu einem ScrumPoker-Spiel. Durch die Mitgabe einer `room_id` in der URL werden alle relevanten Daten dieser Session aus der Datenbank abgerufen, verarbeitet und als PDF-Dokument formatiert.

1. Empfang der `room_id`
    a) Die API empfängt die `room_id` aus der URL, somit weiß sie genau welche Daten, für die Abfrage benötigt werden.
2. Abruf der Daten aus der Datenbank
    a) Es werden mehrere SQL-Abfragen ausgeführt, um verschiedenste Daten des Raums mit der `room_id` d zu erhalten. Dazu gehören:
        i. Fragen und deren Durchschnittsbewertung aus der QuestionLog-Tabelle
        ii. Teilnehmerinformation wie Name, Scrum Master, Beitrittszeit und Verlassen-Zeit aus der UserLog- Tabelle.
        iii. Abstimmungsergebnisse aus der SelectedCardChangeLog-Tabelle
3. Erstellung des PDF-Dokuments
    a) Die Daten werden strukturiert und in verschiedene Abschnitte aufgeteilt, nämlich in die Teilebereiche des PDFs wie oben beschrieben.
        i. Datum und Uhrzeit
        ii. Frageübersicht
        iii. Teilnehmer
        iv. Abstimmungen
4. Bereitstellung des PDF-Downloads
    a) Sobald das PDF fertig ist, wird es als Datei über den Endpunkt zum Download bereitgestellt.
    b) Der Nutzer erhält das PDF als Antwort und kann es lokal speichern

4.6 Datenbank

4.6.1 Einführung

Um alle Datensätze zentral zu speichern, wird eine PostgreSQL-Datenbank in einem Docker-Container benutzt. PostgreSQL ist eine Datenbank, die stabil läuft, viele Möglichkeiten bietet und kostenlos ist. Mit dem Docker-Container bekommt man eine eigene, abgeschlossene Umgebung, die gut in unsere bestehende Infrastruktur passt.

Da in unserem Fall nur strukturierte Daten kleinerer Mengen gespeichert werden und die Sicherstellung von Integrität und Konsistenz essenziell ist, stellt das objektrelationale Modell mit PostgreSQL die ideale Lösung dar. Das ERD-Diagramm sieht wie in der nachfolgenden Abbildung 35 ersichtlich aus.

4.6.2 ERD

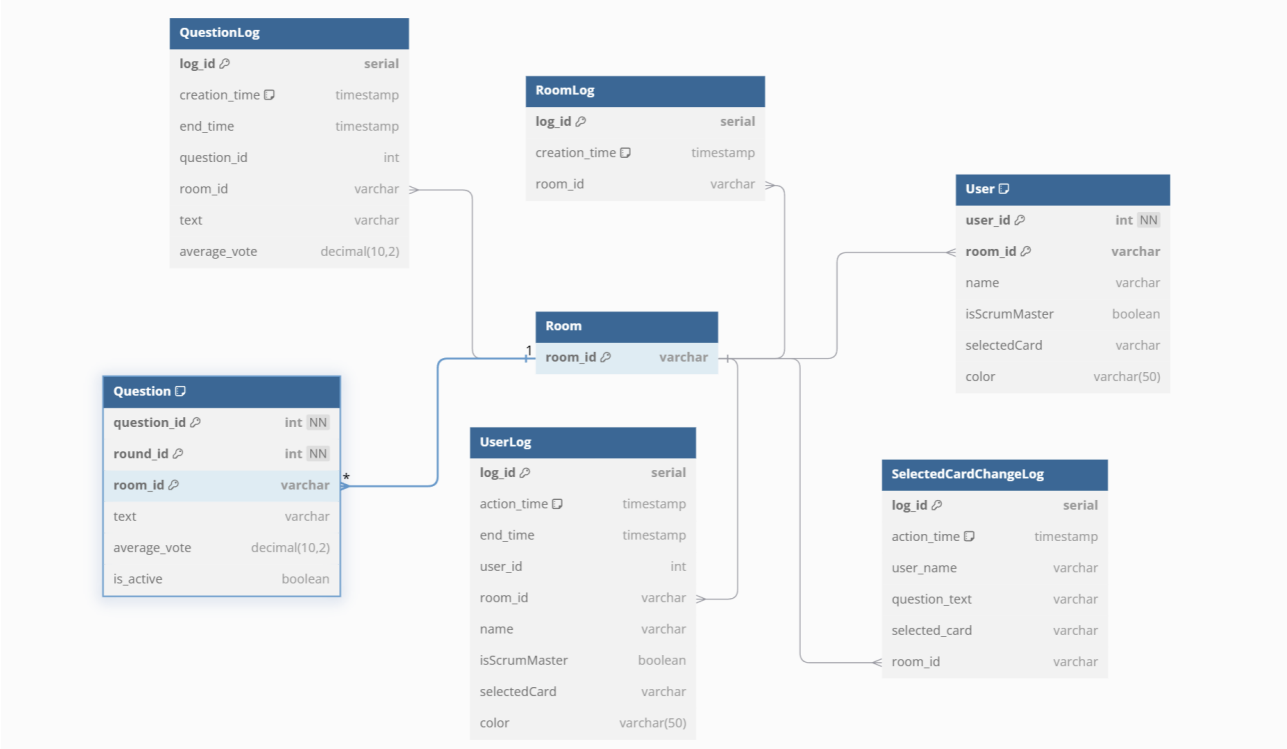


Abbildung 35: Datenmodell

Diese Abbildung bietet einen Überblick über die existierenden Entitäten und deren Beziehungen. Im weiteren Verlauf wird auf die einzelnen Komponenten im Detail eingegangen.

4.6.3 Umsetzung

Room Tabelle

Alle Räume, die erstellt werden, sind in dieser Tabelle gespeichert. Die Room-Tabelle ist wie in der folgenden Tabelle aufgebaut Tabelle 8

Name des Attributs	Datentyp	Beschreibung
room_id	varchar(255)	Die room_id ist das einzige Attribut der Tabelle Room und ist somit auch zugleich der Primärschlüssel. Sie identifiziert jeden Raum eindeutig. Der Primärschlüssel wird aus einer zufälligen Zeichenfolge von 6 Zeichen generiert, die aus den Zeichen 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789' besteht.

Tabelle 8: Tabelle mit Attributen der Room-Tabelle

User Tabelle

Alle User, die bei einer Schätzung teilnehmen, werden in der Tabelle 'User' gespeichert. Welche aufgebaut ist, wie in Tabelle 9 dargestellt.

Question Tabelle

Alle Fragen, die in einem Raum stattfinden, werden in der Tabelle 'Question' gespeichert. Die Question-Tabelle ist, wie in der nachstehenden Tabelle 10 aufgebaut.

Name des Attributs	Datentyp	Beschreibung
user_id	int	Die user_id stellt zusammen mit der room_id den Primärschlüssel dar und macht jeden Datensatz eindeutig.
room_id	varchar(255)	Stellt zusammen mit user_id den Primärschlüssel dar. Dieser wird benötigt, um zu identifizieren, in welchem Raum welcher Benutzer ist beziehungsweise war. Er ist ein Fremdschlüssel und verweist auf den Primärschlüssel der Tabelle Room.
name	varchar(255)	Speichert den Namen des Benutzers.
isScrumMaster	boolean	Bestimmt, ob der User Scrum Master ist oder nur ein Teammitglied. true bedeutet, er ist Scrum Master und darf Fragen stellen. false bedeutet, er ist kein Scrum Master und darf nur bei den Fragen abstimmen.
selectedCard	varchar(255)	Speichert, welche Fibonacci-Zahl ausgewählt wurde.
color	varchar(50)	Bestimmt die Darstellungsfarbe des Benutzers. Die Farbe wird als Hex-Farbcode angegeben.

Tabelle 9: Tabelle mit Attributen der User-Tabelle

Name des Attributs	Datentyp	Beschreibung
question_id	int	Die question_id stellt zusammen mit room_id und round_id den Primärschlüssel dar und macht jeden Datensatz eindeutig.
round_id	int	Stellt zusammen mit question_id und room_id den Primärschlüssel dar. Die round_id wird benötigt, um anzuzeigen, wie viele Versuche man bei einer Frage brauchte, um auf einen gemeinsamen Nenner zu kommen.
room_id	varchar(255)	Stellt zusammen mit question_id und round_id den Primärschlüssel dar. Room_id wird benötigt, um zu speichern, in welchem Raum welche Frage gespeichert wird.
text	varchar(255)	Speichert die Frage im Klartext.
average_vote	decimal(10,2)	Speichert den Durchschnitt, der nach Beendigung einer Frage entsteht.
is_active	boolean	Bestimmt, ob bei dieser Frage immer noch gespielt wird oder ob bereits eine andere Frage gestellt wird.

Tabelle 10: Tabelle mit Attributen der Question-Tabelle

4.6.4 Log Tabellen

Für jede Haupttabelle existiert eine dazugehörige Log-Tabelle. Diese speichert zusätzlich Timestamps, die schlussendlich für die Python-API-Analysen verwendet werden. Die Daten in den Log-Tabellen werden durch Trigger eingefügt.

Bei jeder Tabelle gibt es einen Trigger, der ausgelöst wird, wenn eine neue Zeile in die Haupttabelle eingefügt wird. In diesem Fall wird die Zeile eins zu eins in die entsprechende Log-Tabelle kopiert, wobei zusätzlich ein Timestamp als Startzeit beziehungsweise Join-Zeit gespeichert wird.

Ebenso gibt es Trigger für das Löschen oder Ändern von Daten. Beispielsweise wird bei der Löschung eines Benutzers in der Log-Tabelle dessen Endzeit gespeichert, um nachverfolgen zu können, wann der Benutzer die Sitzung verlassen hat. Auch Änderungen an ausgewählten Karten oder Fragen werden entsprechend geloggt, um eine vollständige Nachverfolgbarkeit zu gewährleisten.

In einer ScrumPoker-Umgebung ist es essenziell, sämtliche relevanten Aktivitäten und Ereignisse in einer Datenbank zu speichern, um eine vollständige Nachverfolgbarkeit und spätere Analysen zu ermöglichen. Dazu dienen Log-Tabellen, die verschiedene Informationen zu Räumen, Benutzern, Fragen und Kartenänderungen speichern. Jede dieser Tabellen enthält spezifische Attribute, die eine detaillierte Protokollierung erlauben.

In einem ScrumPoker-Spiel speichert man viel in einer Datenbank, für eine spätere Analyse. Für diesen Fall gibt es Log-Tabellen. Sie speichern nahezu dasselbe wie die normalen Tabellen. Der kleine und feine Unterschied ist jedoch, dass Log-Tabellen zusätzlich noch mehr Attribute zur detaillierten Protokollierung speichern.

RoomLog Tabelle

Die 'RoomLog-Tabelle' speichert alle erstellten Räume sowie den Zeitpunkt ihrer Erstellung. Diese Informationen sind essenziell, um nachvollziehen zu können, wann welche Räume genutzt wurden und für welche Zeiträume sie existierten. Dies ist in Tabelle 11 ersichtlich.

UserLog Tabelle

Die 'UserLog-Tabelle' protokolliert die Benutzeraktivitäten in den Räumen. Sie speichert, wann ein Benutzer einem Raum beigetreten ist, wann er diesen verlassen hat und ob er die Rolle des Scrum Masters hatte. Die 'UserLog-Tabelle' ist in Tabelle 12 ersichtlich.

Name des Attributs	Datentyp	Beschreibung
log_id	serial	Dieses Attribut ist der Primärschlüssel und macht jeden Datensatz eindeutig.
creation_time	timestamp	Speichert das Erstellungsdatum und die Erstellungszeit des Raumes.
room_id	varchar(255)	Ist ein Fremdschlüssel, der auf die Tabelle Room referenziert, um eine Verbindung zum Raum herzustellen.

Tabelle 11: Tabelle mit Attributen der RoomLog-Tabelle

Name des Attributs	Datentyp	Beschreibung
log_id	serial	Dieses Attribut ist der Primärschlüssel und stellt sicher, dass jeder Eintrag eindeutig ist.
action_time	timestamp	Speichert den Zeitpunkt, zu dem der Benutzer dem Raum beigetreten ist.
end_time	timestamp	Speichert den Zeitpunkt, zu dem der Benutzer den Raum verlassen hat. Bleibt leer, falls der Benutzer noch aktiv ist.
user_id	int	Speichert die ID des Benutzers, um diesen eindeutig zu identifizieren.
room_id	varchar(255)	Verweist auf den Raum, in dem sich der Benutzer befindet.
name	varchar(255)	Speichert den Namen des Benutzers.
isScrumMaster	boolean	Gibt an, ob der Benutzer Scrum Master ist (true/false).
selectedCard	varchar(255)	Speichert die aktuell ausgewählte Karte des Benutzers während einer Abstimmung.
color	varchar(50)	Speichert die Farbe des Benutzers, die zur besseren Visualisierung genutzt wird.

Tabelle 12: Tabelle mit Attributen der UserLog-Tabelle

QuestionLog Tabelle

Die 'QuestionLog-Tabelle', abgebildet in Tabelle 13, speichert alle Fragen, die in einem Raum während eines ScrumPoker-Spiels gestellt wurden. Sie enthält außerdem Informationen darüber, wie lange die Frage aktiv war und welches durchschnittliche Abstimmungsergebnis erzielt wurde.

Name des Attributs	Datentyp	Beschreibung
log_id	serial	Primärschlüssel, der jeden Datensatz eindeutig identifiziert.
creation_time	timestamp	Speichert den Zeitpunkt, an dem die Frage gestellt wurde.
end_time	timestamp	Speichert den Zeitpunkt, zu dem die Abstimmung zu dieser Frage abgeschlossen wurde.
question_id	int	Enthält die ID der jeweiligen Frage.
room_id	varchar(255)	Speichert die ID des Raums, in dem die Frage gestellt wurde.
text	varchar(255)	Enthält die eigentliche Frage als Text.
average_vote	decimal(10,2)	Speichert den Durchschnitt der abgegebenen Stimmen für die Frage.

Tabelle 13: Tabelle mit Attributen der QuestionLog-Tabelle

SelectedCardChangeLog Tabelle

Die 'SelectedCardChangeLog-Tabelle' dokumentiert alle Änderungen der ausgewählten Karten, dies ist in Tabelle 14 zu sehen.

Name des Attributs	Datentyp	Beschreibung
log_id	serial	Eindeutiger Primärschlüssel für jeden Log-Eintrag.
action_time	timestamp	Speichert den Zeitpunkt der Kartenänderung.
user_id	int	Enthält die ID des Benutzers, der seine Karte geändert hat.
room_id	varchar(255)	Speichert die ID des Raums, in dem die Änderung stattgefunden hat.
selectedCard	varchar(255)	Speichert die neue ausgewählte Karte des Benutzers.

Tabelle 14: Tabelle mit Attributen der SelectedCardChangeLog-Tabelle

5 Deployment

Die REST-API wird als Microservice in einem Docker-Container bereitgestellt. Ebenso werden auch die anderen Teile dieser Anwendung in einem Docker-Container gehostet. Dies hat den Vorteil, dass es einfach zu skalieren ist und ohne großen Mehraufwand in unterschiedlichen Umgebungen gehostet werden kann. Um die Docker-Container zu erstellen, wurden docker-compose-Dateien genutzt, über welche die Anwendung mittels eines docker-compose-Befehls gestartet werden kann. Diese 'docker-compose.yml'-Datei in Listing 53 greift auf die jeweiligen Dockerfiles in den einzelnen Programmteilen zu. Das Dockerfile beinhaltet drei unterschiedliche Services, einmal die API unter dem Namen 'app', dann das Frontend unter 'frontend' und die Python-API unter 'python'. Der Container erhält den definierten Namen und nicht einen zufällig generierten Namen. Das Service wird mit dem Dockerfile des Backends im '/Backend'-Verzeichnis erstellt. Diese Dockerfiles werden in Listing 54, 55 und 56 gezeigt. Als Nächstes werden die Environment-Variablen gesetzt, welche hier beispielhaft mit den Standarddaten gesetzt werden. Der Port, auf welchem das Backend läuft, nachdem der Docker-Container gestartet wurde, wird unter 'ports' festgelegt. Links befindet sich der 'Host Port', dieser gibt an, auf welchem Port die Anwendung letztendlich am Host erreichbar ist. Der zweite angegebene Port ist der 'Container Port'. Der Host empfängt also die Anfrage auf Port 8080 und die Anfrage wird an den Container weitergeleitet. Wenn man den 'Host Port' ändert, läuft die Anwendung im Container weiterhin auf Port 8080, ist aber von außerhalb dann auf dem 'Host Port' erreichbar. Es wurde ein 'network' erstellt, mit dem Namen 'scrumpoker-network'. Dieses schafft eine isolierte Umgebung, in welcher die Anwendungen untereinander besser kommunizieren können. [37]

Das Frontend wird mittels des Dockerfile im Verzeichnis '/Frontend/frontend' gebaut. Es ist unter dem Port 80, also dem Standard-HTTP-Port, erreichbar. Unter diesem läuft es dann auf einem Webserver, welcher bei unserem Auftraggeber genutzt wird. Im Entwicklungsmodus ist es unter dem Port 4200 erreichbar.

Die Python-API wird ebenfalls in dieser 'docker-compose.yml' gestartet. Dafür wird ebenfalls ein Dockerfile genutzt. Die Python-API läuft auf Port 5000 und es werden die Environment-Variablen wieder beispielhaft hier dargestellt. Diese werden benötigt, um die Python-API zur Datenbank verbinden zu können. Die Python-API wird wie die anderen Container ebenfalls im 'scrumpoker-network' angelegt, um mit den anderen Containern kommunizieren zu können.

Listing 53: Docker-Compose Datei für die Anwendung

```
services:
  app:
    image: scrumpokerapi:latest
    container_name: scrumpoker-api
    build:
      context: ./Backend
      dockerfile: Dockerfile
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://schaetzpoker-db:5432/schaetzpoker
      SPRING_DATASOURCE_USERNAME: postgres
      SPRING_DATASOURCE_PASSWORD: Passme1234!
    ports:
      - "8080:8080"
    networks:
      - scrumpoker-network

  frontend:
    image: angular-frontend:latest
    container_name: scrumpoker-frontend
    build:
      context: ./Frontend/frontend
      dockerfile: Dockerfile
    ports:
      - "80:80"
      - "4200:4200"
    networks:
      - scrumpoker-network

  python:
    container_name: scrumpoker-python
    build:
      context: ./Datenbank/Analyses
    ports:
      - "5000:5000"
    environment:
      - DATABASE_HOST=localhost
      - DATABASE_PORT=5432
      - DATABASE_NAME=schaetzpoker
      - DATABASE_USER=postgres
      - DATABASE_PASSWORD=Passme1234!
    networks:
      - scrumpoker-network

networks:
  scrumpoker-network:
    external: true
```

Das Dockerfile des Backends in Listing 54 benötigt die Abhängigkeit 'openjdk:17-jdk-alpine', um die Java Spring Boot API starten zu können. Die Java-API wird aus den Target-Dateien erstellt, diese werden im Container als '/scrumpokerapi-0.0.1.jar' gespeichert. Diese Target-Dateien werden beim Ausführen des Build-Commands von Spring Boot automatisch generiert. Der Entrypoint definiert das Startkommando für den Container, sodass die API automatisch gestartet wird, wenn der Container gestartet wird. [38]

Listing 54: Dockerfile Backend

```
    FROM openjdk:17-jdk-alpine
    LABEL baeldung.com
    COPY TargetDateien/target/scrumpokerapi-0.0.1-SNAPSHOT.jar scrumpokerapi-0.0.1.jar
    ENTRYPOINT ["java","-jar","/scrumpokerapi-0.0.1.jar"]
```

Das Dockerfile des Frontends in Listing 55 beinhaltet die Abhängigkeit 'node:16-alpine' welche benötigt wird, um den Code bauen und kompilieren zu können. Das 'WORKDIR' gibt das Verzeichnis an, in welchem die Angular-Anwendung liegt. Dann wird der Befehl 'npm install'

ausgeführt, und alle Projekte werden vom ausgewählten Verzeichnis in den Docker-Container kopiert. So kann das Frontend dann gebaut werden. In diesem Dockerfile wird die Anwendung direkt gebaut, anders als beim Backend, wo man zuerst das Projekt bauen lassen muss und dann erst den Docker-Container starten kann. Dieses Dockerfile kann man direkt ausführen. Anschließend kommt noch der Teil, in welchem die Applikation gestartet wird. Es wird definiert, dass nginx genutzt werden soll, und die benötigten Dateien werden kopiert. [39]

Listing 55: Dockerfile Frontend

```
1       FROM node:16-alpine AS build
2       WORKDIR /app
3       COPY package*.json ./
4       RUN npm install
5       COPY . .
6       RUN npm run build --prod
7
8       FROM nginx:alpine
9       COPY --from=build /app/dist/frontend /usr/share/nginx/html
10      COPY nginx.conf /etc/nginx/conf.d/default.conf
```

Das Dockerfile in Listing 56 zeigt, dass das Docker-Image 'python:3.9-slim' genutzt wird. Folgend werden die Abhängigkeiten aus der 'requirements.txt' in den Container unter '/app' kopiert. Diese werden mittels 'pip install' installiert. In dieser Datei sind Abhängigkeiten, die benötigt werden, um die Python-API zu starten. Dann wird das gesamte Arbeitsverzeichnis in den Container kopiert und die Python-Datei 'app.py' wird mittels Kommandozeile ausgeführt.

Listing 56: Dockerfile Python-API

```
1  FROM python:3.9-slim
2  WORKDIR /app
3  COPY requirements.txt /app/
4  RUN pip install --no-cache-dir -r requirements.txt
5  COPY . /app/
6  EXPOSE 5000
7  CMD ["python", "app.py"]
```

Die PostgreSQL-Datenbank hat eine separate docker-compose.yml-Datei, welche in Listing 57 ersichtlich ist. Somit befindet sich die Datenbank auch in einem eigenen Docker-Container. Für die Datenbank wurde kein Dockerfile benötigt, da das 'postgres' Image verwendet wurde. In diesem Docker-Compose-File wurden wieder beispielhafte Umgebungsvariablen genutzt. Die Datenbank läuft auf Port 5432 und ist ebenso im Docker-Netzwerk 'scrumpoker-network'. Ebenso wurde ein Volume festgelegt, unter welchem dann die Daten in der Datenbank auch persistent gespeichert werden können. Wenn kein Volume konfiguriert ist, werden alle Daten auf dem Docker-Container beim Neustart gelöscht.

Listing 57: Docker-Compose für Datenbank

```
services:
  db:
    image: postgres:latest
    container_name: schaetzpoker-db
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: Passme1234!
      POSTGRES_DB: schaetzpoker
    ports:
      - "5432:5432"
    networks:
      - scrumpoker-network
    volumes:
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql

networks:
  scrumpoker-network:
    external: true
```

Diese Docker-Container wurden als Images an unseren Auftraggeber überreicht, zusätzlich wurde auch der gesamte Code unserer Anwendung nach umfangreicher Testung übergeben. Dazu wurden auch diese gesamten Compose- und Docker-Dateien an den Auftraggeber übergeben, sodass dieser die App auch selbst neu im Docker-Container bauen kann.

6 Ergebnis

6.1 Ionic Mobileapplication

Die ScrumPoker-App wurde erfolgreich entwickelt und getestet. Die Anwendung ermöglicht nun agilen Teams, Schätzungen in Echtzeit durchzuführen und die Ergebnisse automatisch zu dokumentieren.
Der folgende Abschnitt präsentiert die wichtigsten Ergebnisse und beschreibt den typischen Ablauf einer ScrumPoker-Sitzung.

6.1.1 Typischer Programmablauf

Um den Ablauf zu veranschaulichen, werden in der Abbildung 36 und Abbildung 37 jeweils drei Szenen der App visualisiert und im nächsten Schritt detailliert erläutert.

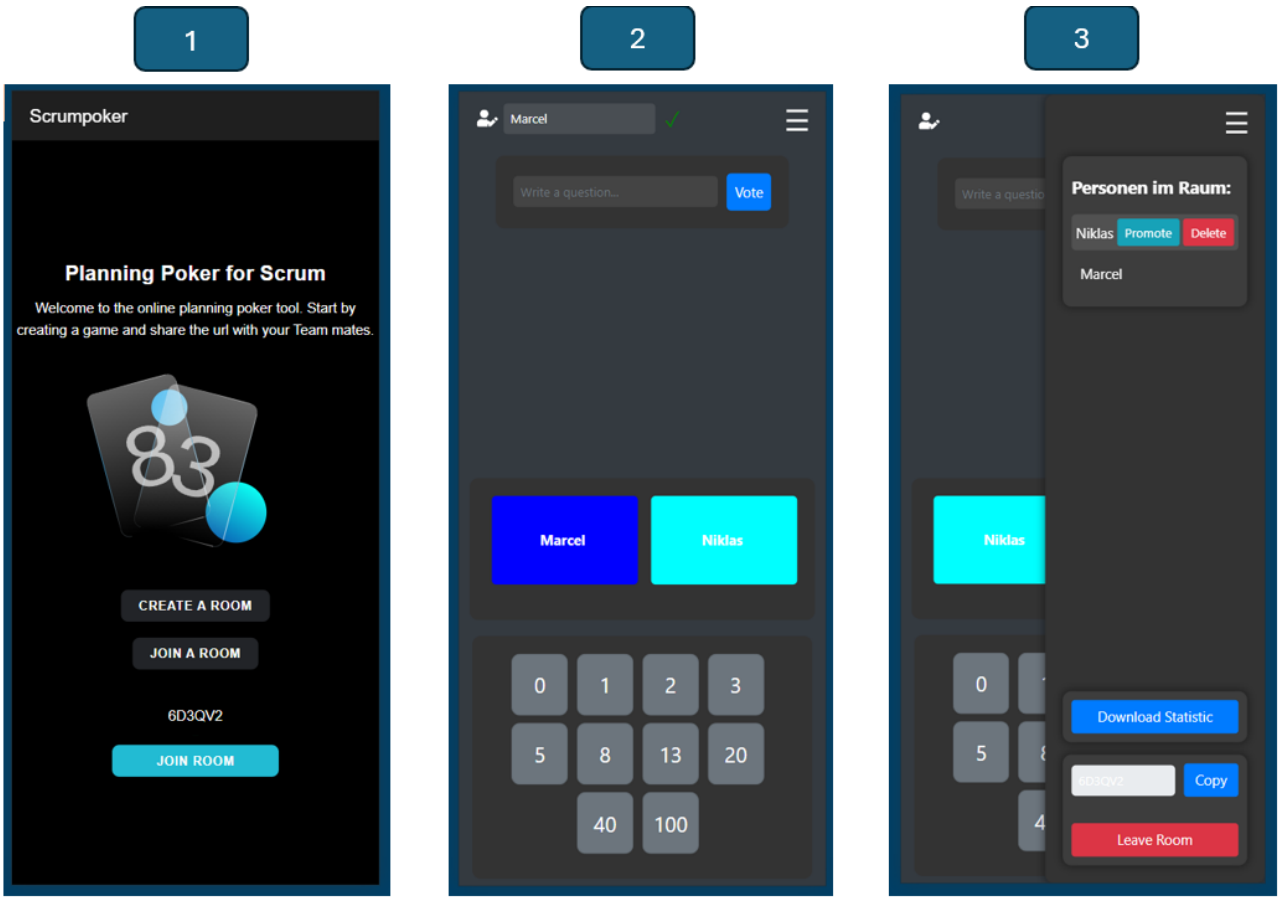


Abbildung 36: Programmablauf 1-3

(1) Raum erstellen oder beitreten

Beim Starten der App wird der Nutzer auf der Startseite mit einem kurzen Einleitungstext empfangen. Hierbei kann er entweder einen neuen Raum erstellen oder einem bestehenden Raum beitreten. Zum Beitreten muss der Nutzer den sechsstelligen Raumcode eingeben, den er zuvor vom Scrum Master erhalten hat. Daraufhin wird der Nutzer automatisch in den Spielraum hinzugefügt, wie in Szene zwei ersichtlich.

(2) Name ändern

Alle Teilnehmer sind nun in einem gemeinsamen Raum. Zur Identifikation haben sie die Möglichkeit, ihre Namen zu ändern. Dies wird in Echtzeit synchronisiert und für alle Teilnehmer sichtbar gemacht.

(3) Spieler verwalten

In der Menüleiste hat der Scrum Master die Rechte, Teilnehmer aus einem Raum zu entfernen oder zum Scrum Master zu befördern.
Andere Teilnehmer haben diese Berechtigungen nicht. Dennoch hat jeder Teilnehmer die Möglichkeit, mittels 'Leave Room' den Raum zu verlassen.

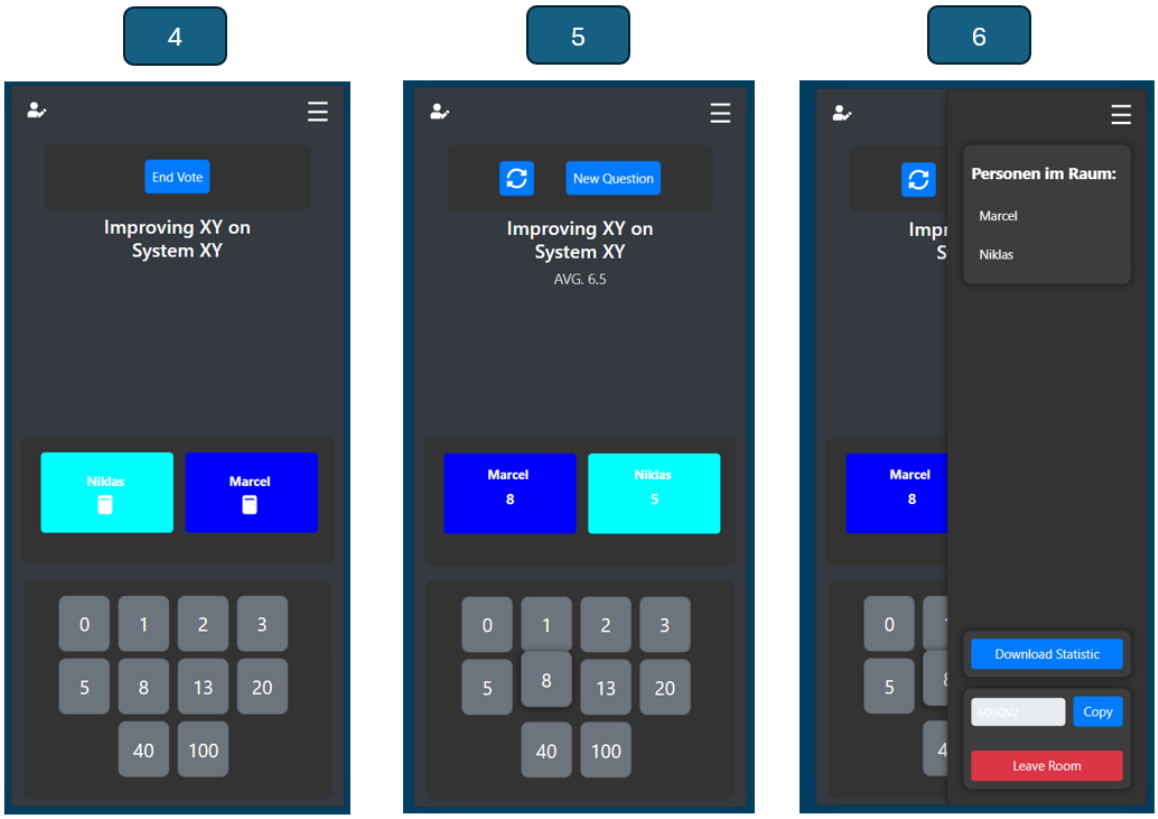


Abbildung 37: Programmablauf 4-6

(4) Frage stellen

Der Scrum Master hat die Berechtigung, eine Frage zu stellen, indem er diese in das Eingabefeld eintippt und auf 'Vote' klickt. Nachdem die Frage gestellt wurde, kann jeder Teilnehmer eine Karte mit einer Fibonacci-Zahl auswählen. Die Abstimmung läuft so lange, bis der Scrum Master die Runde mit einem Klick auf 'End Voting' beendet hat. Danach werden die Karten, wie in Szene fünf dargestellt, visualisiert.

(5) Ergebnisse auswerten

Nach Beendigung der Abstimmung werden alle Karten aufgedeckt und der Durchschnittswert angezeigt. Sind die Ergebnisse nicht zufriedenstellend, kann der Scrum Master dieselbe Frage mittels eines Klicks wiederholen. Sobald die Frage abgeschlossen ist, kann er mittels 'New Question' eine neue Frage starten.

(6) Statistik abrufen

Ist die Sitzung beendet, kann jeder Teilnehmer die Statistik über die vergangenen Abstimmungen abrufen. Dies geschieht mittels eines Klicks auf 'Download Statistic'. Dabei werden die Ergebnisse als PDF-Datei heruntergeladen.

6.2 Angular Webapplikation

Die ScrumPoker Website wurde erfolgreich programmiert und getestet. Agile Teams können nun über verschiedenen Regionen Aufwandsschätzungen betreiben. Der andere große Vorteil ist auch der Datenexport. Es muss jetzt keiner mehr mitschreiben. Im folgenden Abschnitt wird der richtige Ablauf einer ScrumPoker Session gezeigt.

Spiel Ablauf

Mittels Abbildungen und detaillierter Beschreibung wird der Prozess einer Sitzung dargestellt.

(1) Startseite

Wenn man die Webseite öffnet, kommt man zuerst zu einer Homepage. Bei dieser kann man dann auswählen, ob man einen Raum erstellen will oder mittels einer RaumID beitreten möchte. Man kann aber auch direkt mit der richtigen URL in den Raum gelangen und überspringt somit die Startseite. Als kleines Detail ist die Illustration mit einem Hover-Effekt ausgestattet. Die Startseite sieht man in der Abbildung 38.

Abbildung 38: Startseite - Web Anwendung

(2) Spielfeld

Nachdem man die ID eingegeben hat oder einen Raum erstellt hat, landet man auf dem Spielfeld. Dies kann je nach Rolle kleine Unterschiede beinhalten. Wenn man Scrum Master ist, dann verfügt man über die Möglichkeit, Fragen zu schreiben. Ebenfalls leitet man als Scrum Master das Spielfeld und entscheidet, wann weitergespielt wird. Tritt man nun einem Raum bei oder

gibt man den Posten als Scrum Master auf, verfügt man nicht mehr über diese Fähigkeiten. Das Spielfeld wird in der Abbildung 39 abgebildet.

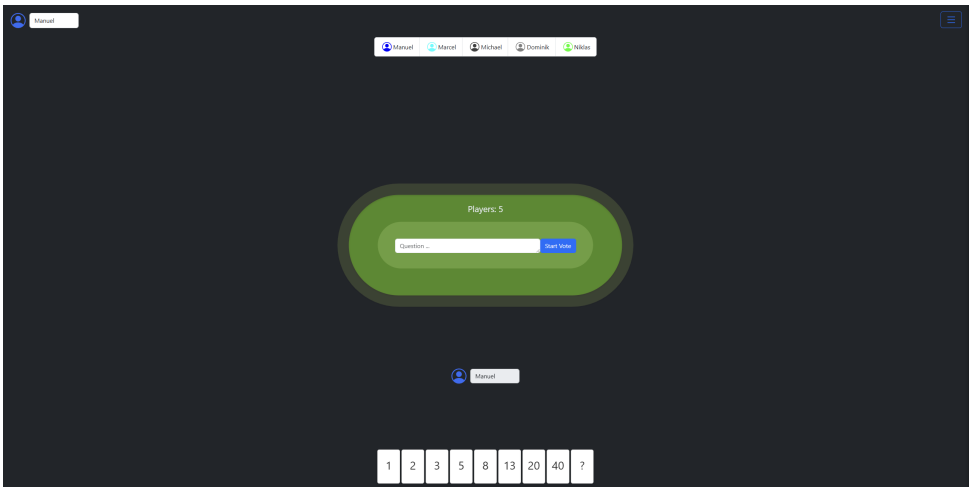


Abbildung 39: Spielfeld

(3) Spielablauf

Der Scrum Master gibt als Erstes eine Frage in die Mitte des Spielfelds ein und drückt dann Spielstart. Danach verfügen die anderen Benutzer und der Scrum Master über die Fähigkeit, abzustimmen. Das heißt, die Benutzer können mit dem Feld unten in der Mitte, also den Karten ihre Schätzung abgeben. In der Mitte sieht man auch, wie viele Benutzer nach abstimmen müssen. Sobald der Scrum Master dann die Abstimmung beendet, ändert sich die Mitte und man kann den Durchschnitt der Abstimmungen sehen. In den Abbildungen Abbildung 40, Abbildung 41 und Abbildung 42 wird der Ablauf veranschaulicht. Ebenfalls sieht man bei der Benutzerliste, wer welche Karte ausgewählt hat. Dies wird in der Abbildung 43 abgebildet.

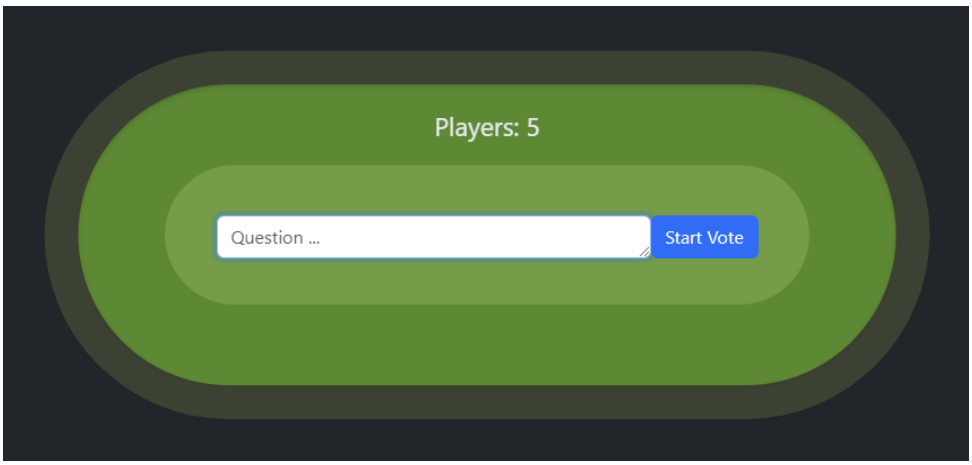


Abbildung 40: Spielfeld - Poker Tisch

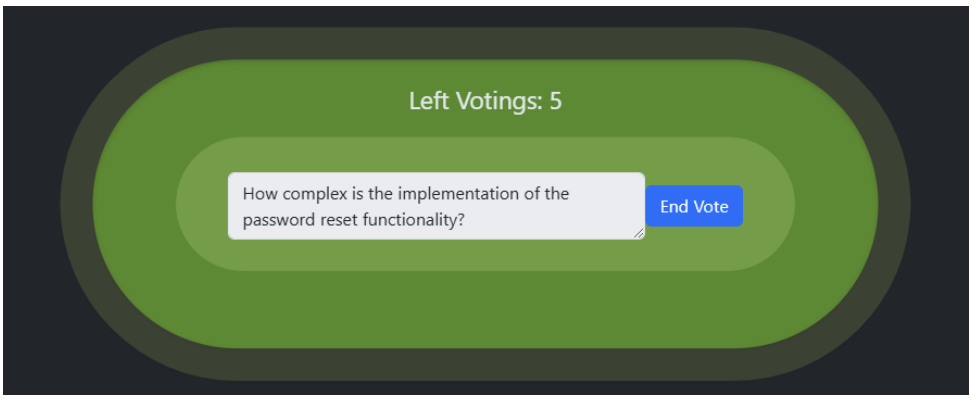


Abbildung 41: Spielfeld - Poker Tisch - Frage

Abbildung 42: Spielfeld - Poker Tisch - Ergebnis

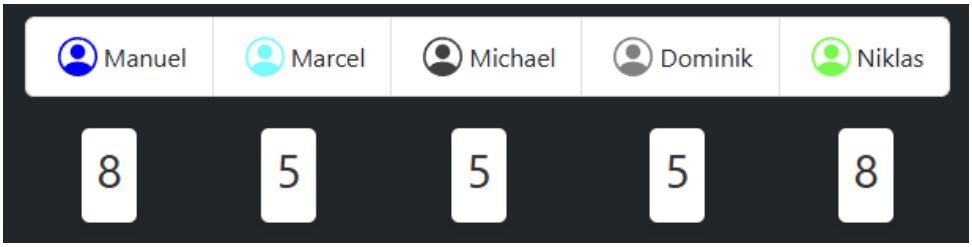


Abbildung 43: Spielfeld - Benutzerliste

(4) Spielerverwaltung

Der Scrum Master verfügt über die Funktion, Benutzer zu befördern oder zu kicken. Wenn man einen Benutzer befördert, wird man selbst zum Benutzer dekretiert und der ausgewählte Benutzer ist der neue Scrum Master. Wenn man einen Benutzer kicken möchte, wird er zur Startseite weitergeleitet und kann somit nicht mehr mitspielen. In der Abbildung 44 wird dies abgebildet.

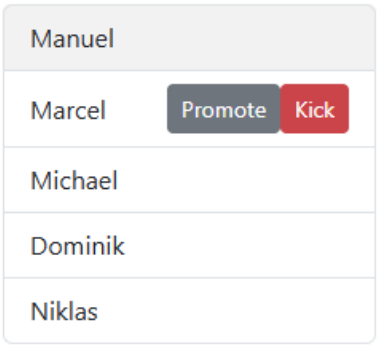


Abbildung 44: Spielfeld - Spielerverwaltung

(5) Benutzer Einstellungen

Der Benutzer selbst verfügt noch über die Möglichkeit, seinen Benutzernamen zu ändern. Dies wird in der Abbildung 45 abgebildet. Dieser hat die Voraussetzung, dass er mindestens vier Buchstaben lang sein muss. Ebenfalls hat der Benutzer die Möglichkeit, den Raum selbst zu verlassen. Er wird dann auf die Startseite weitergeleitet. Diese Funktion befindet sich in der Seitenleiste.

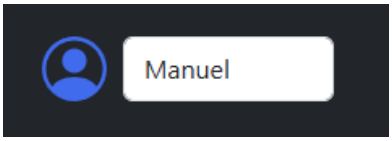


Abbildung 45: Spielfeld - Namen ändern

(6) Ergebnisse herunterladen

Jeder Benutzer wie auch der Scrum Master sind in der Lage, die Daten zu exportieren. Wenn man dies macht wird ein PDF mit einer Zusammenfassung der Fragen und Antworten heruntergeladen.

6.3 Java Spring Boot API

Die Java Spring Boot REST-API wurde erfolgreich umgesetzt und getestet. Die API ist in der Lage, die gesamten logischen Operationen, welche im Frontend angefordert werden, durchzuführen und benötigte Daten in der Datenbank zu persistieren. Es werden drei unterschiedliche Arten von Requests angeboten. So gibt es einen User Controller zum Behandeln aller benutzerbezogenen Requests. Der Question Controller behandelt die Logik einer Runde vom Stellen der Frage bis zum Beenden des Schätzvorgangs. Der Room Controller stellt den Endpunkt zum Abfragen aller Räume und zum Erstellen eines Raums bereit.

User Controller

In der Abbildung 46 sieht man alle verfügbaren Endpunkte, die die REST-API für die Benutzerverwaltung bereitstellt. Der Endpunkt '/user/promote' ist verantwortlich, den im Body übergebenen Benutzer im mitgegebenen Raum zum Scrum Master zu befördern. Der PUT-Endpunkt '/user/name' stellt den Endpunkt zum Ändern des Benutzernamens zur Verfügung. Der Endpunkt '/select/card' speichert die momentan gespeicherte Karte. Es wird auch ein Endpunkt zum Erstellen eines Benutzers bereitgestellt. Einmal gibt es einen Endpunkt, welcher alle Benutzer aus der Datenbank, unabhängig davon, in welchem Raum sie sich befinden, zurückgibt, und einmal gibt es den Endpunkt, welcher alle Benutzer aus einem Raum zurückgibt. Dieser Endpunkt wird ausgeführt, wenn ein neuer Benutzer beitritt und das Backend dem Frontend eine Notifikation sendet, dass ein neuer Benutzer dem Raum beigetreten ist oder den Raum verlassen hat. Sobald das Frontend diese Notifikation erhält, wird ein GET-Request an das Backend gesendet, um die aktuellen Benutzer anzeigen zu können. Der letzte Endpunkt im User-Controller bietet eine DELETE-Operation, um einen Benutzer aus einem Raum löschen zu können. Dieser wird entweder aufgerufen, wenn ein Benutzer das Browserfenster schließt oder auf den Raum-verlassen-Button im Frontend drückt.

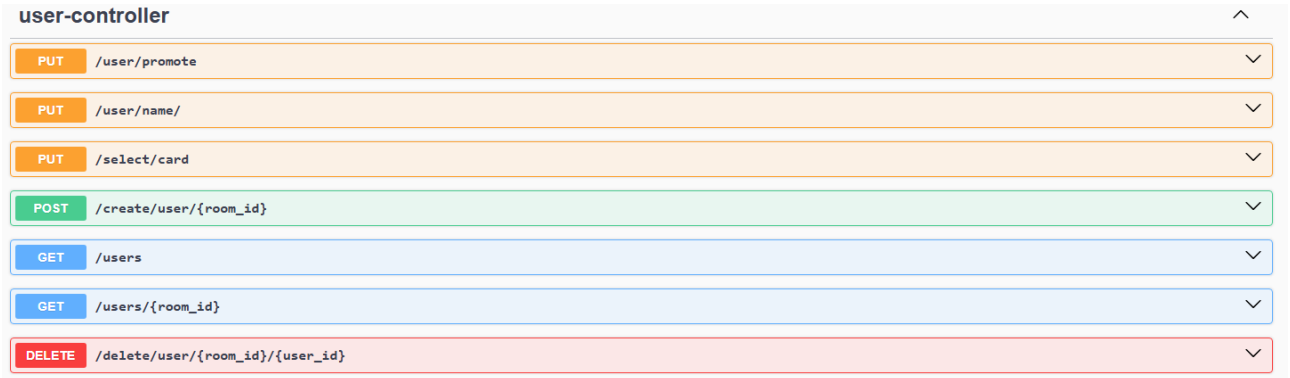


Abbildung 46: Schematische Abbildung User Controller

Question Controller

In der Abbildung 47 sieht man alle verfügbaren Endpunkte, die die REST-API für die Verwaltung von Runden in einem Raum bereitstellt. Dieser Controller ist hauptsächlich für die Spielmechanik in einer Runde verantwortlich. Ähnlich wie beim User Controller werden vom Frontend Requests meist nach einer Benachrichtigung vom Question-WebSocket abgeschickt. Die drei POST-Endpunkte werden genutzt, um eine Frage zu stellen, diese zu beenden, die Frage erneut zu stellen oder mit der nächsten Frage weiterzumachen. Wenn einer dieser Endpunkte aufgerufen wird, sendet auch der Question-WebSocket eine Benachrichtigung, sodass alle Clients die gestellte Frage sehen und darüber abstimmen können. Der Endpunkt '/status/room-id' gibt zurück, ob im Raum mit der übergebenen Raum-ID gerade eine Frage gestellt wird oder nicht und ist wichtig für den Fall, dass ein Benutzer während einer aktiven Runde dem Raum beitritt. Der letzte Endpunkt wird genutzt, um alle Fragen, die in einem Raum gestellt wurden, abzufragen.

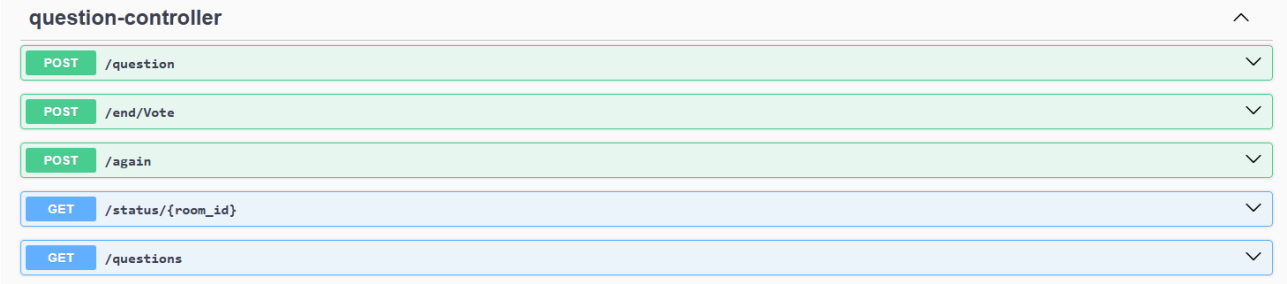


Abbildung 47: Schematische Abbildung Question Controller

Room Controller

In der Abbildung 48 sieht man die beiden verfügbaren Endpunkte, die die REST-API für die Verwaltung von Räumen bereitstellt. Es gibt zwei GET-Endpunkte, beim Aufruf von '/rooms' erhält man alle momentan vorhandenen Räume. Beim Aufruf von '/create/room' wird ein neuer Raum erstellt und man erhält die Raum-ID im Response.

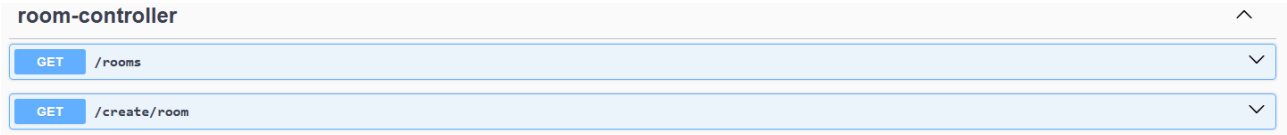


Abbildung 48: Schematische Abbildung Room Controller

WebSockets

Die WebSockets wurden erfolgreich implementiert. Diese bieten nun die Funktionalität der bidirektionalen Verbindung. Dies bedeutet, dass eine Anfrage nicht zwingend vom Client gestellt

werden muss, sondern es wird beim Betreten eines Benutzers eine Session erstellt. Durch diese Session besteht die Möglichkeit, dass der Client sowie der Server dauerhaft Nachrichten senden. So kann der Client vom Server benachrichtigt werden, wenn beispielsweise ein Benutzer neu beitritt, sein Name geändert wird oder er den Raum verlässt. Dies waren Funktionen des 'UserWebSocketHandler', der 'QuestionWebSocketHandler' stellt jegliche Echtzeit-Kommunikation zwischen Frontend und Backend bereit, die einzelne Fragen betrifft. Also, wenn eine Frage gestellt wurde, ein Benutzer eine Karte selektiert oder das Voting abgeschlossen wurde.

6.4 Python

Die Python-API wird verwendet zur Protokollierung einer ScrumPoker-Runde. Die Python-API erstellt ein PDF, in welchem man alle benötigten Informationen einer Runde sieht. Dies erspart die manuelle Protokollierung und spart so viel Aufwand.

1. Datum und Uhrzeit

   a) Das Datum ist bereits im Namen des Dokuments ersichtlich, wie in der nachstehenden Abbildung 49 ersichtlich.

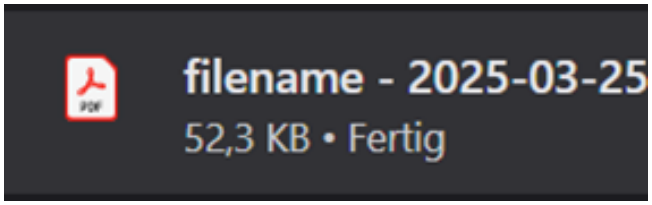


Abbildung 49: Name des Dokuments

   b) Weiters ist es ersichtlich in der Überschrift des Dokuments, auch zu sehend in der nachstehenden Abbildung 50.

Scrumpoker at 25-03-25

Abbildung 50: Überschrift des Dokuments

2. Darstellung aller Fragen

   a) Alle Informationen zu den Fragen befinden sich im Abschnitt 'Question Overview'. Wie man es in der Abbildung 51 sieht.

   b) Dort sieht man den Text der Frage, den durchschnittlichen Wert der jeweiligen Frage sowie das Datum und die Uhrzeit, wann die Frage gestellt wurde.

Question overview:

Start time	Question	Average Vote
25-03-25 14:01	Arbeitspaket 1	16.50
25-03-25 14:01	Arbeitspaket 1	13.00
25-03-25 14:01	Arbeitspaket 2	5.00
25-03-25 14:01	Arbeitspaket 3	10.50

Abbildung 51: Question-Overview

3. Teilnehmerübersicht

   a) Die Informationen zu den Teilnehmern befinden sich im Abschnitt 'User Information'. Wie man es in der Abbildung 52 sieht.

   b) Dort werden der Name, das Datum und die Uhrzeit, zu der der Benutzer den Raum betreten hat, die Zeit als er den Raum verlassen hat und ob er Scrum Master oder Teammitglied ist. Wenn der Benutzer den Raum noch nicht verlassen hat, steht statt dem Datum und der Uhrzeit 'Aktiv' im Feld der 'Leave Time'.

User information:

Name	Scrum Master	Join Time	Leave Time
Alice	Yes	25-03-25 13:34	Active
Bob	No	25-03-25 14:00	Active

Abbildung 52: User Information

4. Abstimmungen

   a) Die Informationen zu den Abstimmungen findet man im Abschnitt 'User Votes'. Wie man es in der Abbildung 53 sieht.

   b) Hier sind alle Abstimmungen zu jeder Frage von jedem Benutzer aufgelistet.

Vote details:

User	Question	Selected Card
Alice	Arbeitspaket 1	13
Alice	Arbeitspaket 1	13
Alice	Arbeitspaket 2	5
Alice	Arbeitspaket 3	13
Bob	Arbeitspaket 1	13
Bob	Arbeitspaket 1	20
Bob	Arbeitspaket 1	13
Bob	Arbeitspaket 2	5
Bob	Arbeitspaket 3	8

Abbildung 53: User Votes

Durch die strukturierte Darstellung des PDFs kann sie für Dokumentation und analytische Zwecke verwendet werden. Weiters kann man sie für die Kontrolle nutzen, um zu sehen, wie lange man für das Arbeitspaket einberechnet hat und wie lange es wirklich gedauert hat. Weiters ist es auch wichtig für Teammitglieder, denn sie können nachschauen, wie viel Zeit für ihr Arbeitspaket einberechnet wurde und können somit abschätzen, ob sie gut in der Zeit liegen oder nicht.

Praktische Einsatzbereiche

Einsatzbereiche:

- Dokumentation für Sprint-Reviews/Projektmeetings
- Für externe Nachweise für den Projektverlauf.

Der Download des PDFs erfolgt einfach über einen Button-Klick im Frontend auf den Button 'Export Data', der wie folgt aussieht:Abbildung 54. Dieser Button befindet sich im Menü auf der rechten Seite unterhalb des 'Copy Link'-Buttons.

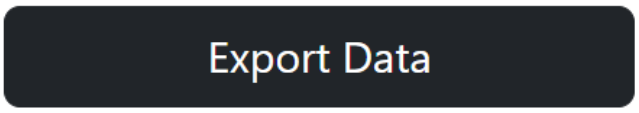


Abbildung 54: Export Data Button

6.5 Datenbank

Die objektrelationale Datenbank wurde erfolgreich implementiert, getestet und in die Gesamtarchitektur eingebettet. Eben diese Datenbank dient als zentrales Speichermodul für alle Räume, Benutzer und Fragen sowie deren Änderungen im Zeitverlauf.Die Daten werden in mehreren Tabellen gespeichert. Zusätzlich werden die Daten noch in Log-Tabellen gespeichert, die dann die Python-API für die Protokollierung verwenden.

6.5.1 Ablauf einer ScrumPoker-Sitzung aus Sicht der Datenbank

Um die Funktionsweise der Datenbank anschaulich zu erklären, wird der typische technische Ablauf einer ScrumPoker-Sitzung mithilfe von Bildern Schritt für Schritt dargestellt.

(1)Raum erstellen

Wenn ein neuer Raum erstellt wird, wird eine zufällig generierte 6-stellige Zeichenkette in die RoomTabelle eingefügt. Wie in Abbildung 55 ersichtlich.

(2)Raum erstellen (Log)

Wenn ein neuer Raum in die Room-Tabelle eingefügt wird, erstellt ein Trigger einen Eintrag in die RoomLogTabelle. In der Log-Tabelle wird nun der Zeitpunkt der Erstellung des Raums dokumentiert. Die Log-Tabelle zur Raumerstellung sieht aus, wie in Abbildung 55 abgebildet.

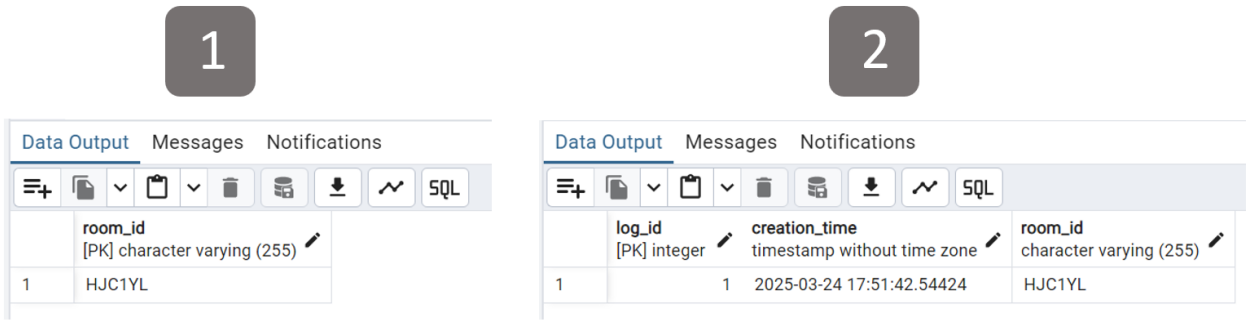


Abbildung 55: Ablauf 1-2

(3)Nutzer beitreten

Wenn nun ein Teammitglied den Raum betritt, wird dieser Benutzer in die User-Tabelle gespeichert. Dort werden Name, Farbe und auch die Rolle gespeichert. Die Rolle des beigetretenen Benutzers kann entweder Scrum Master oder Teammitglied sein. Diese User-Tabelle wird in Abbildung 56 gezeigt.

(4)Nutzer beitreten (Log)

Auch hier wird wieder ein Trigger ausgelöst, der die Daten in die UserLogTabelle speichert, mit einem Zeitstempel, der zeigt, zu welcher Zeit der User den Raum betreten hat, wie in Abbildung 56 ersichtlich.

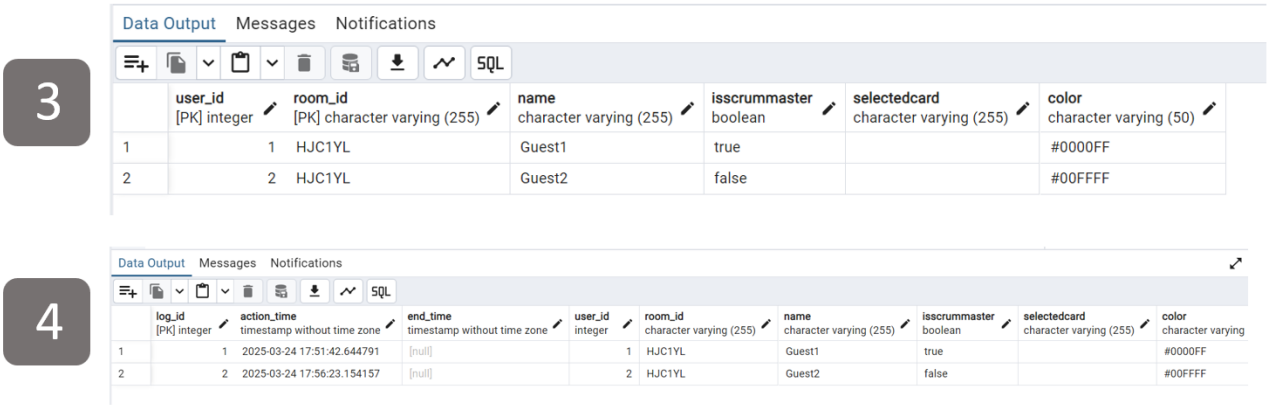


Abbildung 56: Ablauf 3-4

(5)Frage stellen

Wenn nun eine Frage gestellt wird, wird die Frage in die Question-Tabelle gespeichert. Wie sich aus der Abbildung 57 schließen lässt.

(6)Frage stellen (Log)

Gleichzeitig wird ein Log-Eintrag in der QuestionLog-Tabelle erstellt, der die Aktivierungszeit erfasst. Wie man es in Abbildung 57 sieht.

4

Data Output | Messages | Notifications

	question_id [PK] integer	round_id [PK] integer	room_id [PK] character varying (255)	text character varying (255)	average_vote numeric (10,2)	is_active boolean
1	1	1	HJC1YL	Arbeitspaket xy	-2.00	true

5

Data Output | Messages | Notifications

	log_id [PK] integer	creation_time timestamp without time zone	end_time timestamp without time zone	question_id integer	room_id character varying (255)	text character varying (255)	average_vote numeric (10,2)
1	1	2025-03-24 18:02:37.663657	[null]	1	HJC1YL	Arbeitspaket xy	-2.00

Abbildung 57: Ablauf 5-6

(7)Karten auswählen

Sobald ein Teilnehmer eine Karte auswählt, wird diese Auswahl in der Spalte selectedCard der User-Tabelle gespeichert, wie man in Abbildung 58 sieht.

(8)Karten auswählen (Log)

Über einen weiteren Trigger wird jede Änderung zusätzlich in der SelectedCardChangeLog-Tabelle dokumentiert, mit Zeitstempel. So kann auch später nachvollzogen werden, wann welche Karte von welchem User ausgewählt wurde. Wie man es in Abbildung 58 sieht.

7

Data Output | Messages | Notifications

	user_id [PK] integer	room_id [PK] character varying (255)	name character varying (255)	isscrummaster boolean	selectedcard character varying (255)	color character varying (50)
1	1	HJC1YL	Guest1	true	8	#0000FF
2	2	HJC1YL	Guest2	false	8	#00FFFF

8

Data Output | Messages | Notifications

	log_id [PK] integer	action_time timestamp without time zone	user_name character varying (255)	question_text character varying (255)	selected_card character varying (255)	room_id character varying (255)
1	1	2025-03-24 18:02:41.45142	Guest1	Arbeitspaket xy	8	HJC1YL
2	2	2025-03-24 18:02:43.368028	Guest2	Arbeitspaket xy	8	HJC1YL
3	3	2025-03-24 18:02:49.839131	Guest1	Arbeitspaket xy	0	HJC1YL
4	4	2025-03-24 18:02:51.767367	Guest2	Arbeitspaket xy	0	HJC1YL
5	5	2025-03-24 18:05:50.253525	Guest1	Arbeitspaket xy	8	HJC1YL
6	6	2025-03-24 18:05:51.599976	Guest2	Arbeitspaket xy	8	HJC1YL
7	7	2025-03-24 18:06:36.911671	Guest1	Arbeitspaket xy	5	HJC1YL
8	8	2025-03-24 18:06:38.345254	Guest2	Arbeitspaket xy	5	HJC1YL

Abbildung 58: Ablauf 7-8

(9)Abstimmung auswerten

Nach Abschluss der Abstimmung wird in der Question-Tabelle der Durchschnittswert unter average_vote gespeichert. Wie bei diesem Beispiel zu dem Arbeitspaket xy in Abbildung 59 ersichtlich.

(10)Abstimmung auswerten

Auch dieser Wert wird in der QuestionLog-Tabelle dokumentiert, zusammen mit dem Endzeitpunkt der Frage. So kann im Nachhinein exakt nachvollzogen werden, wie lange eine Frage aktiv war und wie abgestimmt wurde. Dies wird später in der Python-API verwendet, wie in Abbildung 59 zu sehen ist.

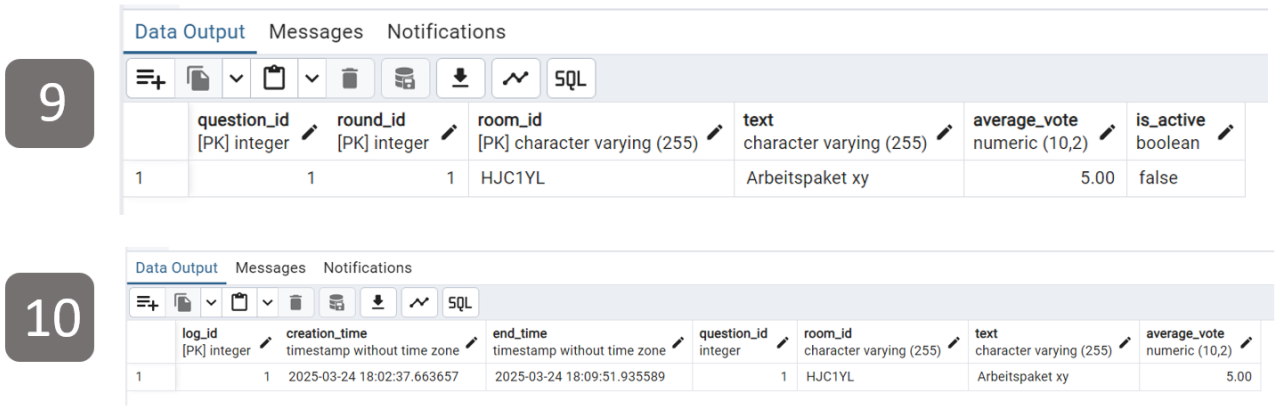


Abbildung 59: Ablauf 9-10

(11)Raum verlassen

Verlässt ein Benutzer den Raum, wird dieser aus der User-Tabelle gelöscht. Die Tabelle stellt sich nach dem Löschvorgang wie in Abbildung 60 dargestellt dar.

(12)Raum verlassen

Gleichzeitig wird in der zugehörigen User Log-Tabelle die end_time gesetzt, um den Austrittszeitpunkt zu speichern. Wie in der Abbildung 60 ersichtlich ist.

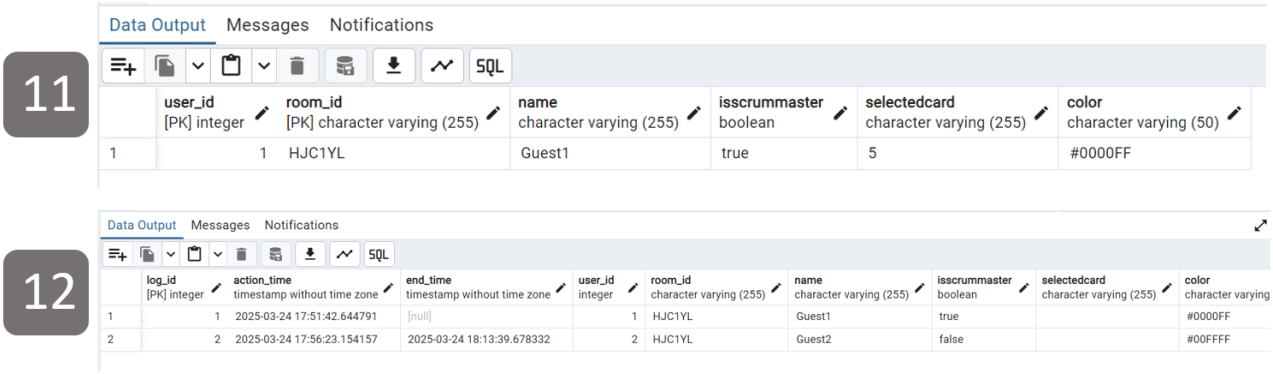


Abbildung 60: Ablauf 11-12

7 Resümee

Die Diplomarbeit ScrumPoker, entstand in der Kooperation mit der Porsche Informatik und wurde entwickelt, um den Prozess des Sprint Planning Meetings zu verbessern. Das Ziel dabei war es, eine Anwendung zu entwerfen, die es dem Unternehmen ermöglicht, einfache und dynamische Sitzungen abzuhalten. Zusätzlich bietet unsere Software die Möglichkeit, Sprints effizienter zu planen und deren Ergebnisse als Dokument zu exportieren.
Da unser Betreuer der Porsche Informatik in Salzburg tätig ist, hatten wir zu Beginn Schwierigkeiten, die Kommunikation am Laufen zu halten. Dieses Problem konnten wir durch die Verwendung von Microsoft Teams und Festlegung fester Termine lösen. Durch diese regelmäßigen Meetings konnten wir Anforderungen immer klar erkennen und entsprechend umsetzen. Außerdem konnten wir durch laufende Rückmeldungen und entsprechende Dokumentation das Projekt erfolgreich abwickeln.
Im Laufe der Umsetzung konnten wir bekannte Technologien in einem praktischen Umfeld festigen und neue Technologien kennenlernen. Dadurch bekamen wir einen praxisnahen Einblick in die Umsetzung eines Softwareprojekts. In folgenden Themenbereichen konnten wir unser Wissen erweitern.

- Client-Server Kommunikation
- PostgreSQL
- Angular
- Ionic
- Springboot

Zusammenfassend kann man sagen, dass die Diplomarbeit erfolgreich umgesetzt werden konnte. Unser Auftraggeber ist mit dem Ergebnis zufrieden und arbeitet gerade an der Integration in ihre Cloud-Infrastruktur, sodass sie von allen Projektteams eingesetzt werden kann. Auch wir als Projektteam konnten unsere Ansprüche erfüllen und sind stolz auf unsere Umsetzung.

8 Aufgabenverteilung

8.1 Marcel Hochgatterer

In der Entwicklung bestand die Aufgabe darin, die Handy-Applikation zu entwickeln. Hierbei handelt es sich um die Benutzeroberfläche für eine hybride App-Lösung. Ebenso wurde Capacitor verwendet, um die nativen Funktionen des Geräts zu integrieren, wie in Kapitel [Capacitor] beschrieben. Zusätzlich war Marcel Hochgatterer für die Protokollierung der Meetings zuständig. Dies umfasst die Kapitel und deren Unterkapitel:

- 4.2 Mobile-Anwendung
- 6.1 Ionic Mobileapplikation

Ebenso wurden folgende Kapitel verfasst:

- 1.3 Projektinhalt-Überblick
- 3 Planung und Realisierung

8.2 Manuel Mayrhofer

Die Aufgabe war es, die Webanwendung zu entwickeln. Hierbei lag der Hauptfokus auf Designentwicklung und Benutzerfreundlichkeit. Somit war es auch seine Aufgabe, das Logo zu entwerfen. Aus diesem Grund führte er die Interviews in den Meetings, um Entscheidungen zu Design und Benutzerfreundlichkeit treffen zu können.
Seine Kapitel sind:

- 4.3 Webapplikation
- 6.2 Angular Webapplikation
- 2 Technische und fachpraktische Grundlagen und Methoden

8.3 Niklas Schnabel

Die technische Aufgabe bestand darin, die REST-API zu entwickeln und somit alle Funktionalitäten zu implementieren. Zusätzlich dazu kümmerte er sich um die WebSocket-Kommunikation, um die Echtzeit-Funktionalität der Anwendung zu ermöglichen. Ebenso war es Niklas Schnabels Aufgabe, das Projekt mittels Docker-Containern an den Auftraggeber zu übermitteln.
Folgende Kapitel wurden verfasst:

- 4.4 REST-API
- 6.3 Java Spring Boot API
- 5 Deployment

Zudem umfasst es die weiteren Kapitel:

- Zusammenfassung
- Abstract
- 1.1 Motivation
- 1.2 Zielsetzung

8.4 Michael Klaus

Die Aufgabe war es, die Datenbank zu entwickeln und aufzusetzen. Dabei war die Aufgabe, das Schema zu entwerfen und mit Funktionalitäten in Form von Triggern zu erweitern. Zudem wurde die Python-API umgesetzt, welche die Protokollierung und Auswertung der Anwendung übernimmt. Dies umfasst folgende Kapitel:

- 4.1 Technischer Überblick
- 4.5 Phyton-API
- 4.6 Datenbank
- 6.4 Phyton
- 6.5 Datenbank

Weiters wurden folgende Kapitel verfasst:

- 1.4 Projektumfeld
- 2.5 Verwendete Bibliotheken und Plug-Ins

Glossar

ACID Wichtigste Eigenschaften einer Datenbank: Atomität, Konsistenz, Isolation und Langlebigkeit

ASCII ASCII ist ein 7-Bit Code und somit sind 128 Zeichen definiert

CRUD Operationen, Create, Read, Update, Delete

GUI Graphical User Interface

JSON Java Script Object Notation

nginx Webserver Software, Reverse Proxy

SASS Stylesheetsprache

SQL Structured Query Language

TCP Transmission Control Protocol/ Internet Protocol

UI User Interface

Literaturverzeichnis

[1] P. Consultant, „Scrum Poker," 2025, letzter Zugriff am 17.03.2025. Online verfügbar: https://www.pureconsultant.de/de/scrum/scrum-poker/

[2] A. Diehl, „Scrum," letzter Zugriff am 25.03.2025. Online verfügbar: https://digitaleneuordnung.de/blog/scrum-methode#product-backlog

[3] Angular, „Angular," letzter Zugriff am 26.03.2025. Online verfügbar: https://v17.angular.io/guide/what-is-angular

[4] Ionic, „Ionic," letzter Zugriff am 26.03.2025. Online verfügbar: https://ionicframework.com/

[5] Angular, „Angular," letzter Zugriff am 26.03.2025. Online verfügbar: https://docs.python.org/3/

[6] Java, „Java," letzter Zugriff am 26.03.2025. Online verfügbar: https://docs.oracle.com/en/java/

[7] PostgreSQL, „PostgreSQL," letzter Zugriff am 26.03.2025. Online verfügbar: https://www.postgresql.org/

[8] Docker, „Docker," letzter Zugriff am 26.03.2025. Online verfügbar: https://www.docker.com/

[9] Gitlab, „Gitlab," letzter Zugriff am 26.03.2025. Online verfügbar: https://docs.gitlab.com/

[10] Clockify, „Clockify," letzter Zugriff am 26.03.2025. Online verfügbar: https://clockify.me/google-docs-time-tracking

[11] IntelliJ, „IntelliJ," letzter Zugriff am 26.03.2025. Online verfügbar: https://www.jetbrains.com/de-de/idea/

[12] webstorm, „webstorm," letzter Zugriff am 26.03.2025. Online verfügbar: https://www.jetbrains.com/de-de/webstorm/

[13] microsoft, „VSCode," letzter Zugriff am 26.03.2025. Online verfügbar: https://code.visualstudio.com/

[14] Docker, „Docker," letzter Zugriff am 26.03.2025. Online verfügbar: https://docs.docker.com/desktop/

[15] E. E. Dave Gandy, Jason Otero, „Font Awesome," letzter Zugriff am 24.03.2025. Online verfügbar: https://github.com/FortAwesome/Font-Awesome

[16] B. Troncone, „RxJS," letzter Zugriff am 24.03.2025. Online verfügbar: https://www.learnrxjs.io/

[17] SimplyTest, „Was ist Jasmine?" letzter Zugriff am 24.03.2025. Online verfügbar: https://www.testautomatisierung.org/lexikon/jasmine/

[18] Angular, „@angular/platform-browser," letzter Zugriff am 24.03.2025. Online verfügbar: https://v17.angular.io/api/platform-browser

[19] ——, „FormsModule," letzter Zugriff am 24.03.2025. Online verfügbar: https://v17.angular.io/api/forms/FormsModule

[20] J. Hoeller, „Spring Framework," letzter Zugriff am 24.03.2025. Online verfügbar: https://spring.io/projects/spring-framework

[21] Wikipedia, „Java Database Connectivity," letzter Zugriff am 24.03.2025. Online verfügbar: https://de.wikipedia.org/wiki/Java_Database_Connectivity#:~:text=Java%20Database%20Connectivity%20(JDBC%2C%20englisch,auf%20relationale%20Datenbanken%20ausgerichtet%20ist.

[22] D. Varrazzo, „psycopg2 2.9.10," letzter Zugriff am 24.03.2025. Online verfügbar: https://pypi.org/user/piro/

[23] ReportLab, „ReportLab," letzter Zugriff am 24.03.2025. Online verfügbar: https://wiki.ubuntuusers.de/ReportLab/

[24] Matthias Scharwies und Rolf Borchmann, „JavaScript/WebSocket," 2025, letzter Zugriff am 21.01.2025. Online verfügbar: https://wiki.selfhtml.org/wiki/JavaScript/WebSocket#:~:text=Die%20WebSocket%20API%20ermÃűglicht%20es,die%20Antwort%20abfragen%20zu%20mÃijssen.

[25] Angular Team. (2023), „Using observables for streams of values," 2023, letzter Zugriff am 21.01.2025. Online verfügbar: https://v17.angular.io/guide/observables

[26] D. Maximini, „Relatives Schätzen (z.B. in Story Points)," 2025, letzter Zugriff am 25.01.2025. Online verfügbar: https://www.valuerise-consulting.de/kategorie/relatives-schaetzen-z-b-in-story-points/

[27] D.-I. S. Wilde, „Capacitor," 2022, letzter Zugriff am 15.02.2025. Online verfügbar: https://www.wilde-it.com/capacitor/#:~:text=Ionic%20Capacitor%20wird%20mit%20leistungsstarken,groÃ§artigen%20Alternative%20zu%20Cordova%20macht.

[28] C. Pust, „Was ist Base64 und was bringt es?" 2023, letzter Zugriff am 22.02.2025. Online verfügbar: https://fastwp.de/was-ist-base64-und-was-bringt-es/#was-ist-base64

[29] developer mozilla, „Websocket," letzter Zugriff am 25.03.2025. Online verfügbar: https://developer.mozilla.org/de/docs/Web/API/WebSocket

[30] GeeksforGeeks, „Spring Boot – Application Properties," letzter Zugriff am 26.03.2025. Online verfügbar: https://www.geeksforgeeks.org/spring-boot-application-properties/

[31] Tembo, „Connecting to Postgres with Java using JDBC," letzter Zugriff am 25.01.2025. Online verfügbar: https://tembo.io/docs/getting-started/postgres_guides/connecting-to-postgres-with-java#2-using-maven

[32] Oracle, „Using Prepared Statements," 2024, letzter Zugriff am 25.01.2025. Online verfügbar: https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html

[33] RyteWiki, „Websocket," letzter Zugriff am 24.03.2025. Online verfügbar: https://de.ryte.com/wiki/Websocket/

[34] S. Gangwar, „Spring Boot WebSockets: A Guide to Seamless Real-Time Interactions," 2025, letzter Zugriff am 27.01.2025. Online verfügbar: https://www.mindbowser.com/springboot-websockets-real-time-guide/

[35] R. Stoyanchev, „Interface WebSocketSession," letzter Zugriff am 18.03.2025. Online verfügbar: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/socket/WebSocketSession.html

[36] VMware, „Interface WebSocketHandler," letzter Zugriff am 26.01.2025. Online verfügbar: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/socket/WebSocketHandler.html

[37] I. Redaktion, „Compose: Docker-Container-Anwendungen orchestrieren," letzter Zugriff am 27.03.2025. Online verfügbar: https://www.ionos.at/digitalguide/server/konfiguration/docker-compose-tutorial/

[38] Spring, „Spring Boot with Docker," letzter Zugriff am 27.03.2025. Online verfügbar: https://docs.docker.com/desktop/

[39] D. Chhaniyara, „Docker + Angular + Nginx," letzter Zugriff am 27.03.2025. Online verfügbar: https://dev.to/oneofthedevs/docker-angular-nginx-37e4

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang

Abbildung 61: Projekt Logo

Abbildung 62: Projekt Logo

PORSCHE INFORMATIK

Schnabel Niklas

Backend Entwickler mit Java und Springboot

Mayrhofer Manuel

Frontend Entwickler mit Angular Framework

Dieses Tool ist weit mehr als ein einfaches Schätzspiel – es verbindet das bewährte Scrum-Poker-Prinzip mit exklusiven Anpassungen und optimierten Funktionen, die gezielt auf die Bedürfnisse von Porsche Informatik zugeschnitten sind.

Scrum Poker

Mit Scrum Poker wird das Planen im Team einfacher, genauer und transparenter. Unsere benutzerfreundliche, intuitive Oberfläche sorgt dafür, dass jeder sofort loslegen kann – ohne lange Einarbeitungszeit. Dank zusätzlicher Analyse-Features behält man außerdem den Überblick über vergangene Schätzungen und trifft fundierte Entscheidungen für die kommenden Sprints.

Klaus Michael

Datenbank Manager mit Postgres

Hochgatterer Marcel

Handy App Entwickler mit Ionic Framework

htlperg

Abbildung 63: Plakat

Höhere Technische Bundeslehranstalt Perg
Machlandstraße 48, 4320 Perg
E-Mail office@htl-perg.ac.at | Web www.htl-perg.ac.at
Tel 07262 53926 | Fax 07262 53926-6

Kooperationsangebot an die HTL-Perg

☒ Höhere Abteilung für Informatik ☐ Fachschule für Informationstechnik

Unternehmen:	Porsche Informatik GmbH
Straße:	Louise-Piëch-Straße 9
PLZ, Ort, Staat:	5020, Salzburg, Österreich
Ansprechperson:	Markus Watzl
Position:	Teamleitung
Telefon:	+43 66246706697
FAX:	
E-Mail:	markus.watzl@porscheinformatik.com
Web-Adresse:	https://www.porscheinformatik.com/

Gewünschte Kooperation (Mehrfachnennungen möglich):

☐ Ferialpraktikum
☐ Schulprojekt
☒ Diplomarbeit
☐ Berufspraktikum (Fachschule)
☐ Abschlussarbeit (Fachschule)

Betroffene SchülerInnen (falls bekannt) Klaus Michael, Hochgatterer Marcel, Mayrhofer Manuel, Schnabel Niklas

Titel/Thema der Kooperationsidee:

Konzipierung und Umsetzung eines individuellen Schätzpoker-Systems für ein verteiltes Team

Kurze Beschreibung der Kooperationsidee:

Als Porsche Informatik ist es uns wichtig für die Planung und Schätzung unserer Sprints ein auf uns zugeschnittenes Tool nutzen zu können.

Beispiel: https://zilverline.poker/ (bietet nicht den benötigten Funktionsanfang, den wir brauchen, soll aber ein Gefühl vermitteln, was wir uns vorstellen)
- Aus der agilen Entwicklung ein Tool für Schätzpoker mit Fibonacci-Zahlen
- Webanwendung mit Datenbank dahinter
- Möglichkeit, dass ein großes, verteiltes Team teilnimmt (bis zu 20 Leute)
- Auswertungen erzeugen, in dem im Nachhinein nachgesehen werden kann, was geschätzt wurde
- Anmeldung ganz ohne Email-Adresse o.ä. (betrifft nicht zilverline, aber diverse Konkurrenzprodukte)
- Außerdem müssen Emojis und unterschiedliche Schriften (z.B. kyrillisch) erlaubt sein
- Weitere Anforderungen müssen im Zuge von Interviews identifiziert werden

Frühest möglicher Beginn:	01.01.2024
Spätest mögliches Ende:	31.03.2025

17.01.2024
Datum

Unterschrift

Bitte senden Sie dieses Formular an die HTL-Perg, Machlandstraße 48, A-4320 Perg, oder via E-Mail an office@htl-perg.ac.at.

	HTBLA Perg Höhere Lehranstalt für Informatik	Diplomprüfung

Abnahmeprotokoll HTBLA Perg

Projektname : *ScrumPoker*	**Projektnummer :** -
Auftraggeber : *Porsche Informatik*	**Empfänger :** *Markus Watzl*

Das Werk „ScrumPoker" Version 1.0

basierend auf dem Vertrag vom: 17.01.2024

wurde am 26.02.2025 zur Abnahme angemeldet.

Zu dem Projekt gehören folgende **Komponenten**:

- ☑ Dieses Protokoll
- ☑ Anleitung zum Starten inkl. Script
- ☑ Java Backend
- ☑ Postgres Datenbank
- ☑ Phyton Backend
- ☑ Angular Frontend
- ☑ Docker Compose Dateien zum Starten der Anwendung

Das Werk „ScrumPoker" Version 1.0

basierend auf dem Vertrag vom: 17.01.2024

wurde am 06.03.2025 abgenommen und zur Umsetzung freigegeben.

Mit seiner Unterschrift bestätigt der Auftraggeber die Abnahme der Diplomarbeit.

Datum: 06.03.2025

Unterschrift AG

Unterschrift AN

Meeting 07/02/2024

Teilnehmer:	
HTL-Perg:	Marcel Hochgatterer, Niklas Schnabel, Manuel Mayrhofer, Michael Klaus
Porsche Informatik:	Markus Watzl, Rainer Schmidt

Vorüberlegte Fragen unserer Seite:

1. Kurze Einführung
2. Ziele und Anforderungen
3. Gibt es bereits bestehende? https://zilverline.poker (wie viel sollte es diesem ähneln) hat man schon einmal versucht so eine Schätzsystemplattform zu entwickeln?
4. Welche Funktionen sind besonders wichtig? Welche optional noch möglich?
5. Welche Technologien oder Plattformen sollen verwendet werden?
6. Wie soll die Kommunikation ablaufen?
7. Wie soll das Vorgehensmodell laufen? Wöchentliche/ monatliche Meetings mit Vorstellung?
8. Praktikum – wird abgesprochen?
9. Sonstiges

Antworten die im Meeting aufgekommen sind:

Website, Angular, Java, Datenbank – (flexible, open source, kein mssql, oracle)

Funktionen:

- Rechte – Autorisierung/Authentifizierung- login
- Fibonacci
- keine Mengen Begrenzung
- Admin
    - erstell Menü
    - Karten aufdecken (ohne das alle geschätzt haben, jemanden entfernen, …)
    - Einsicht auf die abgeschätzten Werten (Protokoll), mit Zeitstempel
        - Man zeichnet jede Runde auf unterscheidet jedoch zwischen Diskussionen und fertiges Ergebnis
        - Nice to have: Fragen sollen mitgeschrieben werden, um Werte zuzuordnen
- Ein Meeting enthält mehrere Stories

Einfacher einstieg – generell einfach

Mit URL beitreten und Namen erst im Raum einstellen und sonstige Features -> so schnell wie möglich verwendbar sein

Design

Dark/light mode, eventuell Vorschlag ausarbeiten