



# Coronattack

Infiziere die Menschen des Gegners und heile sie danach wieder!

**Autoren: Theologos Baxevanos und Chantale Gihara**

BSC\_INF\_WEB\_E

Dozent: Heinrich Zimmermann

# Inhaltsverzeichnis

1	Ausgangslage .....	3
1.1	Projektauftrag .....	3
1.2	Projektidee .....	3
1.3	Spielregeln .....	3
1.4	Ziele.....	4
1.5	Abgrenzung.....	4
1.5.1	Zeitliche Abgrenzung: .....	4
1.5.2	Sachliche Abgrenzung: .....	4
1.5.3	Soziale Abgrenzung: .....	4
1.5.4	Stakeholder .....	4
1.5.5	Fachlicher Kontext.....	5
1.5.6	Technischer Kontext .....	5
1.6	Information und Kommunikation .....	6
1.7	Iterationsmodell .....	6
1.8	Projektstrukturplan .....	6
1.9	Arbeitsplan / Balkendiagramm .....	8
1.9.1	Meilenstein 1 / Iteration 1 .....	8
1.9.2	Meilenstein 2 / Iteration 2 .....	8
1.9.3	Meilenstein 3 / Iteration 3 .....	9
1.9.4	Meilenstein 4 / Iteration 4 .....	9
1.10	Qualitätsszenarien .....	9
1.11	Randbedingungen .....	10
1.11.1	Technisch .....	11
1.11.2	Organisatorisch .....	11
1.11.3	Konventionen .....	11
1.12	Risiken & technische Schulden.....	11
2	Softwareengineering .....	12
2.1	Anforderungen.....	12
2.1.1	Funktionale Anforderungen .....	12
2.1.2	Qualitätsanforderung (Nicht funktionale Anforderungen) .....	15
2.2	Use Cases.....	16
2.2.1	Use Case 1: Registrierung .....	17
2.2.2	Use Case 2: Login.....	17
2.2.3	Use Case 3: Create Game .....	17
2.2.4	Use Case 4: Schwierigkeit setzen.....	18
2.2.5	Use Case 5: Join Game .....	18
2.2.6	Use Case 6: Spiel 1 .....	18
2.2.7	Use Case 7: Spiel 2 .....	19
2.2.8	Use Case 8: High Score anzeigen .....	19
2.3	Mockups .....	19
2.3.1	Startseite .....	20
2.3.2	About Seite .....	20
2.3.3	Registrierung .....	21
2.3.4	Spiel erzeugen / beitreten .....	21

2.3.5	Spielraum Beitreten.....	22
2.3.6	Spielanfrage .....	22
2.3.7	Spiel erzeugen .....	23
2.3.8	Der Spielraum .....	23
2.4	Protokolle .....	24
2.4.1	Protokoll Client-Server .....	24
2.4.2	Netzwerkprotokolle.....	25
2.5	Server, Middleware, Datenbank und Chat Funktion .....	31
2.5.1	Server .....	31
2.5.2	Middleware .....	32
2.5.3	Datenbank .....	32
2.5.4	Chat Funktion .....	32
2.5.5	Docker .....	32
2.6	Architektur .....	33
2.6.1	Architektur und Coderichtlinien .....	33
2.6.2	Coderichtlinien.....	35
2.7	Bausteinsicht.....	36
2.7.1	Struktur .....	36
2.8	Canvas API .....	36
3	Annexes .....	37
3.1	Protokolle .....	37
3.1.1	Protokoll vom 02.09.2021 .....	37
3.1.2	Protokoll vom 21.08.2021 .....	38
3.1.3	Protokoll vom 04.09.2021 .....	39
3.1.4	Protokoll vom 27.09.2021 .....	39
3.1.5	Protokoll vom 30.09.2021 .....	39
3.1.6	1.1.1 Protokoll vom 15.10.2021 .....	40
3.1.7	1.1.2 Protokoll vom 29.10.2021 .....	40
3.1.8	Protokoll vom 08.10.2021 .....	40
3.1.9	Protokoll vom 15.10.2021 .....	41
3.1.10	Protokoll vom 29.10.2021 .....	41
	Tabellen.....	42
	Abbildungen .....	42
	Quellen .....	43

# 1 Ausgangslage

## 1.1 Projektauftrag

Im Modul WebE haben wir den Auftrag erhalten ein Game mit folgenden Vorgaben zu entwickeln und dabei eine Woche vor jeder PVA den jeweiligen Meilenstein abzugeben.

Entwicklung eins Spiels mittels Web-Technologien vom folgenden Typ:

1. Runden-basiert oder Educational, oder Datensammler
2. Das Spiel muss eine Client/Server Architektur haben
3. Der Server und die Clients kommunizieren über ein Text-basiertes Protokoll. Das Protokoll muss lesbar sein.
4. Die Server-Funktionalität ist wie folgt definiert:
  - a. Er verwaltet den Spielverlauf (überprüft und stellt sicher, dass alle Spielzüge regelkonform sind, erkennt das Ende des Spiels, zählt Punkte, etc.)
  - b. Wenn alle Spieler das Spiel verlassen, dann beendet der Server das Spiel.
5. Ein Client hat folgende Eigenschaften:
  - a. Er nimmt Benutzereingaben durch eine grafische Schnittstelle (graphical User Interface, GUI) entgegen
  - b. Er gleicht den lokalen Status eines Spiels mit dem Status des Servers ab (Synchronisation)
  - c. Er erlaubt den Spielern eines Spiels zu chatten.
6. Folgende Aspekte sollen beachtet werden: Internationalisierung, Usability, Accessibility, Levels (das Spiel muss mind. 3 Levels haben), Responsiveness  
Am Ende des Projekts muss eine komplette Distribution des Spiels abgegeben werden (lauffähiges Spiel inklusive Quellcode, Installationsanleitung, Handbuch)

## 1.2 Projektidee

Wir haben uns für das Spiel «Coronattack» entschieden, dieses basiert auf einem Ping-Pong-Spiel. Es ist ein Dualplayer - Spiel und kann auf verschiedenen Schwierigkeitsgraden (unterschiedlich schneller Ball) gespielt werden.

Dabei sollte es zwei Modi geben:

1. mit dem Virus (Ball) müssen die Menschen des Gegenspielers getroffen werden, wer alle Menschen getroffen hat gewinnt.
2. mit der Impfung (Ball) müssen die Menschen wieder genesen werden, wer zuerst alle Menschen wieder genesen hat, hat gewonnen.

Hier eine kleine Skizze wie das Spiel ungefähr aussehen soll:

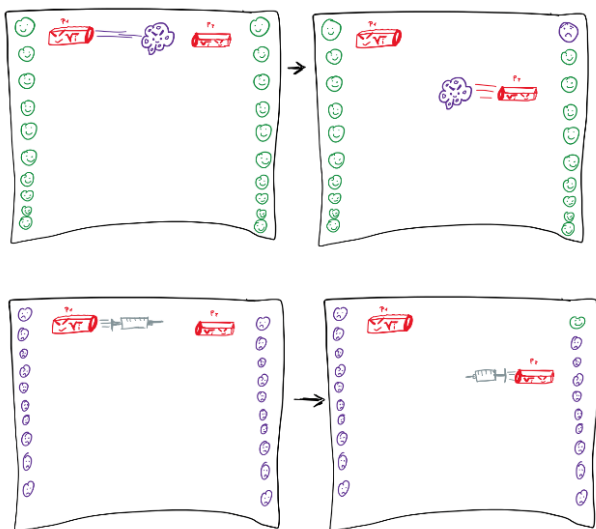


Abbildung 1: Skizze «Coronattack» Spiel

## 1.3 Spielregeln

Jeder Spieler muss sich zuerst registrieren und erhält dabei eine ID (diese ist nicht sichtbar), die UserId für den Benutzer ist die E-Mail-Adresse.

Beim Spielstart muss er den Namen und das Level des Spiels eintragen.

Der Spieler kann entweder ein Spiel eröffnen oder einem Spiel beitreten. Falls er ein Spiel eröffnet hat, muss er darauf warten, dass ein Spieler dem Spiel beitrifft, dabei kann er den Spieler zulassen oder zurückweisen. Wenn der Spieler einem Spiel beitrifft, muss er auf die Zulassung des Spielers, welcher das Spiel eröffnet hat, warten:

Spiel 1:

- Zuerst wird das Spiel «infiziere» aufgeschaltet.
- Dabei müssen die Spielgegner ihre Menschen vor dem Virus (Ball) mit einer fahrenden Maske schützen.
- Konnte der Virus (Ball) den Gegner nicht aufhalten, wird der Menschenkopf infiziert.
- Sind alle Menschenköpfe des Gegenspielers oder des Spielers infiziert ist das Spiel beendet
- Dem Verlierer werden seine erhaltenen Punkte zugeteilt
- Dem Gewinner werde die Punkte im Gesamtscore zugeteilt

Spiel 2:

- Ist Spiel 1 beendet geht es in Runde 2
- Die Spieler müssen jetzt sicherstellen, dass die Menschen krank bleiben.
- Der Spieler, welcher beim Gegenspieler zuerst alle Menschen geheilt hat, hat gewonnen
- Dem Gewinner werden die Punkte im Gesamtscore zugeteilt.

Falls es unentschieden steht, geht das Spiel weiter, bis jemand gewinnt. Am Ende wird eine Rangliste mit den Gesamtpunktzahlen der Spieler sichtbar.

## 1.4 Ziele

- Alle Meilensteine werden komplett und zeitgerecht abgegeben
- Das Spiel ist vor der letzten Präsenzveranstaltung umgesetzt und spielbereit
- Alle Voraussetzungen gemäss Projektauftrag 1.1 werden umgesetzt
- Spieler können sich registrieren und erhalten eine ID (diese ist nicht sichtbar)
- Spieler können sich den Gegner in der Liste aussuchen.
- Spieler kann sich sein Schwierigkeitslevel (Geschwindigkeit) selbst aussuchen

## 1.5 Abgrenzung

Anhand folgender zeitlicher, sachlicher und sozialer Abgrenzungspunkte wird das Projekt definiert:

### 1.5.1 Zeitliche Abgrenzung:

Das Projekt wurde vom Dozenten, Herr Dr. Heinrich Zimmermann freigegeben und endet mit der Abgabe der Game- Software am 26.11.2021.

### 1.5.2 Sachliche Abgrenzung:

Das Projekt hat zum Ziel, ein Spiel gemäss Auftrag 1.1 zu realisieren, welche es ermöglicht, Spielern sich zu registrieren, einzuloggen, zu Spielen und den High-Score einzusehen.

### 1.5.3 Soziale Abgrenzung:

Zum Projektteam gehören Chantale Gihara und Theologos Baxevanos.

Nicht zum Projekt gehören:

- Die Wartung des Spieles nach der Abgabe.
- Die Sicherheit und Verschlüsselung der Kundendaten. Die Kunden sind selbst verantwortlich für die Sicherheit ihrer Daten, sowie für eine Firewall- und Port-Konfiguration.

### 1.5.4 Stakeholder

Stakeholder	Interesse
Benutzer	Online spielen
IT-Personal (Studenten)	Sicherstellen, dass der IT-Teil der Applikation läuft
Fern Fachhochschule Schweiz (Auftraggeber)	Semesterarbeit, funktionelles Spiel

### 1.5.5 Fachlicher Kontext

Element	Bedeutung
Benutzer	Ein Benutzer der «coronattack» spielen möchte
Chat	Chat Funktion, die via «coronattack möglich ist»
Spiel	Spiel zwischen Spieler und KI oder Spieler und Spieler

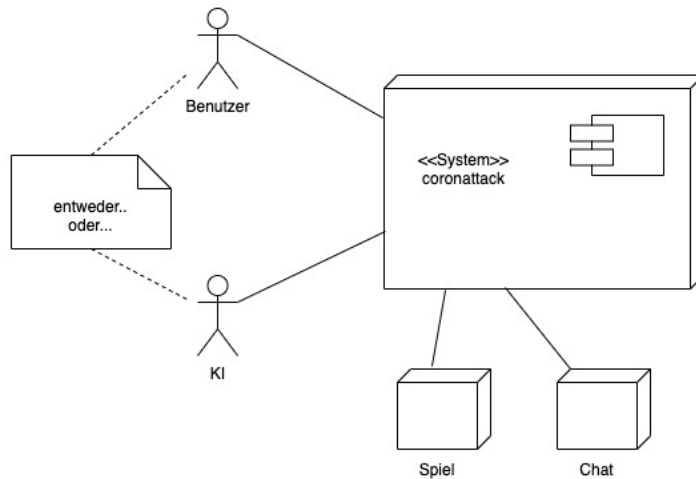


Abbildung 2: Skizze «Coronattack» Spiel

### 1.5.6 Technischer Kontext

#### Frontend Client

Die "Anbindung" menschlichen Benutzern erfolgt über ein grafisches Frontend, dessen Entwicklung ein Teil von «coronattack» ist. Für die Kommunikation zwischen des Front- und Backends wird das Websocketsprotokoll (via Socket.io) verwendet.

#### PostgreSQL

Die Daten werden in der PostgreSQL gespeichert und der Klient kommuniziert über das API der «pg» Library mit der Datenbank.

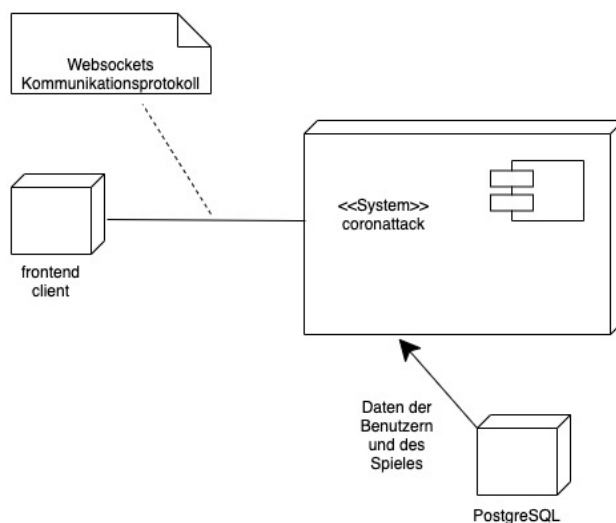


Abbildung 3: Skizze «Coronattack» Spiel

## 1.6 Information und Kommunikation

Die Kommunikation im Team erfolgt nach Absprache und übers Trello Board:

<https://trello.com/b/7HNVSgbQ/kanban-webe-coronattack> oder wenn möglich vor Ort, 2 wöchentlich Starbucks Zürich Oerlikon Bahnhof.

Während den Meetings wird von einem der Mitglieder ein Protokoll geführt, welches gleich hier im Annex 3.1 nachgeführt wird.

Dazu haben wir auch eine WhatsApp, in der wir auch ausserhalb des zwei wöchentlichen Meetings miteinander kommunizieren.

Die geplanten Stunden und Wochen pro Arbeitspaket werden in der Board Card im Trello eingeplant. Nachdem das Arbeitspaket abgeschlossen wurde, werden die effektiv verbrauchten Stunden und Wochen eingetragen. So kann der Aufwand gut überwacht werden und falls nötig angepasst werden.

Informationen zum Projekt werden an nachfolgenden Stellen gepflegt bzw. sind dort verfügbar:

- Projektdokument «Coronattack»: Wird in einem Word-Dokument geführt und wird auf One-Drive abgelegt, welches für beide von uns zugänglich ist. Dieses Dokument wird von uns beiden aktualisiert und gepflegt.
- Meeting-Protokolle werden wie oben erwähnt im Annex 3.1 aufgeführt und gepflegt.
- Zentraler Zugriffspunkt für die Dokumentation zum Projekt: Die Projektseite auf OneDrive.
- Zentraler Zugriffspunkt für die Arbeitspakete und Zeitplaner zum Projekt: Trello Board
- Code Repository: Auf GitLab, gehostet von der FFHS. Alle Beide pflegen das Repository. <https://git.ffhs.ch/chantale.gihara/coronattack>
- Die Visualisierung der Aufgaben mittels Kanban: Das Board wird auf Trello von beiden gepflegt und fortlaufend aktualisiert. Darunter fällt auch die Zeiterfassung auf den Board-Karten. <https://trello.com/b/7HNVSgbQ/kanban-webe-coronattack>

## 1.7 Iterationsmodell

Wir werden im Projekt gemäss den 4 vergebenen Meilenstein iterativ aufteilen:

Dabei werden die Arbeitspakete gemäss den gewünschten Vorgaben in der Iteration berücksichtigt. Siehe Projektstrukturplan 1.8

## 1.8 Projektstrukturplan

Der Projektstrukturplan wurde gemäss iterativem Vorgehen entwickelt. Am Ende jeder Iteration gilt der der Meilenstein als abgeschlossen. Ausser bei der Iteration 2. 3 Arbeitspakete «2.11 Spiel 1», «Spiel 2», «Show High Score», werden in der Iteration 2 gestartet, werden aber erst in der Iteration 3 als abgeschlossen gelten.

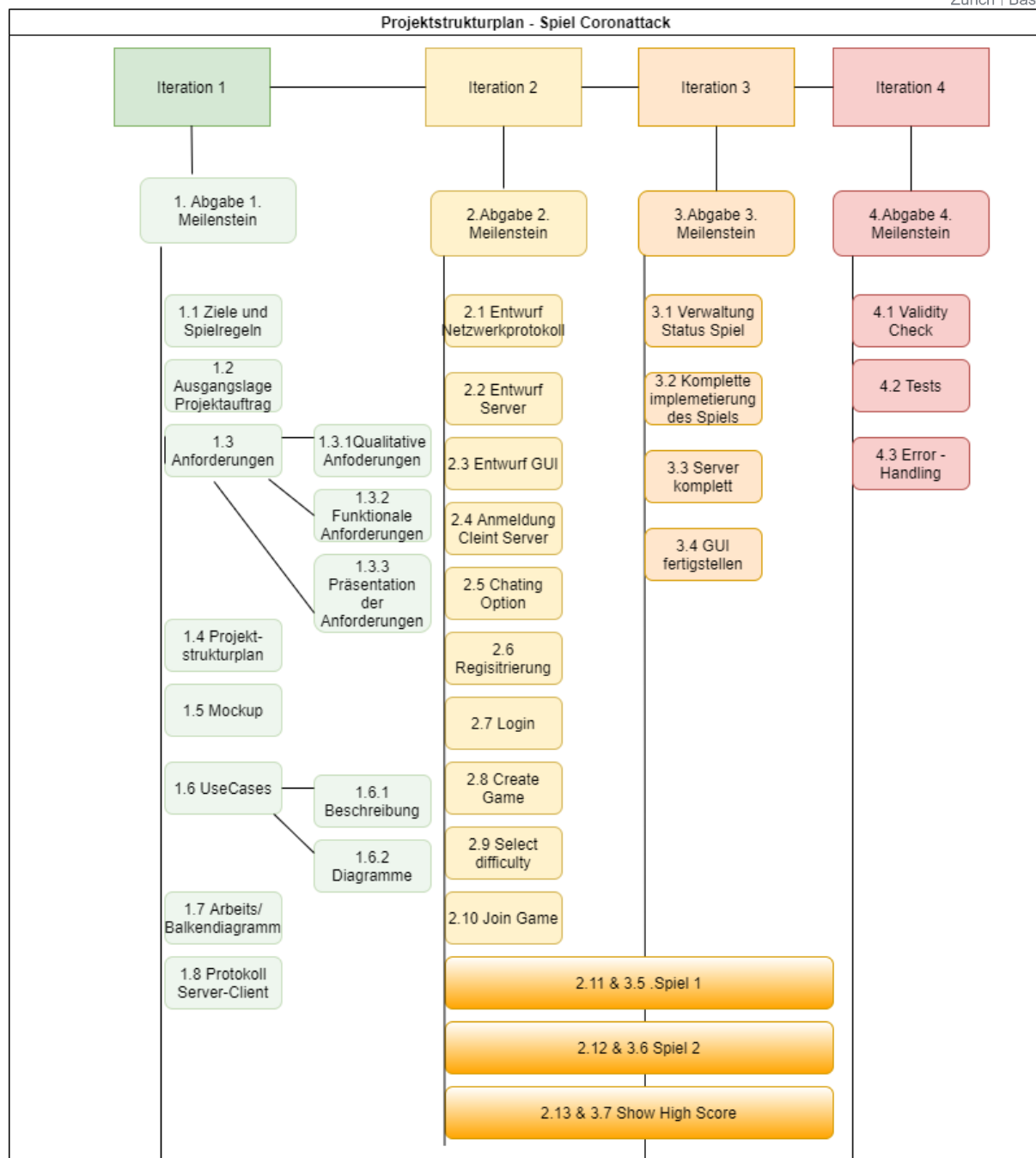


Abbildung 4: Projektstrukturplan



## 1.9 Arbeitsplan / Balkendiagramm

Die Arbeitsbalken und Timeline sind in Trello ersichtlich: <https://trello.com/b/7HNVSgbQ/kanban-webe-coronattack/timeline>

### 1.9.1 Meilenstein 1 / Iteration 1

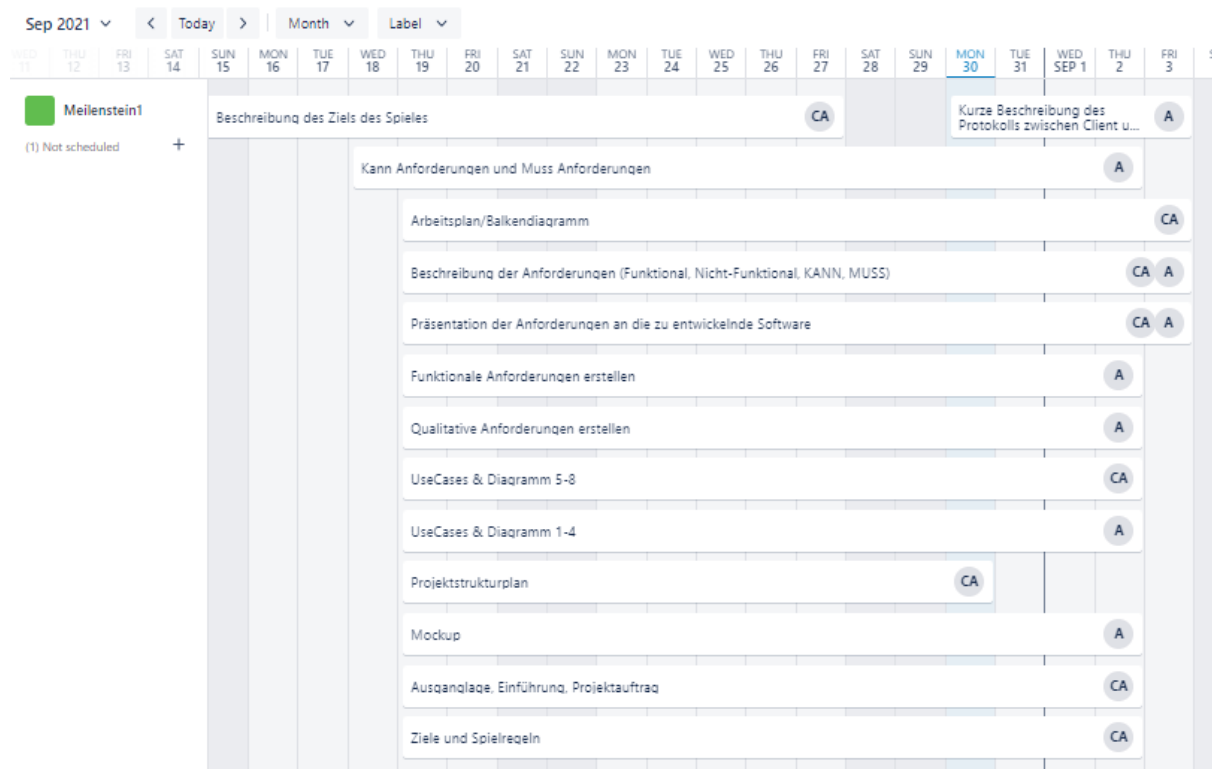


Abbildung 5: Arbeitsplan/Balkendiagramm - Meilenstein 1

### 1.9.2 Meilenstein 2 / Iteration 2

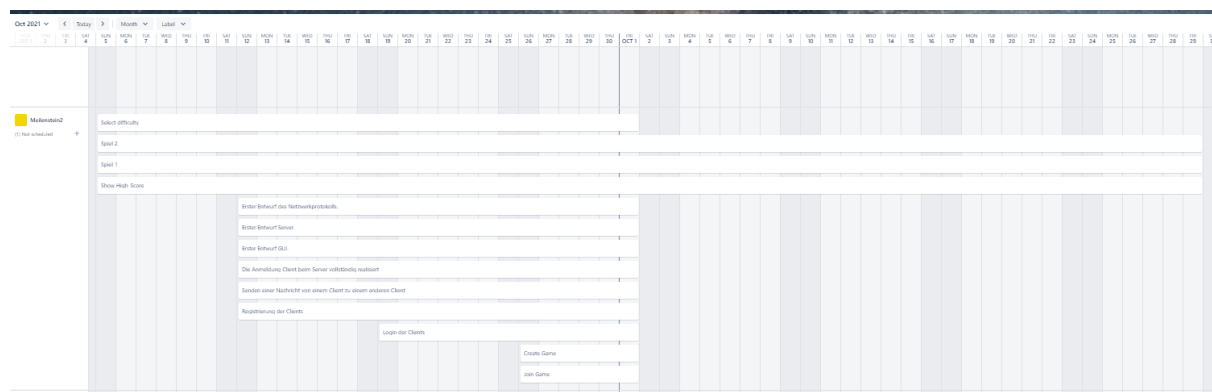


Abbildung 6: Arbeitsplan/Balkendiagramm - 2. Meilenstein

### 1.9.3 Meilenstein 3 / Iteration 3

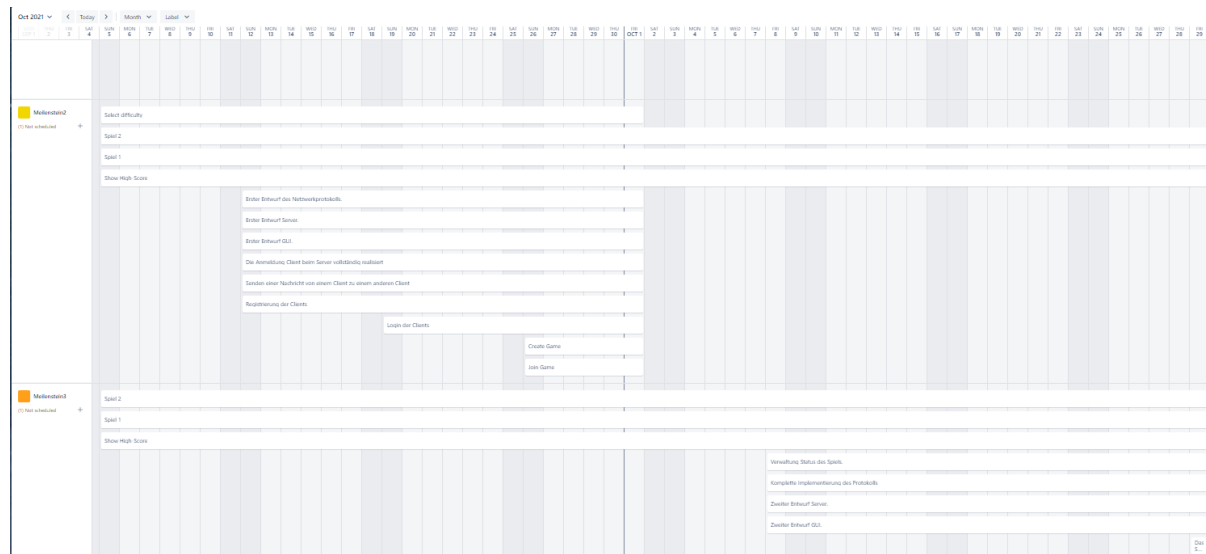


Abbildung 7:Arbeitsplan/Balkendiagramm - 3. Meilenstein

### 1.9.4 Meilenstein 4 / Iteration 4

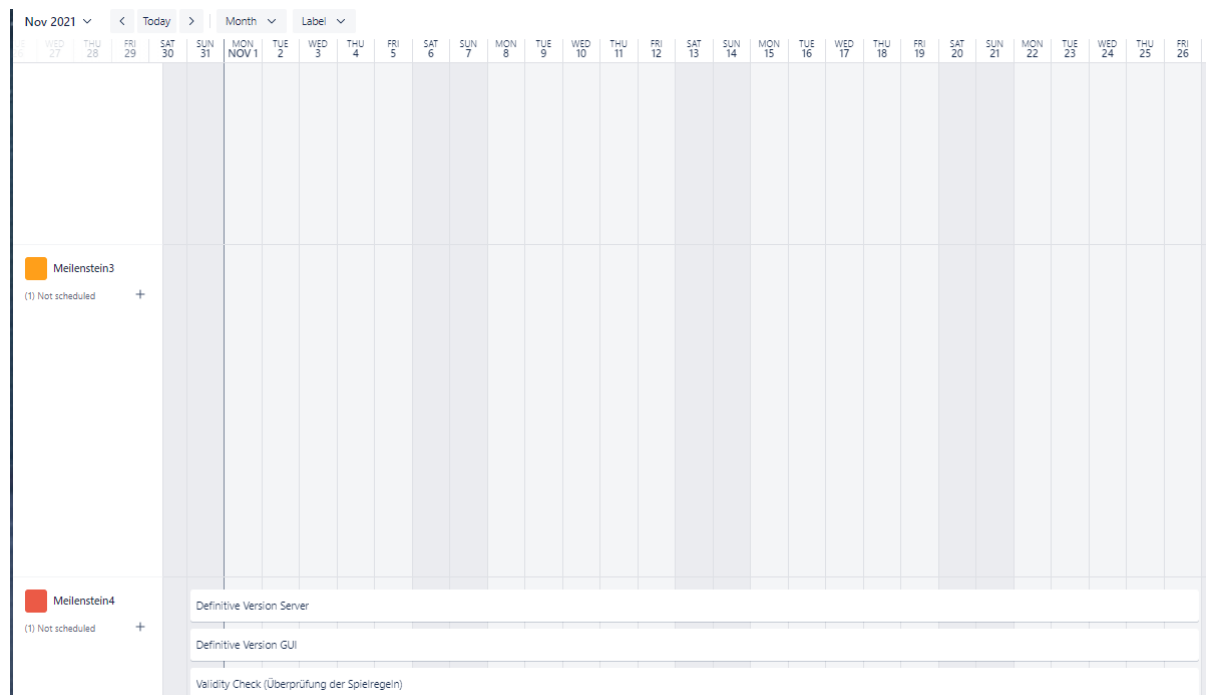


Abbildung 8: Arbeitsplan/Balkendiagramm - 4. Meilenstein

## 1.10 Qualitätsszenarien

Dieser Abschnitt beinhaltet konkrete Qualitätsszenarien, welche die zentralen Qualitätsziele, aber auch andere geforderte Qualitätseigenschaften besser fassen.

Mit Hilfe dieses Abschnitts wird es ermöglicht, Entscheidungsoptionen zu bewerten.

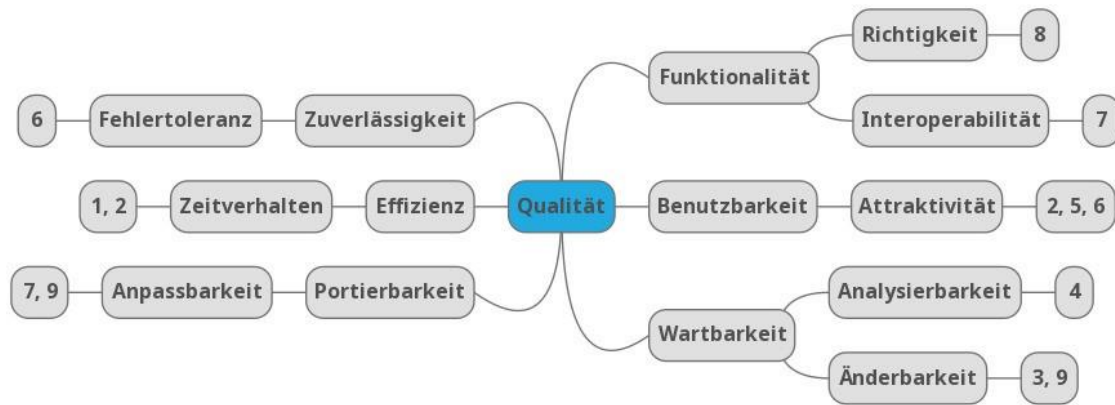


Abbildung 9: Qualitätsbaum des Spieles «Coronattack»

Nr	Szenario
1	Ein Benutzer möchte Corona-Ping Pong online spielen. Registrieren und online spielen erschließen sich ihm weniger als 5 Minuten.
2	Ein Benutzer erstellt ein Spiel online und kann in diesem virtuellen Raum ohne Probleme spielen.
3	Ein Entwickler implementiert eine neue Funktion in der Applikation. Er kann sie ohne Änderung und ohne Übersetzung vorhandenen Codes in bestehende Strategien integrieren.
4	Ein Entwickler möchte saubere Datenübertragung über die Schnittstellen analysieren. Der Aufwand dazu beträgt maximal eine Woche.
5	Ein Benutzer versucht sich mit einer E-Mail-Adresse anzumelden, die in der Datenbank bereits existiert. Eine Meldung wird angezeigt um den Kunden entsprechend darüber zu informieren.
6	Ein Benutzer versucht sich mit einem falschen Passwort einzuloggen. Eine Meldung wird angezeigt um den Kunden entsprechend darüber zu informieren.
7	Ein Entwickler möchte das Spiel verwenden (zum Laufen bringen), jedoch mit einer anderen Datenbank (nicht PostgreSQL). Das Einbinden erfordert keinerlei Programmieraufwand, die Konfiguration ist innerhalb von 10 Minuten durchgeführt und getestet.
8	Das Seiten-Routing funktioniert einwandfrei, wenn der Benutzer einen Hyperlink anklickt.
9	Ein Javascript-Programmierer will in „coronattack“ neue Komponenten hinzufügen. Die neuen Komponenten können ohne grosse Änderung am bestehenden Code implementiert und die Engine anschließend wie gewohnt eingebunden werden.

## 1.11 Randbedingungen

### 1.11.1 Technisch

Randbedingung	Erläuterungen, Hintergrund
Moderate Hardwareausstattung	Betrieb der Lösung auf einem Standard-Notebook
Webbrowser unabhängig von Betriebssystemen	Öffnen mit einem beliebigen Webbrowser da sie eine web basierte Betriebssystem unabhängige Applikation ist
Implementierung in Javascript	Entwicklung mit Javascript und JS Frameworks wie ReactJS
PostgreSQL DB frei verfügbar via Docker	DB frei verfügbar und kostenlos – Ausführen via Docker, DB Installation nicht nötig

### 1.11.2 Organisatorisch

Randbedingung	Erläuterungen, Hintergrund
Team	Theologos Baxevanos, Chantale Gihara
Zeitplan	Siehe Abschnitt 1.9.1
Entwicklungswerkzeuge	Entwurf mit Stift und Papier und Digitalisierung mit draw.io. Erstellung der Java-Quelltexte in VSCode.
Konfigurations- und Versionsverwaltung	Mit Git bei GitLab
Veröffentlichung als Open Source	Lizenz: <a href="https://opensource.org/licenses/MIT">https://opensource.org/licenses/MIT</a>

### 1.11.3 Konventionen

Konvention	Erläuterungen, Hintergrund
Architekturdokumentation	Terminologie und Gliederung nach dem deutschen arc42-Template
Kodierrichtlinien für Javascript	Siehe Abschnitt 2.6.2
Sprache (Deutsch vs. Englisch)	Benennung von Dingen (Komponenten, Schnittstellen) in Diagrammen und Texten innerhalb dieser (deutschen) arc42-Architekturdokumentation in Deutsch.

## 1.12 Risiken & technische Schulden

Risiken in einem Projekt für die Entwicklung einer Applikation sind reale oder virtuelle Ereignisse, die einen realen Schaden wie "Zeit", "Qualität" oder "Kosten" nach sich ziehen können.

Folgende Risiken wurden für die Durchführung des Projekts ermittelt:

- Überschreitung der geplanten Zeit aufgrund der fehlenden JS Erfahrung der Teammitglieder.
- Scheitern der Kommunikation aufgrund Fehler Dritter wie z. B. Netzwerk-Provider
- Ein oder beide Teammitglieder können nicht die geplante Verfügbarkeit erbringen.
- Ausfall von Teammitgliedern wegen Krankheit
- Applikation ist vor dem fünften Präsenz nicht im Abgabestatus.

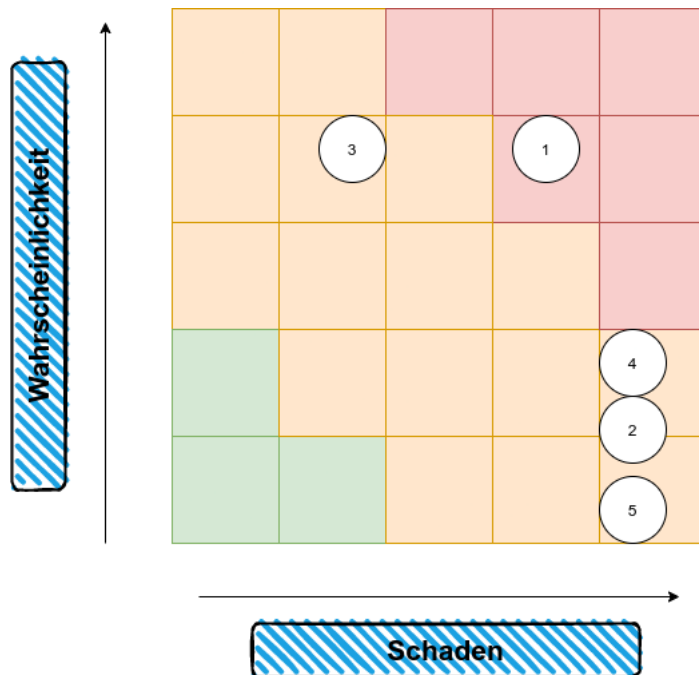


Abbildung 10: Risikomatrix

Die Risikomatrix ist eine Methode zur Risikoanalyse. Sie dient der systematischen Abschätzung und Bewertung von Risiken und ist ein Werkzeug, das bei der Erfassung, Bewertung und Visualisierung der erkannten Risiken unterstützt und als kontinuierliches Werkzeug genutzt wird inkl. zur Massnahmendefinition und Visualisierung der Restrisiken.

### Massnahmen gegen Risiken

Mit folgenden Massnahmen wird den erkannten Risiken entgegengewirkt:

- Regelmässige Austausche, um den Zwischenstand zu besprechen und evaluieren, ob die offenen Tasks anders priorisiert werden müssen (siehe Abschnitt 3.1)
- Mehrere Kommunikationskanäle (WhatsApp, E-Mails, MS Teams)
- Eskalation an Dozent

## 2 Softwareengineering

### 2.1 Anforderungen

Wir haben die Vorlage für die Anforderungen von (Ludewig, 2013) übernommen:

#### 2.1.1 Funktionale Anforderungen

Tabelle 1: FR - 001

ID	FR-001	
Typ	Funktionale Anforderung	
Titel	Plattformunabhängigkeit	
Aussage	Das Spiel kann auf der Endanwenderseite Plattform- und Gerätunabhängig verwendet werden.	
Begründung	Heutzutage spielt man webbasierte Spiele nicht nur am PC, sondern auch auf dem Handy bzw. Tablet. Das Betriebssystem variiert auch oft und ein vorausgesetztes Betriebssystem soll nicht verlangt werden. Somit soll die Applikation plattform- und Gerätunabhängig sein.	
Verbindlichkeit	Pflicht	
Priorität	Hoch	
Abnahmekriterien	ID	AC1
	Kriterium	Anwendung ist

	OK-Entscheid	Gerätunabhängig Die Anwendung kann auf verschiedenen Geräten und Browser abgerufen werden
Version	1.0	
Änderungsdatum	02.09.2021	
Autor	Theologos Baxevanos	
Zustand	genehmigt	

Tabelle 2: FR – 002

ID	FR-002	
Typ	Funktionale Anforderung	
Titel	Bedienerfreundliche Oberfläche	
Aussage	Das Spiel verfügt über eine bedienerfreundliche, grafische Oberfläche.	
Begründung	Das Zielpublikum für das Spiel ist nicht nur ICT-Affin Personen. Somit soll die Oberfläche bedienerfreundliche und verständlich sein.	
Verbindlichkeit	Pflicht	
Priorität	Hoch	
Abnahmekriterien	ID	AC2
	Kriterium	Oberfläche ist bedienerfreundlich.
	OK-Entscheid	UX-Tester geben Feedback, dass es benutzerfreundlich ist,
Version	1.0	
Änderungsdatum		
Autor	Theologos Baxevanos	
Zustand	genehmigt	

Tabelle 3: FR – 003

ID	FR-003	
Typ	Funktionale Anforderung	
Titel	Multiplayer	
Aussage	Das Spiel kann ohne einen Gegner nicht gestartet werden	
Begründung	"Coronattack" besitzt keine Single-Player Funktion, somit muss ein Gegner dem Spielraum betreten, sonst soll das Spiel nicht gestartet werden können.	
Verbindlichkeit	Pflicht	
Priorität	Hoch	
Abnahmekriterien	ID	AC3
	Kriterium	Ein Spieler allein kann das Spiel nicht starten
	OK-Entscheid	Spiel mit einem Spieler startet nicht.
Version	1.0	
Änderungsdatum	02.09.2021	
Autor	Theologos Baxevanos / Chantale Gihara	
Zustand	genehmigt	

Tabelle 4: FR – 004

ID	FR-004	
Typ	Funktionale Anforderung	
Titel	Registrierung	

Aussage	Nur registrierte Spieler können ins Spiel einloggen und ein neues Spiel starten bzw. einem Spielraum betreten. <small>Mitglied der SUPS</small>	
Begründung	Die Zug-Punkte werden in einer Tabelle "High Scores" eingetragen. Somit müssen alle Spieler identifizierbar sein.	
Verbindlichkeit	Pflicht	
Priorität	Hoch	
Abnahmekriterien	ID	AC4
	Kriterium	Nur registrierte Spieler können dem Spiel beitreten
	OK-Entscheid	Registrierte Spieler können dem Spiel beitreten. Nicht registrierter kann kein Spiel eröffnen oder beitreten.
Version	1.0	
Änderungsdatum	02.09.2021	
Autor	Theologos Baxevanos / Chantale Gihara	
Zustand	Genehmigt	

Tabelle 5: FR – 005

ID	FR-005	
Typ	Funktionale Anforderung	
Titel	Chat Funktion	
Aussage	Eine Chat Funktion steht den Spielern zur Verfügung, mit der sie während des Spiels miteinander kommunizieren können.	
Begründung	Es geht um ein interaktives online Spiel, wo der Spieler gegen einen echten Menschen spielt, somit soll eine Chat Funktion vorhanden sein.	
Verbindlichkeit	Pflicht	
Priorität	Hoch	
Abnahmekriterien	ID	AC5
	Kriterium	Eine Chatfunktion ist zur Verfügung
	OK-Entscheid	Spieler sendet eine Message zu einem anderen Spieler und die Nachricht kommt an und vice versa auch.
Version	1.0	
Änderungsdatum	02.02.2021	
Autor	Chantale Gihara / Theologos Baxevanos	
Zustand	Genehmigt	

Tabelle 6: FR – 006

ID	FR-006	
Typ	Funktionale Anforderung	
Titel	Schwierigkeitsauswahl	
Aussage	Die Schwierigkeitsauswahl steht für die Spieler zur Verfügung. Die Schwierigkeit hängt mit der Geschwindigkeit des Balles (Corona) zusammen. Je höher die Schwierigkeit, je schneller geht der Ball (Corona) hin und her. Es sollte min 3 Stufen geben. Easy, medium, hard.	

Begründung	Es war eine Vorgabe des Modules Schwierigkeiten einzubauen. <small>Mitglied der SUPS</small>	
Verbindlichkeit	Pflicht	
Priorität	Hoch	
Abnahmekriterien	ID	AC6
	Kriterium	Schwierigkeiten hard, medium, hard können ausgewählt werden
	OK-Entscheid	Die Schwierigkeiten können frei gewählt werden bei einer Spiel Eröffnung. Simple: das langsamste Medium: Schneller als Simple, aber langsamer als hard Hard: ist das schnellste der 3 Schwierigkeitsgraden.
Version	1.0	
Änderungsdatum	02.09.2021	
Autor	Chantale Gihara / Theologos Baxevanos	
Zustand	Genehmigt	

## 2.1.2 Qualitätsanforderung (Nicht funktionale Anforderungen)

Tabelle 7: QR – 001

ID	QR-001	
Typ	Qualitative Anforderung	
Titel	JavaScript Front & Backend	
Aussage	Das Spiel soll mit JavaScript (Front & Backend) entwickelt werden.	
Begründung	Beide Autoren haben geringe Erfahrung mit der Programmierungssprache "JavaScript" und möchten die Gelegenheit nutzen, das Spiel vollständig in JS zu programmieren, um Ihre JS Kenntnisse zu vertiefen.	
Verbindlichkeit	Von Vorteil	
Priorität	Mittel	
Abnahmekriterien	ID	AC7
	Kriterium	Frontend ist in JavaScript geschrieben
	OK-Entscheid	JavaScript ist im Frontend erkennbar
Version	1.0	
Änderungsdatum	02.09.2021	
Autor	Chantale Gihara / Theologos Baxevanos	
Zustand	genehmigt	

Tabelle 8: QR - 002

ID	QR-002	
Typ	Qualitative Anforderung	
Titel	Mehrsprachigkeit	
Aussage	Der Spieler kann die Spielanzeigesprache umstellen	



Begründung	Es soll möglich sein, die Sprache der Benutzeroberfläche auf Deutsch umstellen zu können, da wir das Spiel auf Englisch programmieren werden, jedoch unsere ersten Spieler sich in der Schweiz befinden werden. <small>Mitglied der SUPS</small>	
Verbindlichkeit	Von Vorteil	
Priorität	Mittel	
Abnahmekriterien	ID	AC8
	Kriterium	Sprache ist auf Deutsch und Englisch umstellbar
	OK-Entscheid	Sprachwechsel funktioniert überall und ist vorhanden.
Version	1.0	
Änderungsdatum	02.09.2021	
Autor	Chantale Gihara / Theologos Baxevanos	
Zustand	genehmigt	

Tabelle 9: QR - 003

ID	QR-003	
Typ	Qualitative Anforderung	
Titel	Anpassbare Benutzeroberfläche	
Aussage	Die Benutzeroberfläche ist vom Benutzer anpassbar	
Begründung	Dem Benutzer stehen verschiedene Optionen für die "in-game" Figuren zur Verfügung und er kann sie vor dem Beginn des Spiels ändern (den Ball, die Schläger und die "Smilies"). Somit spielt er immer mit Figuren seines Geschmacks.	
Verbindlichkeit	Von Vorteil	
Priorität	Mittel	
Abnahmekriterien	ID	AC9
	Kriterium	Verschiedene Graphische Icons für die Auswahl der Avatare steht zur Verfügung.
	OK-Entscheid	Avatare und Icons sind wählbar und kann hin und her gewechselt werden und wird übernommen.
Version	1.0	
Änderungsdatum	02.09.2021	
Autor	Chantale Gihara / Theologos Baxevanos	
Zustand	genehmigt	

(Ludewig, 2013)

## 2.2 Use Cases

Wir haben unser Spiel in folgende Use Cases unterteilt

- Registrierung
- Login
- Create Game
- Select Difficulty
- Join Game
- Spiel 1

- Spiel 2
- Scoring

Die Use Case Diagramme habe wir nach (Seidl, 2012) und (creately) erstellt.

### 2.2.1 Use Case 1: Registrierung

Tabelle 10: Registrierung

Merkmal	Beschreibung
Use Case Nr.	1
Bezeichnung	Registrierung
Kurzbeschreibung	Der Akteur (Spieler 1) registriert sich, um das Spiel zu spielen. Mit dem Registrieren wird ein Benutzeraccount erstellt. Er muss eine E-Mail-Adresse und ein Passwort eingeben. Mit diesen Zugangsdaten kann er sich ins Spiel einloggen.
Auslösendes Ereignis	Webseite des Spieles aufrufen
Akteur	Der Spieler 1
Vorbedingung	Zugriff auf die Webseite des Spieles
Nachbedingung	Der Benutzer meldet sich mit seinen Zugangsdaten an
Ergebnis	Benutzeraccount wurde erstellt und der Spieler ist eingeloggt
Szenarios	1) Der Spieler 1 registriert sich 2) Der Spieler loggt sich ein

### 2.2.2 Use Case 2: Login

Tabelle 11: Login

Merkmal	Beschreibung
Use Case Nr.	2
Bezeichnung	Login
Kurzbeschreibung	Der Akteur (Spieler) loggt sich ein, um das Spiel zu spielen. Mit dem Einloggen werden dem Spieler zwei Optionen angezeigt, entweder ein Spiel erzeugen oder einen Spielraum beizutreten.
Auslösendes Ereignis	Webseite des Spieles aufrufen
Akteur	Der Spieler
Vorbedingung	Zugriff auf die Webseite des Spieles
Nachbedingung	Der Spieler wählt, ob er ein Spiel erzeugen oder einen Spielraum beitreten möchte.
Ergebnis	Der Spieler ist mit seinem einzigartigen Account eingeloggt
Szenarios	1) Der Spieler hat sich bereits registriert 2) Der Spieler meldet sich an

### 2.2.3 Use Case 3: Create Game

Tabelle 12: Create Game

Merkmal	Beschreibung
Use Case Nr.	3
Bezeichnung	Create Game
Kurzbeschreibung	Der Akteur (Spieler 1) wählt die Option ein neues Spiel zu erzeugen aus, gibt das Spiel ein Name und setzt wie hoch die Schwierigkeit sein soll.
Auslösendes Ereignis	Ein neues Spiel wird erzeugt
Akteur	Der Spieler 1 (Spiel Erzeuger)
Vorbedingung	Zugriff auf die Webseite des Spieles, registrierter Spieler
Nachbedingung	Der Spieler 1 wartet auf die Anfrage eines weiteren Spielers, der den Spielraum beitreten möchte,
Ergebnis	Das Spiel wurde erzeugt
Szenarios	1) Der Spieler 1 hat sich bereits registriert 2) Der Spieler 1 meldet sich an

	3) Der Spieler 1 wählt die Option ein neues Spiel zu erzeugen 4) Der Spieler 1 gibt einen Namen für das Spiel ein 5) Der Spieler 1 setzt den Grad der Schwierigkeit des Spiels 6) Der Spieler 1 wartet auf die Anfrage eines anderen Spielers (Spieler 2)
--	--

## 2.2.4 Use Case 4: Schwierigkeit setzen

Tabelle 13: Select Difficulty

Merkmal	Beschreibung
Use Case Nr.	4
Bezeichnung	Select Difficulty
Kurzbeschreibung	Der Akteur (Spieler 1) wählt den Grad der Schwierigkeit des Spiels an.
Auslösendes Ereignis	Der Spieler hat angewählt ein neues Spiel zu erzeugen
Akteur	Der Spieler 1 (Spiel Erzeuger)
Vorbedingung	Zugriff auf die Webseite des Spieles, registrierter Spieler
Nachbedingung	Die Schwierigkeit wird gesetzt und der Spieler 1 darf einen Spielnamen eingeben und das Spiel starten
Ergebnis	Anhand der ausgewählten Schwierigkeit wird die Geschwindigkeit des Spielballes definiert
Szenarios	1) Der Spieler 1 hat sich bereits registriert 2) Der Spieler 1 meldet sich an 3) Der Spieler 1 wählt die Option ein neues Spiel zu erzeugen 4) Der Spieler 1 setzt den Grad der Schwierigkeit des Spiels 5) Der Spieler 1 gibt einen Namen für das Spiel ein und darf das Spiel starten

## 2.2.5 Use Case 5: Join Game

Tabelle 14: Join Game

Merkmal	Beschreibung
Use Case Nr.	5
Bezeichnung	Join Game
Kurzbeschreibung	Der Spieler möchte sich in ein schon eröffnetes Spiel einloggen und der Spieler, welcher das Spiel eröffnet hat, akzeptiert die Anfrage des neuen Gegenspielers
Auslösendes Ereignis	Auswahl Join Game
Akteur	Spieler 1 und Spieler 2
Vorbedingung	Ein Spieler hat schon ein Spiel eröffnet (Create Game)
Nachbedingung	Spiel1 startet
Ergebnis	Spieler, welcher das Spiel eröffnet hat, kann die Anfrage annehmen oder ablehnen, falls es angenommen wird, wird das Spiel 1 gestartet.
Szenarios	1) Spieler2 macht eine Anfrage für in einem Spiel beizutreten 2) Spieler1 akzeptiert die Anfrage 3) Spieler1 lehnt die Anfrage ab und wartet auf einen anderen Gegenspieler.

Abbildung 2: Use Case Diagramm 5 «Join Game»

## 2.2.6 Use Case 6: Spiel 1

Tabelle 15: Spiel 1

Merkmal	Beschreibung
Use Case Nr.	6

Bezeichnung	Spiel 1	Mitglied der SUPS
Kurzbeschreibung	Spiel 1 (Runde 1) versucht der Spieler die Menschen des Spielgegners mit Corona zu infizieren.	
Auslösendes Ereignis	Join Game oder Create Game	
Akteur	Spieler, Spiel	
Vorbedingung	2 Spieler sind im Game einer vom Create Game oder vom Join Game	
Nachbedingung	Ein Spieler hat gewonnen, alle Menschen des Gegners infiziert	
Ergebnis	Spiel 1 startet, Spieler gewinnt oder verliert, Spiel 1 wird beendet	
Szenarios	<ol style="list-style-type: none"> <li>1. Spieler versucht mit der Maske seine Menschen zu beschützen</li> <li>2. Spieler versucht mit Maske den Corona zu werfen, dass er die Menschen des Spielgegners trifft.</li> </ol>	

## 2.2.7 Use Case 7: Spiel 2

Tabelle 16: Spiel 2

Merkmal	Beschreibung
Use Case Nr.	7
Bezeichnung	Spiel 2
Kurzbeschreibung	Im Spiel 2 (Runde 2) müssen die Menschen des Gegners wieder geheilt werden.
Auslösendes Ereignis	Spiel 1 ist beendet, ein Spieler hat alle Menschen mit dem Corona infiziert
Akteur	Spieler, Spiel
Vorbedingung	Spiel 1 wurde beendet
Nachbedingung	High Score der Gesamttrangliste wird angezeigt
Ergebnis	Spiel 2 startet
Szenarios	<ol style="list-style-type: none"> <li>1. Spieler versucht mit einer fliegenden Spritze die Menschen wieder gesund zu treffen</li> <li>2. Spieler versucht seine Menschen von der Spritze zu beschützen</li> </ol>

## 2.2.8 Use Case 8: High Score anzeigen

Tabelle 17: High Score

Merkmal	Beschreibung
Use Case Nr.	8
Bezeichnung	High Score anzeigen
Kurzbeschreibung	Die High Scores werden angezeigt
Auslösendes Ereignis	Durch das Beenden von Spiel 2 oder durch das Anwählen «High Score» wird die Seite mit den High Score angezeigt
Akteur	Die Spieler (beenden des Spieles, anwählen des Spieles)
Vorbedingung	Spiele müssen beendet sein oder Spieler wählen die Ansicht «High Score»
Nachbedingung	Man kann ins Menu zurück navigieren oder man schliesst die Seite
Ergebnis	High Score der Spieler (Rangliste) wird angezeigt
Szenarios	<ol style="list-style-type: none"> <li>1. Spiel wird beenden</li> <li>2. Ansicht «High Scores» ausgewählt</li> </ol>

## 2.3 Mockups

Mockups werden hier zur Präsentation und Qualitätskontrolle eingesetzt. Sie dienen dazu, Vorstellungen und Anforderungen an die Benutzeroberfläche bezüglich Grundfunktionen, Navigation, Inhaltsarchitektur und Design mit dem Kunden abzustimmen.

Die Mockups wurden mit dem Webbasiereten Prototypentool "Figma" erstellt. Der Prototyp kann unter <https://www.figma.com/proto/GZGqOcwK1Uol5fH2filhUA/coronattack?node-id=2%3A2&scaling=min-zoom&page-id=0%3A1&starting-point-node-id=2%3A2> aufgerufen werden.

### 2.3.1 Startseite

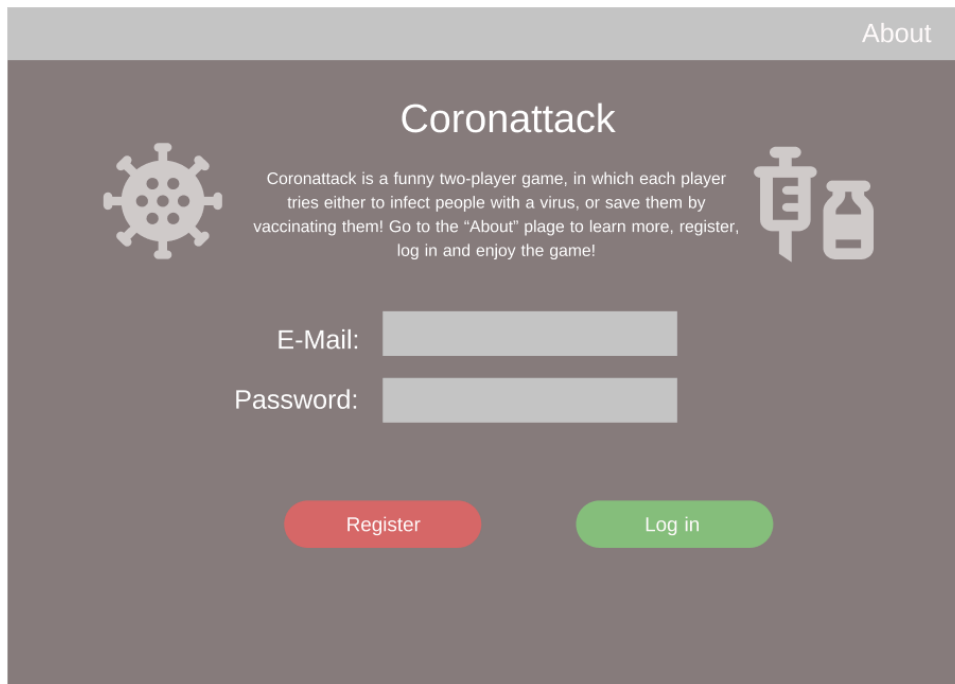


Abbildung 11: Startseite

Die Startseite erklärt in wenigen Worten das Spiel. Der Benutzer kann detaillierte Informationen lesen, wenn er die "About" Seite aufruft, sich registrieren bzw. einloggen.

### 2.3.2 About Seite

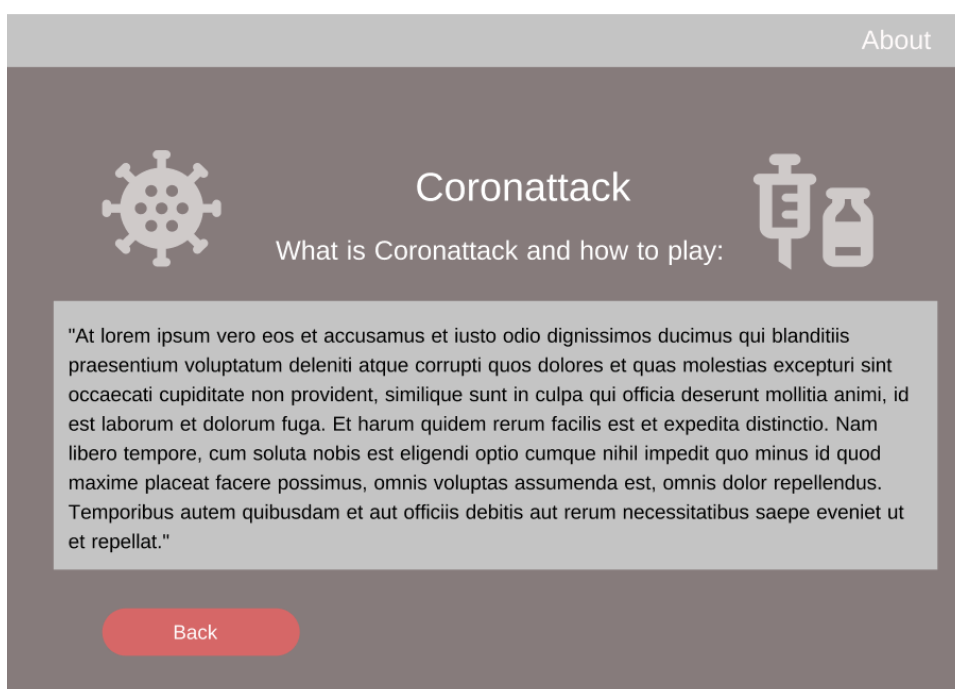


Abbildung 12: About Seite

In der About Seite wird das Spiel in Details erklärt, wie man sich registrieren, einloggen und spielen kann und einige Wörter über die Entwickler des Spiels.

### 2.3.3 Registrierung

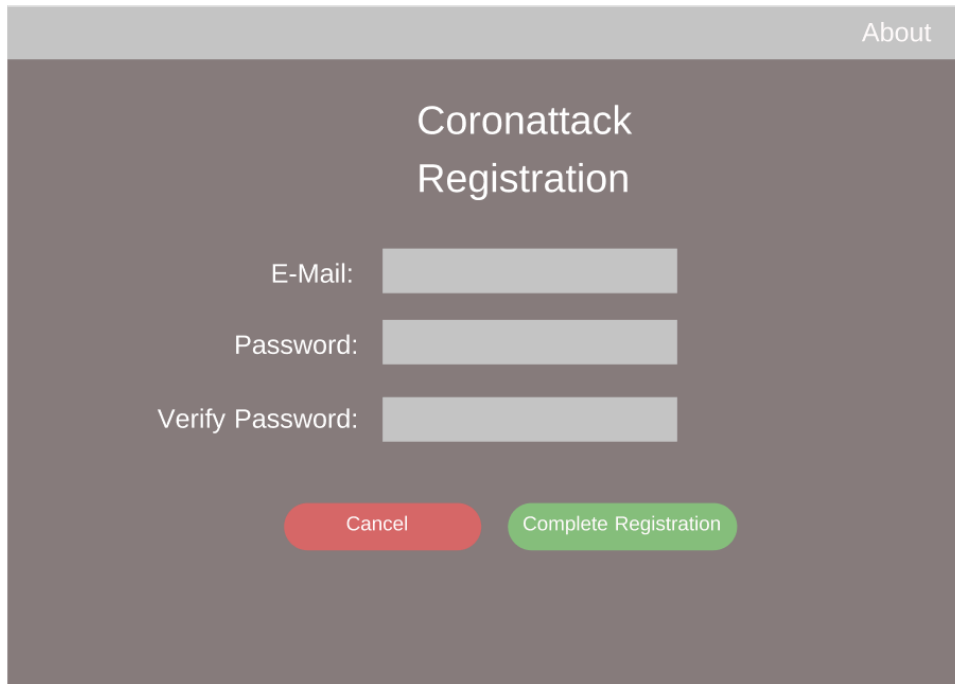


Abbildung 13: Registrierung

Auf dieser Seite kann der Spieler sich registrieren. Er muss eine E-Mail-Adresse und ein Passwort eingeben. Die E-Mail-Adresse dient als Benutzername.

### 2.3.4 Spiel erzeugen / beitreten

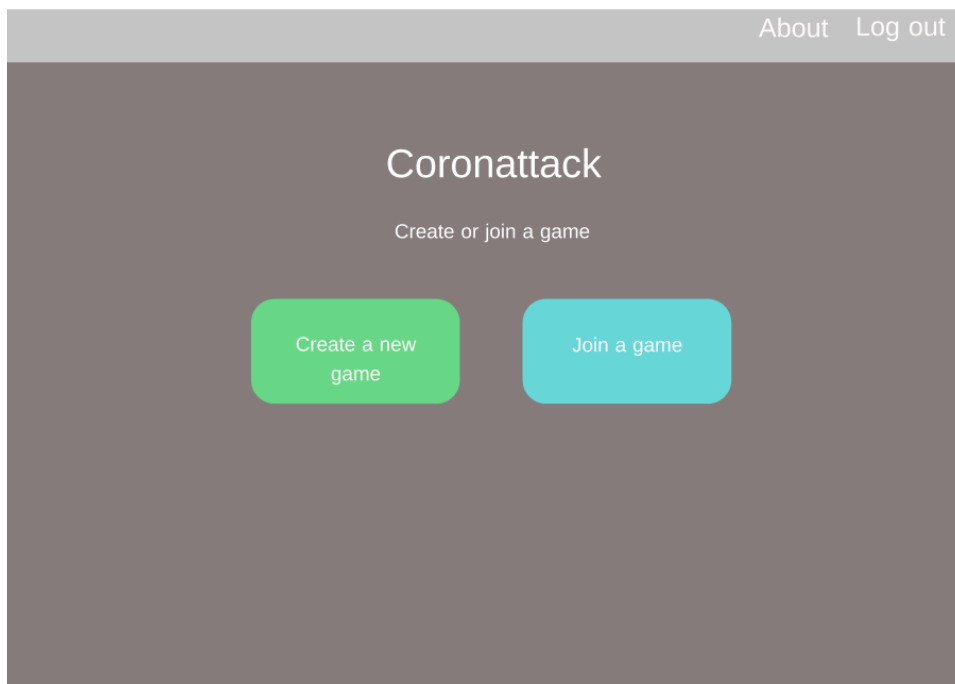


Abbildung 14: Spiel erzeugen oder beitreten

Auf dieser Seite kann der Spieler eine der zwei Optionen auswählen. Entweder kann er ein Spiel erzeugen oder einem existierenden Spielraum beitreten.

### 2.3.5 Spielraum Beitreten

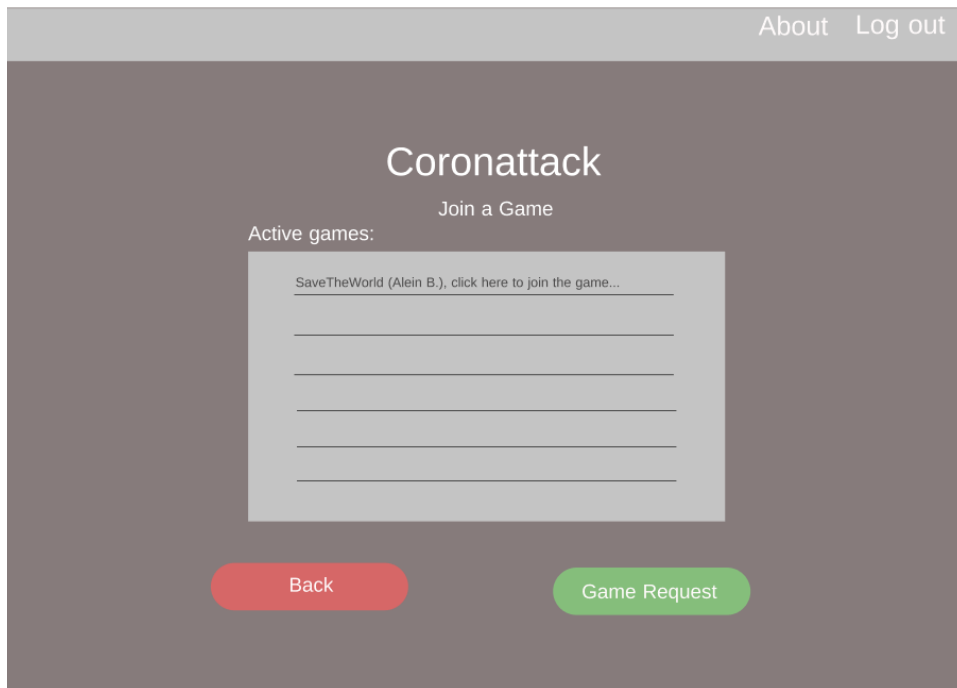


Abbildung 15: Spielraum beitreten

Der Spieler kann einem existierenden Spielraum beitreten. Wenn er auf den Button "Game request" drückt, wird dem anderen Spieler eine Anfrage geschickt. Der andere Spieler (der, der den Spielraum erzeugt hat) kann diese Anfrage entweder ablehnen oder annehmen. Der Spieler hat auf dieser Seite auch die Option eine Seite zurückzugehen, sich auszuloggen oder die About Seite aufzurufen.

### 2.3.6 Spielanfrage

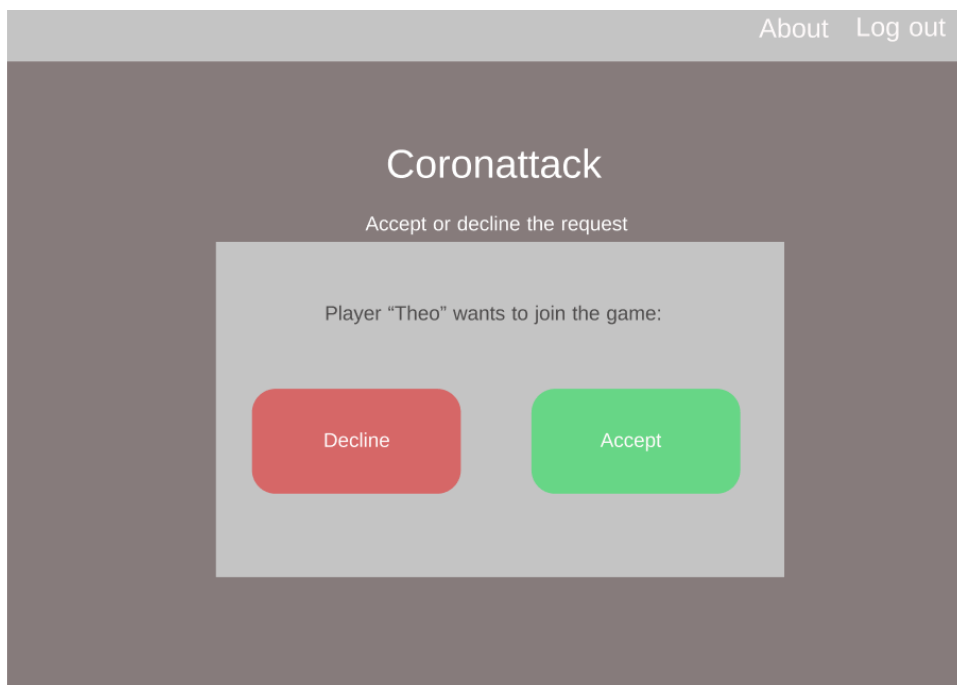


Abbildung 16: Spielanfrage

Das ist ein Pop-up Dialog, der dem Erzeuger des Spiels angezeigt wird, wenn ein anderer Spieler dem Spielraum beitreten möchte. Der Spielerzeuger hat die Möglichkeit die Anfrage abzulehnen oder anzunehmen.

### 2.3.7 Spiel erzeugen

Abbildung 17: Spiel erzeugen

Auf dieser Seite kann der Spieler ein Spiel erzeugen. Für da Spiel muss ein Name und die Schwierigkeit eingegeben werden.

Bei der Schwierigkeit "easy" wird die Ballgeschwindigkeit auf "5" und die Anzahl der infizierten/geheilten Personen auf "10" gesetzt.

Bei der Schwierigkeit "medium" wird die Ballgeschwindigkeit auf "10" und die Anzahl der infizierten/geheilten Personen auf "5" gesetzt.

Bei der Schwierigkeit "hard" wird die Ballgeschwindigkeit auf "15" und die Anzahl der infizierten/geheilten Personen auf "2" gesetzt.

### 2.3.8 Der Spielraum



Abbildung 18: Der Spielraum



Auf dieser Seite wird das Spielbrett dargestellt. Oben links und rechts werden die Punkte und die Namen der Spieler angezeigt. Im unteren Bereich gibt es ein Chatraum, wo sich die zwei Spielern "live" miteinander kommunizieren können. Die Spielregeln werden im Abschnitt 1.3 beschrieben.

## 2.4 Protokolle

### 2.4.1 Protokoll Client-Server

Im Rahmen dieser Semesterarbeit wird ein Spiel mit einer Client-Server Architektur entwickelt. Die Gliederung der Architektur richtet sich nach Arc42 (<https://arc42.org/>). Die Client-Server-Architektur ist ein Computermode, bei dem der Server die meisten vom Client zu verbrauchenden Ressourcen und Dienste hostet, bereitstellt und verwaltet. Bei dieser Art von Architektur sind ein oder mehrere Client-Computer über eine Netzwerk- oder Internetverbindung mit einem zentralen Server verbunden.

Diese Kommunikation zwischen Client und Server erfolgt mit Hilfe eines Protokolls. Das Protokoll, an das wir gedacht haben, ist das "HTTP" Protokoll.

(HyperText Transfer Protocol) ist ein Kommunikationsprotokoll und seine primäre Funktion ist eine Verbindung mit dem Server aufzubauen und HTML-Seiten an den Browser des Benutzers zurückzusenden.

Da wir aber ein Spiel entwickeln möchten, was eine ständig offene Verbindung zwischen Client und Server verlangt, haben wir ein Problem.

HTTP ist ein "zustandsloses" (stateless) Anfrage-/Antwortsystem. Die Verbindung zwischen Client und Server wird nur für die sofortige Anforderung aufrechterhalten und die Verbindung wird geschlossen. Nachdem der HTTP-Client eine TCP-Verbindung mit dem Server aufgebaut und ihm einen Anforderungsbefehl gesendet hat, sendet der Server seine Antwort zurück und schließt die Verbindung.

Die Lösung in diesem Problem ist ein modernes Kommunikationsprotokoll zu verwenden, und zwar "Websockets".

WebSocket ist ein Computer-Kommunikationsprotokoll, das Vollduplex-Kommunikationskanäle über eine einzige TCP-Verbindung bereitstellt. Das WebSocket-Protokoll wurde 2011 von der IETF als RFC 6455 standardisiert, und die WebSocket-API in Web IDL wird vom W3C standardisiert.

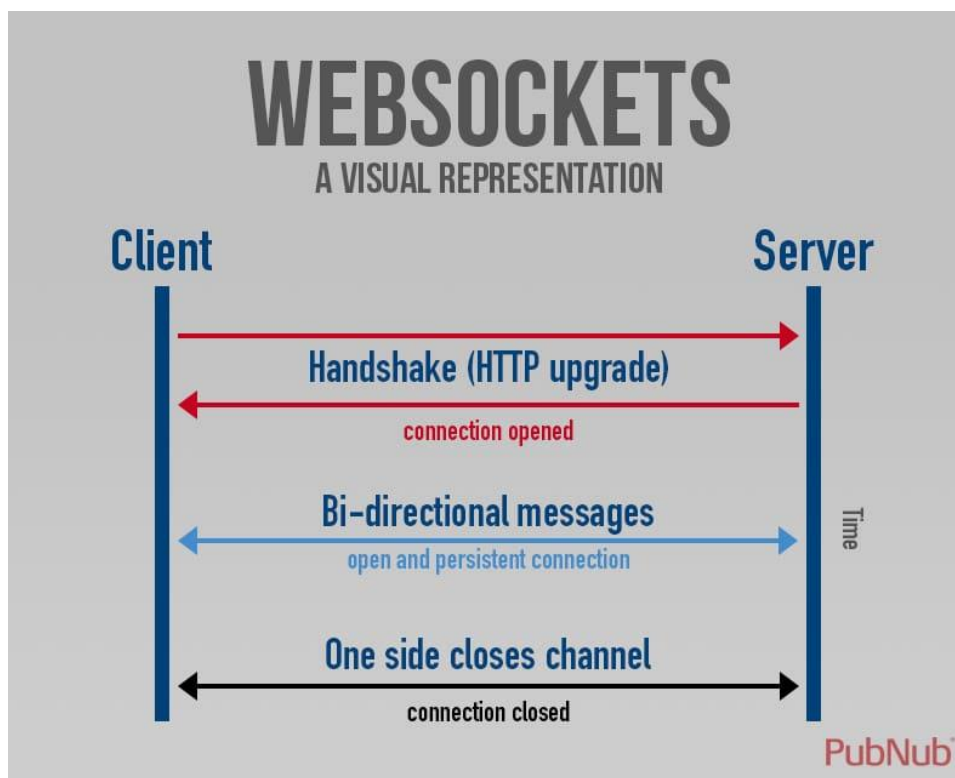


Abbildung 19: Websockets Kommunikation,  
<https://images.ctfassets.net/3prze68gbwl1/6glRdHedHRLNmco97gFajb/2d36a5ddfc47831ca737bbcf24e31d7c/WebSockets2.jpg>

Das WebSocket-Protokoll ermöglicht die Interaktion zwischen einem Webbrowser (oder einer anderen Clientanwendung) und einem Webserver mit geringerem Overhead als Halbduplex-Alternativen, wie HTTP-Polling, wodurch die Datenübertragung in Echtzeit vom und zum Server erleichtert wird. Dies wird ermöglicht, indem dem Server ein standardisierter Weg zur Verfügung gestellt wird, um Inhalte an den Client zu senden, ohne zuvor vom Client angefordert zu werden, und es ermöglicht, Nachrichten hin und her zu übergeben, während die Verbindung geöffnet bleibt. Auf diese Weise kann zwischen dem Client und dem Server eine laufende Konversation in beide Richtungen stattfinden.

Somit lösen wir mit Websockets unser Problem und können sicherstellen, dass es immer eine offene Verbindung für das Spiel gibt.

Die Objekte, die über dieses Protokoll übermittelt werden, werden ein JSON Format haben. JSON (JavaScript Object Notation) ist ein offenes Standarddateiformat und Datenaustauschformat, das lesbaren Text zum Speichern und Übertragen von Datenobjekten verwendet, die aus Attribut-Wert-Paaren und Arrays (oder anderen serialisierbaren Werten) bestehen.

### 2.4.1.1 Datentypen

Wir möchten folgende Datentypen verwenden, jedoch kommen bestimmt während der Programmierung noch weitere Typen dazu.

Tabelle 18: Typen Datenaustausch

Datentype	Beschreibung
username	Der Benutzername des Spielers
data	Objekt welches benötigt wird für den Austausch der Informationen zwischen Server und Client
gameld	UUID des Spiels auch als gameroom bekannt.
timestamp	Der Zeitstempel des Servers
userId	UUID, um den Spieler zu identifizieren für den Server und Client
email	Ist für den Benutzer die userId für sein Login Identifikation
gamestate	Der Status des Spieles, Spiel 1 Start oder Spiel2start oder Spiel1 beendet oder Spiel2 beendet.
userscore	Ist dem User sein high score
highscore	High Score ist eine Liste von den Spielers Scores
rank	Das ist der Rang des Spielers auf der Spielerrangliste
level	LevelId definiert die Schwierigkeit und die Geschwindigkeit des Spieles
message	Nachrichten werden versendet

## 2.4.2 Netzwerkprotokolle

### 2.4.2.1 Protokoll - Schnittstellen Registrierung

Tabelle 19: Protokoll – Schnittstelle Registrierung

Sender	Client
Receiver	Server
Type	register
Methode	POST
Data	<pre> {   "type": "register",   "data": {     "username": "[username]",     "email": "[email]",     "password": "[password]",     "password": "[password]"   } }</pre>
Code	201 created

### 2.4.2.2 Protokoll – Schnittstelle Login

Tabelle 20: Protokoll – Schnittstelle Login

Sender	Client
Receiver	Server
Type	login
Methode	GET

Data	<pre>"type": "", "data": {   "email": "[email]",   "password": "[password]" }</pre>
Code	202 accepted

### 2.4.2.3 Protokoll – Schnittstelle Create Game

Tabelle 21: Protokoll – Schnittstelle Create Game

Sender	Client
Receiver	Server
Type	createGame
Methode	POST
Data	<pre>"type": "createGame", "data": {   "gameId": "[gameId]",   "username": "[username]",   "userId": "[userId]",   "levelId": "[levelId]",   "gamestate": "[ gamestate]", }</pre>
Code	201 created

### 2.4.2.4 Protokoll – Schnittstelle Select level

Tabelle 22: Protokoll – Schnittstelle Select Level

Sender	Client
Receiver	Server
Type	level
Methode	GET
Data	<pre>"type": "level", "data": {   "levelId": "[levelId]",   "gameId": "[gameId]",   "userId": "[userId]",   "games": [ /* Liste aller aktiven Spieler, deren high score und level des Spieles.*/ ] }</pre>
Code	202 accepted

### 2.4.2.5 Protokoll – Schnittstelle Join Game

Tabelle 23: Protokoll – Schnittstelle Join Game

Sender	Server
Receiver	Client
Type	gameJoin
Methode	GET
Data	<pre>"type": "gameJoin", "data": {   "gameId": "[gameId]",   "levelId": "[levelId]",   "username": "[username]",   "gamestate": "[ gamestate]",   "userId": "[userId]" }</pre>
Code	202 accepted

### 2.4.2.6 Protokoll – Schnittstelle Spiel 1start

Tabelle 24: Protokoll – Schnittstelle Spiel 1 startet – Spieler1 / Server

Sender	Client[1]
Receiver	Server
Type	gameStart1
Methode	GET

Data	<pre>{   "type": "gameStart",   "data": {     "username": "[username]",     "gamestate": "[ gamestate]",     "userId": "[userId]",   } }</pre>
Code	200 OK

Tabelle 25: Protokoll – Schnittstelle Spiel 1 startet – Server / Spieler2

Sender	Server
Receiver	Client[2]
Type	gameStart
Methode	GET
Data	<pre>{   "type": "gameStart",   "data": {     "username": "[username]",     "gameld": "[gameld]",     "gamestate": "[ gamestate]",     "message": "[message]",   } }</pre>
Code	200 ok

Tabelle 26: Protokoll – Schnittstelle Spiel 1 startet – Spieler2 /Server

Sender	Client[2]
Receiver	Server
Type	gameStart
Methode	GET
Data	<pre>{   "type": "gameStart",   "data": {     "username": "[username]",     "gameld": "[gameld]",     "gamestate": "[ gamestate]",   } }</pre>
Code	200 ok

Tabelle 27: Protokoll – Schnittstelle Spiel 1 startet – Server / Spieler1

Sender	Server
Receiver	Client[1]
Type	gameStart
Methode	GET
Data	<pre>{   "type": "gameStart"   "data": {     "username": "[username]",     "gameld": "[gameld]",     "message": "[message]",   } }</pre>
Code	200 ok

#### 2.4.2.7 Protokoll – Schnittstelle Spiel 1 beendet

Tabelle 28: Protokoll – Schnittstelle Spiel 1 beendet – Server / Spieler1

Sender	Server
Receiver	Client[1]
Type	gameEnd
Methode	GET
Data	<pre>{   "type": "gameEnd",   "data": {     "username": "[username]",     "gameld": "[gameld]",     "userscore": "[highscore]"     "message": "[message]",   } }</pre>
Code	200 ok

Tabelle 29: Protokoll – Schnittstelle Spiel 1 beendet – Server / Spieler2

Sender	Server
Receiver	Client[2]
Type	gameEnd
Methode	GET
Data	<pre>{   "type": "gameEnd",   "data": {     "username": "[username]",     "gameld": "[gameld]",     "userscore": "[highscore]"     "gamestate": "[ gamestate]",   } }</pre>
Code	200 ok

Tabelle 30: Protokoll – Schnittstelle Spiel 1 – Server / Spieler2

Sender	Server
Receiver	Client[2]
Type	gameEnd
Methode	GET
Data	<pre>{   "type": "gameEnd",   "data": {     "username": "[username]",     "gameld": "[gameld]",     "userscore": "[userscore]"     "gamestate": "[ gamestate]",   } }</pre>
Code	200 ok

#### 2.4.2.8 Protokoll – Schnittstelle Spiel 2 startet

Tabelle 31: Protokoll – Schnittstelle Spiel 2 startet – Spieler1 / Server

Sender	Client[1]
Receiver	Server
Type	gameStart
Methode	GET
Data	<pre>{   "type": "gameStart",   "data": {     "username": "[username]",     "userId": "[userId]",   } }</pre>

	<pre>       "gamestate": "[ gamestate]",     }   } </pre>
Code	200 ok

Tabelle 32: Protokoll – Schnittstelle Spiel 2 startet – Server / Spieler2

Sender	Server
Receiver	Client[2]
Type	gameStart
Methode	GET
Data	<pre> {   "type": "gameStart",   "data": {     "username": "[username]",     "gameld": "[gameld]",     "gamestate": "[ gamestate]",   } } </pre>
Code	200 ok

Tabelle 33: Protokoll – Schnittstelle Spiel 2 startet – Spieler2 /Server

Sender	Client[2]
Receiver	Server
Type	gameStart
Methode	GET
Data	<pre> {   "type": "gamestate",   "data": {     "username": "[username]",     "gameld": "[gameld]",     "gamestate": "[ gamestate]",   } } </pre>
Code	200 ok

Tabelle 34: Protokoll – Schnittstelle Spiel 2 startet – Server / Spieler1

Sender	Server
Receiver	Client[1]
Type	gameStart
Methode	GET
Data	<pre> {   "type": "gameStart"   "data": {     "username": "[username]",     "gameld": "[gameld]",     "gamestate": "[ gamestate]",   } } </pre>
Code	200 ok

#### 2.4.2.9 Protokoll – Schnittstelle Spiel 2 beendet

Tabelle 35: Protokoll – Schnittstelle Spiel 1 beendet – Server / Spieler1

Sender	Server
Receiver	Client[1]
Type	gameEnd
Methode	GET
Data	<pre> {   "type": "gameEnd", </pre>

	<pre> "data": {   "username": "[username]",   "gameld": "[gameld]",   "userscore": "[userscore]"   "gamestate": "[ gamestate]", } } </pre>
Code	200 ok

Tabelle 36: Protokoll – Schnittstelle Spiel 1 beendet – Server / Spieler2

Sender	Server
Receiver	Client[2]
Type	gameEnd
Methode	GET
Data	<pre> {   "type": "gameEnd",   "data": {     "username": "[username]",     "gameld": "[gameld]",     "userscore": "[userscore]"     "gamestate": "[ gamestate]",   } } </pre>
Code	200 ok

#### 2.4.2.10 Protokoll – Schnittstelle High Score

Tabelle 37: Protokoll – Schnittstelle High Score – Server / Client

Sender	Server
Receiver	Client[1]
Type	highScore
Methode	POST
Data	<pre> {   "username": "[username]"   "userscore": "[userscore]" } </pre>
Code	201 created

Tabelle 38: Protokoll – Schnittstelle High Score – Server / Client

Sender	Server
Receiver	Client[1]
Type	highScore
Methode	GET
Data	<pre> {   "username": "[username]"   "userscore": "[userscore]",   "rank": "[rank]",   "highscores": [     {       "rank": "{rank}",       "username": "{username}",       "userId": "{userId}",       "userscore": "{Integer}"     }   ] } </pre>
Code	200 ok

#### 2.4.2.11 Protokoll – Schnittstelle Chat

Tabelle 39: Protokoll – Schnittstelle Chat – Client/Server

Sender	Client[i]
Receiver	Server
Type	chat
Methode	POST
	<pre>{   "type": "chat",   "data": {     "timestamp": "[timestamp   YYYY-MM-DDThh:mm:ss]",     "username": "[username]",     "userId": "[userId]",     "message": "[message]"   } }</pre>
Code	202 accepted

Tabelle 40: Protokoll – Schnittstelle Chat – Server/Client

Sender	Server
Receiver	Client[i]
Type	chat
	<pre>{   "type": "chat",   "data": {     "timestamp": "[timestamp   YYYY-MM-DDThh:mm:ss]",     "username": "[username]",     "userId": "[userId]",     "message": "[message]"   } }</pre>
Code	201 accepted

## 2.5 Server, Middleware, Datenbank und Chat Funktion

### 2.5.1 Server

Als Server Lösung haben wir uns für Express JS entschieden. Express.js, oder einfach Express, ist ein Back-End-Webanwendungs-Framework für Node.js, das als kostenlose Open-Source-Software unter der MIT-Lizenz veröffentlicht wird. Express.js ist sehr beliebt bei Entwicklern, die mit Javascript programmieren. Der war auch der Hauptgrund, wieso wir uns über dieses Framework entschieden haben.

Als Port haben wir den 3001 definiert und mit einer "GET" Anfrage stellen wir sicher, dass er läuft (Zeile 13-16, server.js)

Die ganze Backend Logik wird in der Datei "server.js" programmiert. Das bedeutet, die Schnittstellen werden hier definiert und implementiert, sowie die Datenbankqueries und die Ver- und Entschlüsselung der Passwörter.

Ein Stolperstein für uns war die Kommunikation zwischen React.JS und den Server (Express.JS). Obwohl die Endpoints in der Server.js definiert wurden, und auch seitens Frontend waren die Anfragen korrekt, gelangte die Kommunikation nicht.

Wir haben jedoch im Internet recherchiert und herausgefunden, dass die Nachrichten vom Frontend müssen von einem "Middleware" Dienst "übersetzt" werden. Erst nach der Implementierung des Middlewares hat es mit der Kommunikation funktioniert.

Die Endpoints für die POST Anfragen wurden auch in der "server.js" programmiert. Auf diese Endpoints greifen die Anfragen von Frontend zu und der Server prozessiert alles (zB Passwort Verschlüsselung, Datenbank Queries usw) und antwortet mit einem Reponse zurück.

Ausserdem wurde auch die Websocket (für die Chat Funktion) Kommunikation und Einstellung in dieser "server.js" Datei programmiert (siehe Abschnitt 2.5.4).



## 2.5.2 Middleware

Wie bereits erwähnt könnten wir nicht nachvollziehen wieso obwohl die Schnittstellen programmiert waren, die Kommunikation nicht funktionierte. Die Lösung war die Bibliothek "body-parser". Sie ist eine Node.js body parsing middleware. Sie analysiert die Anfragen, die aus dem Frontend empfangen werden und stellt dem Backend diese Attributen via einem req.body-Eigenschaft zur Verfügung.

## 2.5.3 Datenbank

Als Datenbank Lösung haben wir uns für Postgresql entschieden. PostgreSQL, auch bekannt als Postgres, ist ein kostenloses Open-Source-Managementsystem für relationale Datenbanken, das Wert auf Erweiterbarkeit und SQL-Konformität legt.

Die Konfiguration der Datenbank wurde in der Datei "dbConfig.js" programmiert. Aus Sicherheitsgründen wurden jedoch die Zugangsdaten in einer separaten versteckten Datei ".env" hinterlegt.

Die Queries befinden sich in der entsprechenden Endpoints und sind einfache SQL Anfragen. Mit Hilfe dieser Queries kann das Einloggen, die Registrierung, sowie die Validation realisiert werden.

Zum Beispiel, wenn man sich registrieren möchte, wird der Datenbank eine Anfrage geschickt, ob diese E-Mail Adresse bereits existiert. Sollte das der Fall sein, dann erfolgt die Registrierung nicht und dem Endbenutzer wird eine Fehlermeldung angezeigt.

Ebenfalls beim Einloggen, wird der Datenbank gefragt ob die E-Mail Adresse existiert, wenn ja dann wird das Passwort dieser Zeile entschlüsselt und mit dem angegebenen Passwort verglichen. Sollten die zwei Passwörter identisch sein, dann kann sich der Benutzer ins System einloggen.

## 2.5.4 Chat Funktion

Für die Chat Funktion wurde die Bibliothek Socket.io verwendet. Socket.IO ist eine JavaScript-Bibliothek für Echtzeit-Webanwendungen. Es ermöglicht eine bidirektionale Kommunikation in Echtzeit zwischen Web-Clients und Servern. Es besteht aus zwei Teilen: einer clientseitigen Bibliothek, die im Browser ausgeführt wird, und einer serverseitigen Bibliothek für Node.js.

Die Konfiguration wurde auch in der Datei "server.js" programmiert. Ein http Server wird dazu erzeugt, der über den Port 3002 läuft. Mit einem console.log stellen wir auch sicher dass der Server doch läuft. Dann wird ein socket-End Point im Backend erstellt, der vom Frontend verwendet werden kann.

Der Grund, wieso wir uns für diese Bibliothek entschieden haben und nicht direkt Websockets (wie zum Beispiel in unserem Buch) ist da Socket.IO in erster Linie das WebSocket-Protokoll mit Polling als Fallback-Option verwendet, bietet aber die gleiche Schnittstelle. Obwohl es einfach als Wrapper für WebSocket verwendet werden kann, bietet es viele weitere Funktionen, darunter das Senden an mehrere Sockets, das Speichern von Daten, die jedem Client zugeordnet sind, und asynchrone E/A (I/O).

## 2.5.5 Docker

Um die Applikation ohne grosse Mühe ausführen zu können wollen wir das ganze über Docker laufen lassen. Jedoch funktioniert dies noch nicht ganz. Im Brach "notime" ist dieser Versuch ersichtlich. Jedoch haben wir dies noch nicht ins master migriert. Aus diesem Grund bei der zweiten Abgabe müssen die Servern und die Datenbank manuell gestartet werden (vom Brach "main"). Zurzeit läuft nur die Datenbank über docker.

Das Vorgehen lautet:

- In der "backend" Verzeichnis: "npm run dev" ausführen.
- In der "frontend" Verzeichnis: "npm start" ausführen.
- In der "app" Verzeichnis: "docker compose up" ausführen.
- Dann mit einem beliebigen Webbrowser die Seite "localhost:3000" aufrufen und die Applikation testen.

## 2.6 Architektur

### 2.6.1 Architektur und Coderichtlinien

Für die Architektur der Semesterarbeit wollten wir ein Design Pattern (Architekturentwurf) verwenden, wie zum Beispiel MVC, MVVM oder MVP. Mit dem ReactJS Framework ist dies aber nicht einfach. React hat meistens eine Komponentenbasierte Architektur und das ist auch bei uns der Fall.

Um eine klare Gliederung zu erreichen haben wir unsere Backend und Frontend Umgebung getrennt.

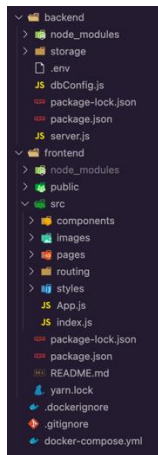


Abbildung 20: Gliederung des Projektes

Im Backend Verzeichnis befindet sich die Datenbank und die Serverkonfiguration. Im Frontend Verzeichnis findet man die Komponenten, das Design, die Seiten, die externe Daten (z.B. Bilder) und die Routing-Logik.

Mit einer «MVC» Logik, in unserem Projekt ist «Pages» das «View», was der Benutzer sieht, wenn er die Seiten aufruft. «Controller» sind die Funktionen der Komponenten, die das «Model» manipulieren. «Model» sind die «react States». State ist ein einfaches JavaScript-Objekt, das von React verwendet wird, um Informationen über die aktuelle Situation der Komponente darzustellen. Diese «states» aktualisieren die View und zeigen dem Benutzer die Änderungen.

Im folgenden Bild wird die Komponentenbasierte React-MVC Architektur visuell dargestellt:

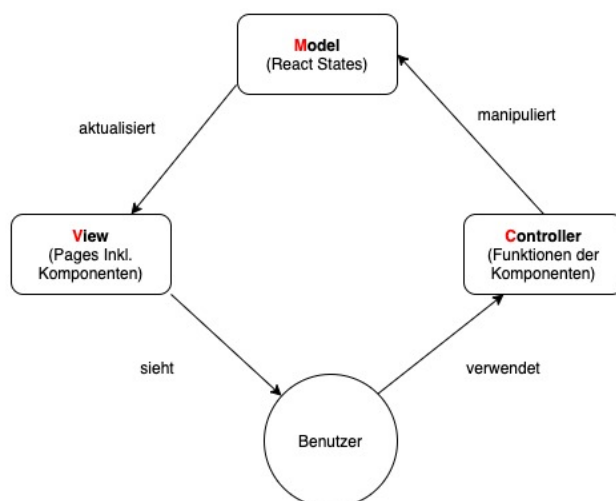


Abbildung 21: Architektur des Projektes- Komponentenbasierte MVC Architektur

### Erklärung des Modells

Model-View-Controller, kurz MVC, ist ein gängiges Muster in Web-Frameworks, wo es überwiegend zum Erstellen von HTML-Anwendungen verwendet wird. Das Modell bezieht sich auf die Daten der Anwendung, die Sicht auf die Datenpräsentation der Anwendung und der Controller auf den Teil des Systems, der für die Verwaltung von Eingaben, die Aktualisierung von Modellen und die Produktion von Ausgaben verantwortlich ist.

Web-UI-Frameworks können als aktionsbasiert oder komponentenbasiert kategorisiert werden. In einem aktionsbasierten Framework werden HTTP-Anforderungen an Controller weitergeleitet, wo sie vom Anwendungscode in Aktionen umgewandelt werden. In einem komponentenbasierten Framework werden HTTP-Anforderungen gruppiert und normalerweise von Framework-Komponenten mit geringer oder keiner Interaktion vom Anwendungscode verarbeitet. Mit anderen Worten, in einem komponentenbasierten Framework wird der Großteil der Controller-Logik vom Framework anstelle der Anwendung bereitgestellt.

Die MVC API ist in der Server.js Datei zu finden. Dort wurde die Logik und die Endpoints programmiert. Die Schnittstellen (siehe 2.4.2) stellen noch sicher dass Frontend mit Backend problemlos kommuniziert.

### **Server: Express.js**

Express.js, oder einfach Express, ist ein Back-End-Webanwendungs-Framework für Node.js, das als kostenlose Open-Source-Software unter der MIT-Lizenz veröffentlicht wird. Es wurde für die Erstellung von Webanwendungen und APIs entwickelt. Es wurde als De-facto-Standardserver-Framework für Node.js bezeichnet.

### **DB: PostgreSQL**

PostgreSQL, auch bekannt als Postgres, ist ein kostenloses, relationales Open-Source-Datenbankverwaltungssystem, bei dem Erweiterbarkeit und SQL-Konformität im Vordergrund stehen.

### **Backend: Node.js**

Node.js ist eine Open-Source-, plattformübergreifende Back-End-JavaScript-Laufzeitumgebung, die auf der V8-Engine ausgeführt wird und JavaScript-Code ausserhalb eines Webbrowsers ausführt. Mit Hilfe des Node.js können wir das Backend in Javascript programmieren und somit haben wir eine Full-Stack Javascript Applikation.

### **Frontend: ReactJS**

React ist eine kostenlose Open-Source-Frontend-JavaScript-Bibliothek zum Erstellen von Benutzeroberflächen oder UI-Komponenten. Es wird von Facebook und einer Community aus einzelnen Entwicklern und Unternehmen gepflegt. React kann als Basis bei der Entwicklung von Single-Page- oder mobilen Anwendungen verwendet werden. Der war auch der Haupt Grund, wieso wir uns für React in unserem Projekt entschieden haben.

### **Relevante Einflussfaktoren**

- Randbedingungen
- Betrieb der Frontends zumindest auf einem modernen Browser (nicht internet explorer)
- Docker muss auf dem Computer installiert sein und den Befehl «docker compose up» auszuführen
- Massgeblich betroffene Qualitätsmerkmale (→ siehe Qualitätsziele)
- Qualitätsziel: Interoperabilität
- Qualitätsziel: Änderbarkeit
- Anpassbarkeit (zukünftige neue Funktionen des Spieles)
- Betroffene Risiken (→ siehe Risiken)

### **Betrachtete Alternativen und Entscheidungen**

Für das Frontend könnten wir auch Java-FX verwenden. JavaFX ist eine Softwareplattform zum Erstellen und Bereitstellen von Desktop-Anwendungen sowie Rich-Web-Anwendungen, die auf einer Vielzahl von Geräten ausgeführt werden können. JavaFX unterstützt Desktop-Computer und Webbrowser unter Microsoft Windows, Linux und macOS sowie mobile Geräte mit iOS und Android. Jedoch wollten wir die ganze Applikation in einer neuen für uns Programmiersprache entwickeln, Javascript. Dies vereinfacht auch den Code, da wir das Front- und Backend in derselben Sprache programmieren.

Für das Backend hätten wir auch andere Technologien auswählen können, wie zum Beispiel Java oder PHP. Aber wie bereits erwähnt, wir wollten diese Chance und gleichzeitig für uns Herausforderung zu nehmen um eine Applikation zu entwickeln, die vollständig in Javascript programmiert ist.

Für den Kommunikationsprotokoll haben wir uns für Websocket entschieden, jedoch wäre das

http/1.1 eine alternative.

HTTP ist ein "zustandsloses" (stateless) Anfrage-/Antwortsystem. Die Verbindung zwischen Client und Server wird nur für die sofortige Anforderung aufrechterhalten und die Verbindung wird geschlossen. Nachdem der HTTP-Client eine TCP-Verbindung mit dem Server aufgebaut und ihm einen Anforderungsbefehl gesendet hat, sendet der Server seine Antwort zurück und schließt die Verbindung.

Die Entscheidung für Websockets wurde getroffen, da mit dem sichergestellt werden kann, dass es immer eine offene Verbindung für die Applikation gibt.

Das WebSocket-Protokoll ermöglicht die Interaktion zwischen einem Webbrowser (oder einer anderen Clientanwendung) und einem Webserver mit geringerem Overhead als Halbduplex-Alternativen, wie HTTP-Polling, wodurch die Datenübertragung in Echtzeit vom und zum Server erleichtert wird. Dies wird ermöglicht, indem dem Server ein standardisierter Weg zur Verfügung gestellt wird, um Inhalte an den Client zu senden, ohne zuvor vom Client angefordert zu werden, und es ermöglicht, Nachrichten hin und her zu übergeben, während die Verbindung geöffnet bleibt. Auf diese Weise kann zwischen dem Client und dem Server eine laufende Konversation in beide Richtungen stattfinden.

Die Implementierung des WebSocket-Protokolls wird mit Hilfe der Bibliothek «Socket.io» ermöglicht.

Socket.IO ist eine JavaScript-Bibliothek für Echtzeit-Webanwendungen. Es ermöglicht eine bidirektionale Kommunikation in Echtzeit zwischen Web-Clients und Servern. Es besteht aus zwei Teilen: einer clientseitigen Bibliothek, die im Browser ausgeführt wird, und einer serverseitigen Bibliothek für Node.js. Beide Komponenten haben eine nahezu identische API.

## **Diagramme**

Die Diagramme werden nach UML-Standard umgesetzt.

## **Glossar**

Fachbegriffe sind, sofern nötig, im globalen Glossar der Projektdokumentation aufgenommen (siehe Abschnitt Glossar).

## **Dokumentationssprache**

Deutsch. Die Kommentare im Source Code sind jedoch in Englischer Sprache geschrieben.

## **Applikationssprache**

Englisch.

### **2.6.2 Coderichtlinien**

Heutzutage kann man diverse Coderichtlinien im Internet finden. Viele grosse Firmen wie Google haben ihre eigenen Coderichtlinien, die sie veröffentlicht haben.

Die Wahl der Richtlinien ist meist subjektiv, im Zentrum steht dabei aber immer die Lesbarkeit und Konsistenz des Codes. Dies ist besonders wertvoll, wenn in einem Team gearbeitet wird. In unserem Projekt werden wir die Richtlinien von Google einsetzen.

Javascript: Javascript Style Guide von Google ( <https://google.github.io/styleguide/jsguide.html> )

Auch wenn die Entwicklung mit verschiedenen Editoren und Tools möglich wäre, zeigte sich in der Vergangenheit ein einfacherer Umgang bei einer einheitlichen Entwicklungsumgebung. Hier haben wir uns für den Texteditor VSCode (aktuellste Version) entschieden.



“verrät”.

Die Spieler bewegen sich nur vertikal, also weisen wir W und S zu, um den linken Spieler zu bewegen und UP und DOWN für den rechten.

Zu diesem Zweck hören wir auf das keydown-Ereignis und jedes Mal, wenn es auftritt, überprüfen wir den keyCode des Ereignisses, um die Bewegung auszuführen.

Um unsere Aufgaben einfach zu erledigen, habe ich eine KeyListener-Klasse erstellt, um dies zu handhaben. Es hört nur auf Keydown/Keyup-Ereignisse und verfolgt, welche Tasten gedrückt werden.

Das Kugelelement wird nicht vom Benutzer gesteuert. Hier herrschen die geometrische Operationen. Stattdessen bewegt es sich mit einer X- und Y-Geschwindigkeit im Ansichtsfenster. Wenn der Ball ein Paddel trifft, ändert er sich in die entgegengesetzte X-Richtung, behält aber die Y-Geschwindigkeit bei. Wenn der Ball den oberen oder unteren Rand des Bildschirms trifft, bewegt er sich in die entgegengesetzte Y-Richtung, während die X-Geschwindigkeit gleich bleibt. Überquert er die linke oder rechte Seite des Bildschirms, punktet der Spieler der gegenüberliegenden Seite. Die Position wird bei jedem Aufruf von update() aktualisiert.

Um den Ballkollision zu handeln (Kollisionerkennung), wir müssen den "Tunneleffekt" vermeiden. Wir müssen also "raten", wo sich der Ball im Moment der Kollision befand und den Ball in diese Position verschieben.

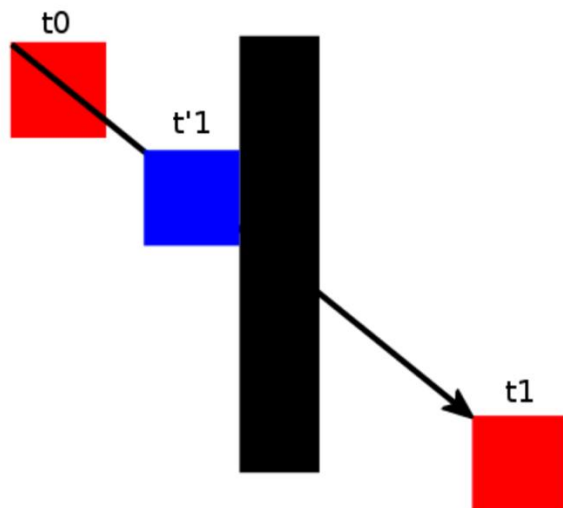


Abbildung 23: vermeiden den Ballkollision

“Last but not least” wird auch der “Score” berechnet, gespeichert und auf dem Canvas (Brett) angezeigt. Jedoch wird er noch nicht in einem react State (hook) gespeichert. Dies wird noch gemacht, damit er auch zwischen Komponenten übertragen werden kann.

## 3 Annexes

### 3.1 Protokolle

#### 3.1.1 Protokoll vom 02.09.2021

Teilnehmer	Theologos Baxevanos
	Chantale Gihara
Datum	02.09.2021
Ort	Starbucks Zürich Oerlikon
Informationen	<ul style="list-style-type: none"> <li>• Use Case Diagramme besprechen</li> <li>• Mock up wird abgenommen</li> <li>• Anforderungen werden zusammen reviewt und validiert.</li> <li>• Abgabe wird besprochen</li> </ul>
Diskussionen	<ul style="list-style-type: none"> <li>• Protokoll und Schnittstelle sind wir nicht ganz sicher. Wir wünschen uns gerne ein Feedback des Dozenten nach unserer Abgabe.</li> <li>• Das Schnittstellen Protokoll ist in Meilenstein 1 noch nicht gefordert, wir möchten es aber schon verstehen, deshalb implementieren wir es</li> </ul>

	schon in die Arbeit, damit wir so rasch als möglich ein Feedback dazu haben.
Meilensteine	<ul style="list-style-type: none"> <li>• Meilenstein 1 abgeschlossen somit Iteration 1 abgeschlossen</li> <li>• Meilenstein 2 kann nun gestartet werden</li> <li>• Zeitplanung der Meilensteine und der Arbeitspaket sind im Kanban Board von Trello siehe Link next Steps ersichtlich.</li> </ul>
Next Steps	<ul style="list-style-type: none"> <li>• <a href="https://trello.com/b/7HNVSgbQ/kanban-webe-coronattack">https://trello.com/b/7HNVSgbQ/kanban-webe-coronattack</a></li> <li>• Chantale startet mich der ersten Phase der Programmierung</li> <li>• Theo zieht in Meilenstein 2 seine Karten, verfeinert die Userstory mit Checklist.</li> </ul>
Nächstes Meeting	<ul style="list-style-type: none"> <li>• 11.09.2021 vor Ort PVA2 WebE</li> </ul>

### 3.1.2 Protokoll vom 21.08.2021

Teilnehmer	Theologos Baxevanos
	Chantale Gihara
Datum	21.08.2021
Ort	Starbucks, Zürich Oerlikon Bahnhof
Informationen	<ul style="list-style-type: none"> <li>• Arbeitsaufteilung</li> <li>• Meilensteine noch einmal revidieren.</li> <li>• Anforderungen werden zusammen reviewt und validiert.</li> <li>• Bis wann muss abgegeben werden</li> <li>• Arbeitspakete und Projektstrukturplan, wie Balkendiagramm wird von Chantale übernommen</li> <li>• Jede Arbeit wird bei Beendung</li> </ul>
Diskussionen	<ul style="list-style-type: none"> <li>• Wie sollte das Spiel aussehen? Theo macht eine Skizze, sendet Chantale das Mock up asap zu.</li> <li>• Arbeitsaufteilung</li> <li>• Use Cases werden zusammen erstellt, dabei werden die Beschreibungen und die Diagramme aufgeteilt.</li> <li>• Server/Client Protokoll wird von Theo übernommen.</li> </ul>
Meilensteine	<ul style="list-style-type: none"> <li>• Meilenstein 1 in Bearbeitung</li> <li>• Kanban Board wird erstellt mit Arbeitspaketen, damit die Meilensteine überwacht werden können. Dafür werden wir Trello nehmen.</li> </ul>
Next Steps	<ul style="list-style-type: none"> <li>• Kanban Board folgen nach Erstellung</li> <li>• Jeder schaut was zu reviewen ist, wenn der andere das Arbeitspaket zur Review schiebt.</li> <li>• Schauen das wir alles bis zum nächsten Meeting fertigstellen, damit wir bereit sind fürs</li> </ul>
Nächstes Meeting	<ul style="list-style-type: none"> <li>• 02.09.2021, Starbucks, Zürich Oerlikon Bahnhof</li> </ul>



### 3.1.3 Protokoll vom 04.09.2021

Teilnehmer	Theologos Baxevanos
	Chantale Gihara
Datum	04.09.2021
Ort	Welle 7, Bern
Informationen	<ul style="list-style-type: none"> <li>Projektstruktur wird angeschaut</li> <li>Erster Draft React erster Versuch wird angeschaut</li> <li>Protokolle und deren Schnittstellen wurde beschrieben</li> </ul>
Diskussionen	<ul style="list-style-type: none"> <li>Wie sollte das Spiel aussehen? Theo macht eine Skizze, sendet Chantale das Mock up asap zu.</li> <li>Arbeitsaufteilung</li> <li>Use Cases werden zusammen erstellt, dabei werden die Beschreibungen und die Diagramme aufgeteilt.</li> <li>Server/Client Protokoll wird von Theo übernommen.</li> </ul>
Meilensteine	<ul style="list-style-type: none"> <li>Meilenstein 2 in Bearbeitung</li> <li>Kanban Board wird erstellt mit Arbeitspaketen, damit die Meilensteine überwacht werden können. Dafür werden wir Trello nehmen.</li> <li>Meilenstein 1 wurde zeitgerecht eingereicht</li> </ul>
Next Steps	<ul style="list-style-type: none"> <li>Docker Container erstellen</li> <li>Registrierung und Login fertigstellen</li> <li>Chat Funktion erstellen</li> </ul>
Nächstes Meeting	<ul style="list-style-type: none"> <li>27.09.2021, Welle 7, Bern</li> </ul>

### 3.1.4 Protokoll vom 27.09.2021

Teilnehmer	Theologos Baxevanos
	Chantale Gihara
Datum	27.09.2021
Ort	Teams, online
Informationen	<ul style="list-style-type: none"> <li>Überprüfung was gemacht werden muss</li> <li>Stand Austausch</li> </ul>
Diskussionen	<ul style="list-style-type: none"> <li>Protokoll anschauen</li> <li>Use Case Diagramme entfernen und zusammenfassen</li> <li>Server/Client Protokoll wird von Theo übernommen.</li> <li>Registration abschliessen und mit Protokoll abgleichen</li> </ul>
Meilensteine	<ul style="list-style-type: none"> <li>Meilenstein 2 in Bearbeitung</li> <li>Abgabe vom 01. Oktober im Auge behalten</li> </ul>
Next Steps	<ul style="list-style-type: none"> <li>Server aufsetzen</li> <li>Client aufsetzen</li> <li>Docker fixieren</li> <li>Registrierung und Login fertigstellen</li> <li>Chat Funktion erstellen</li> </ul>
Nächstes Meeting	<ul style="list-style-type: none"> <li>30.09.2021, Teams online</li> </ul>

### 3.1.5 Protokoll vom 30.09.2021

Teilnehmer	Theologos Baxevanos
	Chantale Gihara
Datum	30.09.2021
Ort	Teams, online
Informationen	<ul style="list-style-type: none"> <li>Überprüfung Server, Client, Chat, Registration</li> <li>Protokoll Besprechung</li> <li>Stand Austausch</li> </ul>
Diskussionen	<ul style="list-style-type: none"> <li>Was wird abgeben und wer</li> <li>Wie funktionieren wir weiter</li> </ul>
Meilensteine	<ul style="list-style-type: none"> <li>Meilenstein 2 in Bearbeitung</li> <li>Meilenstein 2 Abgabe morgen 01.10.2021</li> </ul>
Next Steps	<ul style="list-style-type: none"> <li>Server verbessern</li> <li>Client verbessern</li> <li>Daten in DB speichern</li> <li>Spiel aufsetzen</li> </ul>



Nächstes Meeting	<ul style="list-style-type: none"> <li>08.10.2021, vor Ort PVA3 WebE =&gt; Welle7, Bern</li> </ul>	Mitglied der SUPS
------------------	--	-------------------

### 3.1.6 1.1.1 Protokoll vom 15.10.2021

Teilnehmer	Theologos Baxevanos
	Chantale Gihara
Datum	08.10.2021
Ort	Vor Ort FFHS, Welle 7 Bern
Informationen	<ul style="list-style-type: none"> <li>Docker, funktioniert noch nicht</li> <li>Cors Probleme auf der Frontend</li> <li>Protokoll Besprechung</li> <li>Stand Austausch</li> </ul>
Diskussionen	<ul style="list-style-type: none"> <li>Game wird erstellt als erstes Ping Pong aus Buch</li> <li>Server soll auf einen Port angepasst werden</li> <li>Cors Problem noch nicht gelöst</li> <li>React besteht schon in MVC</li> <li>MVC Dokumentation aufnehmen</li> </ul>
Meilensteine	<ul style="list-style-type: none"> <li>Meilenstein 3 in Bearbeitung</li> <li>Meilenstein 3 Abgabe 05.11.2021</li> </ul>
Next Steps	<ul style="list-style-type: none"> <li>MVC Dokumentation</li> <li>Protokolle ergänzen</li> <li>Game funktionsfähig</li> </ul>
Nächstes Meeting	<ul style="list-style-type: none"> <li>29.10.2021, Teams online</li> </ul>

### 3.1.7 1.1.2 Protokoll vom 29.10.2021

Teilnehmer	Theologos Baxevanos
	Chantale Gihara
Datum	05.11.2021
Ort	Teams, online
Informationen	<ul style="list-style-type: none"> <li>Cors Probleme bestehen immer noch</li> <li>Game erstellt</li> <li>Schwierigkeitsgrad erstellt</li> <li>Server auf 1 Port angepasst</li> </ul>
Diskussionen	<ul style="list-style-type: none"> <li>Game funktioniert aber Ball und Köpfe müssen noch angepasst werden</li> <li>ID ist momentan fix vergeben für das Game, das muss auch noch als UUID erstellt werden</li> <li>2 Players-Funktion erstellt</li> <li>Player gegen den Computer erstellt.</li> </ul>
Meilensteine	<ul style="list-style-type: none"> <li>Meilenstein 3 in Bearbeitung</li> <li>Meilenstein 3 Abgabe 05.11.2021</li> </ul>
Next Steps	<ul style="list-style-type: none"> <li>Ball =&gt; Corona skizzieren</li> <li>Menschen zum Treffen skizzieren.</li> <li>Barren möglich anpassen</li> <li>ID anpassen</li> <li>Dokumentation mit den neuen Infos anpassen</li> <li>Protokoll aufsetzen</li> </ul>
Nächstes Meeting	<ul style="list-style-type: none"> <li>06.11.2021, vor Ort PVA4 WebE =&gt; Welle7, Bern</li> </ul>

### 3.1.8 Protokoll vom 08.10.2021

Teilnehmer	Theologos Baxevanos
	Chantale Gihara
Datum	08.10.2021
Ort	Teams, online
Informationen	<ul style="list-style-type: none"> <li>Docker, Express, middleware</li> <li>Protokoll Besprechung</li> <li>Stand Austausch</li> </ul>
Diskussionen	<ul style="list-style-type: none"> <li>Was wird abgegeben und wer</li> <li>Game Create</li> <li>GameJoin</li> </ul>

	<ul style="list-style-type: none"> <li>• Canvas erstellen</li> <li>• Wie funktionieren wir weiter</li> </ul>
Meilensteine	<ul style="list-style-type: none"> <li>• Meilenstein 3 in Bearbeitung</li> <li>• Meilenstein 3 Abgabe 05.11.2021</li> </ul>
Next Steps	<ul style="list-style-type: none"> <li>• Spiel verbessern</li> <li>• Server anpassen</li> <li>• Client verbessern</li> <li>• Darstellung verbessern</li> </ul>
Nächstes Meeting	<ul style="list-style-type: none"> <li>• 15.10.2021, vor Ort PVA3 WebE =&gt; Welle7, Bern</li> </ul>

### 3.1.9 Protokoll vom 15.10.2021

Teilnehmer	Theologos Baxevanos
	Chantale Gihara
Datum	08.10.2021
Ort	Vor Ort FFHS, Welle7 Bern
Informationen	<ul style="list-style-type: none"> <li>• Docker, funktioniert noch nicht</li> <li>• Cors Probleme auf der Frontend</li> <li>• Protokoll Besprechung</li> <li>• Stand Austausch</li> </ul>
Diskussionen	<ul style="list-style-type: none"> <li>• Game wird erstellt als erstes Ping Pong aus Buch</li> <li>• Server soll auf einen Port angepasst werden</li> <li>• Cors Problem noch nicht gelöst</li> <li>• React besteht schon in MVC</li> <li>• MVC Dokumentation aufnehmen</li> </ul>
Meilensteine	<ul style="list-style-type: none"> <li>• Meilenstein 3 in Bearbeitung</li> <li>• Meilenstein 3 Abgabe 05.11.2021</li> </ul>
Next Steps	<ul style="list-style-type: none"> <li>• MVC Dokumentation</li> <li>• Protokolle ergänzen</li> <li>• Game funktionsfähig</li> </ul>
Nächstes Meeting	<ul style="list-style-type: none"> <li>• 29.10.2021, Teams online</li> </ul>

### 3.1.10 Protokoll vom 29.10.2021

Teilnehmer	Theologos Baxevanos
	Chantale Gihara
Datum	05.11.2021
Ort	Teams, online
Informationen	<ul style="list-style-type: none"> <li>• Cors Probleme bestehen immer noch</li> <li>• Game erstellt</li> <li>• Schwierigkeitsgrad erstellt</li> <li>• Server auf 1 Port angepasst</li> </ul>
Diskussionen	<ul style="list-style-type: none"> <li>• Game funktioniert aber Ball und Köpfe müssen noch angepasst werden</li> <li>• ID ist momentan fix vergeben für das Game, das muss auch noch als UUID erstellt werden</li> <li>• 2 Players-Funktion erstellt</li> <li>• Player gegen den Computer erstellt.</li> </ul>
Meilensteine	<ul style="list-style-type: none"> <li>• Meilenstein 3 in Bearbeitung</li> <li>• Meilenstein 3 Abgabe 05.11.2021</li> </ul>
Next Steps	<ul style="list-style-type: none"> <li>• Ball =&gt; Corona skizzieren</li> <li>• Menschen zum Treffen skizzieren.</li> <li>• Barren möglich anpassen</li> <li>• ID anpassen</li> <li>• Dokumentation mit den neuen Infos anpassen</li> <li>• Protokoll aufsetzen</li> </ul>
Nächstes Meeting	<ul style="list-style-type: none"> <li>• 06.11.2021, vor Ort PVA4 WebE =&gt; Welle7, Bern</li> </ul>

## Tabellen

Tabelle 1: FR - 001 .....	12
Tabelle 2: FR – 002.....	13
Tabelle 3: FR – 003.....	13
Tabelle 4: FR – 004.....	13
Tabelle 5: FR – 005.....	14
Tabelle 6: FR – 006.....	14
Tabelle 7: QR – 001 .....	15
Tabelle 8: QR - 002.....	15
Tabelle 9: QR - 003.....	16
Tabelle 10: Registrierung .....	17
Tabelle 11: Login.....	17
Tabelle 12: Create Game.....	17
Tabelle 13: Select Difficulty.....	18
Tabelle 14: Join Game .....	18
Tabelle 15: Spiel 1 .....	18
Tabelle 16: Spiel 2 .....	19
Tabelle 17: High Score.....	19
Tabelle 18: Typen Datenaustausch .....	25
Tabelle 19: Protokoll – Schnittstelle Registrierung .....	25
Tabelle 20: Protokoll – Schnittstelle Login .....	25
Tabelle 21: Protokoll – Schnittstelle Create Game .....	26
Tabelle 22: Protokoll – Schnittstelle Select Level .....	26
Tabelle 23: Protokoll – Schnittstelle Join Game .....	26
Tabelle 24: Protokoll – Schnittstelle Spiel 1 startet – Spieler1 / Server.....	26
Tabelle 25: Protokoll – Schnittstelle Spiel 1 startet – Server / Spieler2.....	27
Tabelle 26: Protokoll – Schnittstelle Spiel 1 startet – Spieler2 /Server.....	27
Tabelle 27: Protokoll – Schnittstelle Spiel 1 startet – Server / Spieler1.....	27
Tabelle 28: Protokoll – Schnittstelle Spiel 1 beendet – Server / Spieler1.....	27
Tabelle 29: Protokoll – Schnittstelle Spiel 1 beendet – Server / Spieler2.....	28
Tabelle 30: Protokoll – Schnittstelle Spiel 1 – Server / Spieler2.....	28
Tabelle 31: Protokoll – Schnittstelle Spiel 2 startet – Spieler1 / Server.....	28
Tabelle 32: Protokoll – Schnittstelle Spiel 2 startet – Server / Spieler2.....	29
Tabelle 33: Protokoll – Schnittstelle Spiel 2 startet – Spieler2 /Server.....	29
Tabelle 34: Protokoll – Schnittstelle Spiel 2 startet – Server / Spieler1.....	29
Tabelle 35: Protokoll – Schnittstelle Spiel 1 beendet – Server / Spieler1.....	29
Tabelle 36: Protokoll – Schnittstelle Spiel 1 beendet – Server / Spieler2.....	30
Tabelle 37: Protokoll – Schnittstelle High Score – Server / Client.....	30
Tabelle 38: Protokoll – Schnittstelle High Score – Server / Client.....	30
Tabelle 39: Protokoll – Schnittstelle Chat – Client/Server .....	30
Tabelle 40: Protokoll – Schnittstelle Chat – Server/Client .....	31

## Abbildungen

Abbildung 1: Skizze «Coronattack» Spiel.....	3
Abbildung 2:Projektstrukturplan .....	7
Abbildung 3: Arbeitsplan/Balkendiagramm - Meilenstein 1 .....	8
Abbildung 4: Arbeitsplan/Balkendiagramm - 2. Meilenstein .....	8
Abbildung 5:Arbeitsplan/Balkendiagramm - 3. Meilenstein .....	9
Abbildung 6: Arbeitsplan/Balkendiagramm - 4. Meilenstein .....	9
Abbildung 7: Use Case Diagramm 5 «Join Game» .....	18

## Quellen

- Makzan., 2015. HTML5 game development by example beginner's guide. Birmingham: Packt Pub.
- Creately. (n.d.). *Use Case Diagram Tutorial (Guide with examples)*. creately. Retrieved 08 25, 2021, from <https://creately.com/blog/diagrams/use-case-diagram-tutorial/>
- Dieter Kunz, N. S. (2018). Handbuch WBT Projektmanagement. In N. S. Dieter Kunz, *Handbuch WBT Projektmanagement* (Vol. 5. Auflage). ConPlus Gunten + Partner.
- Ludewig, J. u. (2013). Software Engineering. In *Software Engineering* (Vol. 3. Auflage). Horst.
- Martin, R. C. (2009). Clean Code. In R. C. Martin, *Clean Code*.
- Microsystems, S. (1999). Code conventions for the java programming language. In S. Microsystems, *Code conventions for the java programming language*. Sun Microsystems. Retrieved from <http://java.sun.com/docs/codeconv>
- Pohl, K. u. (2015). Basiswissen Requirements Engineering. In *Basiswissen Requirements Engineering* (Vol. 4. Auflage).
- Seidl, M. u. (2012). UML@Classroom. In M. u. Seidl, *UML @Classroom* (Vol. 1. Auflage).
- Starke, G. (2015). Effektive Software-Architekturen. In G. Starke, *Effektive Software-Architekturen* (Vol. 7. Auflage).
- Zörner, S. (n.d.). DokChess nach arc 42. In S. Zörner, *DokChess nach arc 42*. Retrieved 02 22, 2021, from <https://www.dokchess.de>