

Ausarbeitung für das Wahlpflichtmodul  
Software Quality Engineering  
an der Hochschule für Technik und Wirtschaft des Saarlandes  
im Studiengang Praktische Informatik  
der Fakultät für Ingenieurwissenschaften

**Ausarbeitung des Gruppenprojektes**

von

Alexander Stolz, Carolin Becker, Mohammed Al Ali, Nicolas Klein, Tristan Gläs

begutachtet von

Björn Scherer, Jessica Schiffmann

Saarbrücken, 23. Februar 2022



# Selbständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit (bei einer Gruppenarbeit: den entsprechend gekennzeichneten Anteil der Arbeit) selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich erkläre hiermit weiterhin, dass die vorgelegte Arbeit zuvor weder von mir noch von einer anderen Person an dieser oder einer anderen Hochschule eingereicht wurde.

Darüber hinaus ist mir bekannt, dass die Unrichtigkeit dieser Erklärung eine Benotung der Arbeit mit der Note „nicht ausreichend“ zur Folge hat und einen Ausschluss von der Erbringung weiterer Prüfungsleistungen zur Folge haben kann.

*Saarbrücken, 23. Februar 2022*

---

Alexander Stolz, Carolin  
Becker, Mohammed Al Ali,  
Nicolas Klein, Tristan Gläs



# Inhaltsverzeichnis

<b>1 Das Projekt</b>	<b>1</b>
1.1 Die Anwendung . . . . .	1
1.2 Struktur der Arbeit . . . . .	1
<b>2 Qualitätssicherungsplan</b>	<b>3</b>
<b>3 Testplan &amp; Testkonzept</b>	<b>5</b>
3.1 Teststrategie . . . . .	5
3.2 Testspezifikation . . . . .	7
3.3 Testdurchführung und Dokumentation . . . . .	7
3.4 Metriken . . . . .	8
3.4.1 Codemetriken . . . . .	8
3.4.2 Testmetriken . . . . .	9
3.4.3 Prozessmetriken . . . . .	10
3.5 Test-Stufen . . . . .	10
3.5.1 Eingangskriterien . . . . .	11
3.5.2 Testendekriterien . . . . .	11
3.5.3 Unit-Test . . . . .	11
3.5.4 System-Test . . . . .	11
3.5.5 Komponenten-Integrationstests . . . . .	12
3.5.6 System-Integrationstests . . . . .	12
3.6 Eingangskriterien . . . . .	12
3.7 Test-Ende-Kriterien . . . . .	12
3.8 Reviews . . . . .	12
3.9 Statische Analyse . . . . .	13
3.10 Unabhängigkeit . . . . .	13
3.11 Rollen . . . . .	14
3.12 Testschätzung . . . . .	14
3.13 Testfortschrittsüberwachung . . . . .	14
3.14 Testdatenmanagement . . . . .	15
3.15 Merge-Konzept . . . . .	15
3.15.1 Tooling . . . . .	16
3.15.2 Agile Prinzipien . . . . .	17
3.15.3 Issue-Tracking und Bug-Tracking . . . . .	18
3.15.4 Bewusste Defects . . . . .	18
3.16 Test-Arten . . . . .	18
3.16.1 Risikobasiertes Testen . . . . .	18
3.16.2 Automatisiertes Testen . . . . .	19
3.16.3 Sicherheits-Tests . . . . .	19
3.16.4 Regressions-Tests . . . . .	19
3.16.5 Last und Performance-Tests . . . . .	19
3.16.6 Usability Tests . . . . .	20
<b>4 Ausblick</b>	<b>23</b>

<b>Abbildungsverzeichnis</b>	<b>25</b>
<b>A Testspezifikationen</b>	<b>29</b>

# 1 Das Projekt

Als Teil der Prüfungsleistung des Wahlmoduls ‘Software Quality Engineering’ entstand diese Front- und Backend-Applikation. Der Schwerpunkt des Moduls liegt unter anderem in der Qualität von Software und Testen. Im Rahmen der zeitlichen Möglichkeiten war deshalb die Aufgabe der Gruppe, eine Applikation zu entwickeln, die möglichst viele Facetten des Testen gezielt und sinnvoll einsetzt.

Mit agilen Methoden und Werkzeugen in iterativen Entwicklungszyklen simulierte das Team so einen Teil eines realitätsnahem IT-Projekt. Die Komplexität eines realen IT-Projektes, samt Budgetierung, Ressourcenplanung, oder Personen wie Stakeholder, Projectowner, Scrummaster, dem Management, Kunden, etc. existierten in diesem Kontext nicht.

Sie nehmen aber starken Einfluss auf die Komplexität des späteren Systems. An geeigneten Stellen musste das Team daher Einfluss-Faktoren erfinden oder entwickelte Grundlagen und Konzepte für eine spätere Umsetzung.

## 1.1 Die Anwendung

Wie in Anlehnung an bekannte Cyber-Trinkspiele wie bspw. Picilo, basiert die Spielfunktionalität des Drinkgame auf einem Kartendeck, welches Aufgaben für die teilnehmenden Spieler bereithält.

Zu Beginn jeder neuen Spielpartie gibt der Spieldatenmaster (Endnutzer) die Namen der Spieler ein. Nach Bestätigung beginnt das Spiel. Von nun an wird randomisiert eine Karte mit Aufgabe(n) und ein Spielername (der diese Aufgabe erfüllen muss) dargestellt.

Die vorhandenen Aufgaben/Karten müssen zuvor erstellt werden. In einfacher Form gibt der Endnutzer einen Text ein und erstellt die Karte. Er hat auch die Möglichkeit, bestehende Karten aus seinem Kartendeck zu löschen.

Eine permanente Speicherung von eigenen Karten macht die Verwendung einer Authentifizierung notwendig. Eine Registrierung (mit Usernamen) und Login wurde umgesetzt. Die mangelnden Mittel und die im Team noch nicht ausgeprägte Expertise, ein Sicherheitssystem zu entwerfen, sind Grund für die Designentscheid des bestehenden trivialen Mechanismus. (An geeigneterer Stelle in dieser Ausarbeitung wird genauer auf die Designentscheidung eingegangen)

## 1.2 Struktur der Arbeit

Die Ausarbeitung strukturiert sich in zwei Teile. Zunächst wird der Qualitätssicherungsplan in Kapitel 2 betrachtet. In einem zweiten Schritt geht Kapitel 3 auf den Testplan und das Testkonzept ein.



## 2 Qualitätssicherungsplan

Macht ein Qualitätssicherungsplan für dieses Projekt Sinn? Ja, denn nach Abgabe wird das Projekt nicht weiterentwickelt. Das System hat zudem eine geringe Komplexität. Vor jedem Merge in den Master unterlag es allerdings bereits einem in schlanker Form bestehenden (Qualitätssicherungs-)plan.

Vor jedem Merge in den Master, muss ein Feature vollumfänglich getestet werden und den Integrationstest bestehen. Darüber hinaus testet und prüft ein zweiter Entwickler, der nicht mit dem Code vertraut ist, in einem Review das Feature.

Gesetzt den Fall, es würde weiterentwickelt werden, macht ein ausgeprägter QS-Plan in jedem Fall Sinn. Unterschiedliche Personen erachten Qualität aber als unterschiedlich. Man erstellt bei Projektbeginn eine Liste von Anforderungen und misst deren Einhaltungsgrad.

Das einzige Problem: Softwareentwicklung ist ein sehr komplizierter Prozess und viele Kunden haben ein mangelndes Verständnis davon. Kundenanforderungen haben geringeren Einfluss auf den Projekterfolg. Deshalb die Rede von Qualitätsattributen.

Eine Methode, um Qualitätsattributen und Metriken einzustufen, ist der Standard ISO/IEC 25010:2011. Letzterer wird in Grafik 2.1 visualisiert.

Der Kontext des Projektes erlaubt es dem Team nicht zu entscheiden, welche Qualitätsaspekte abgedeckt werden sollen. Das betrifft auch das Maß der Qualitätsaspekte, die man verbessern möchte.

## 2 Qualitätssicherungsplan

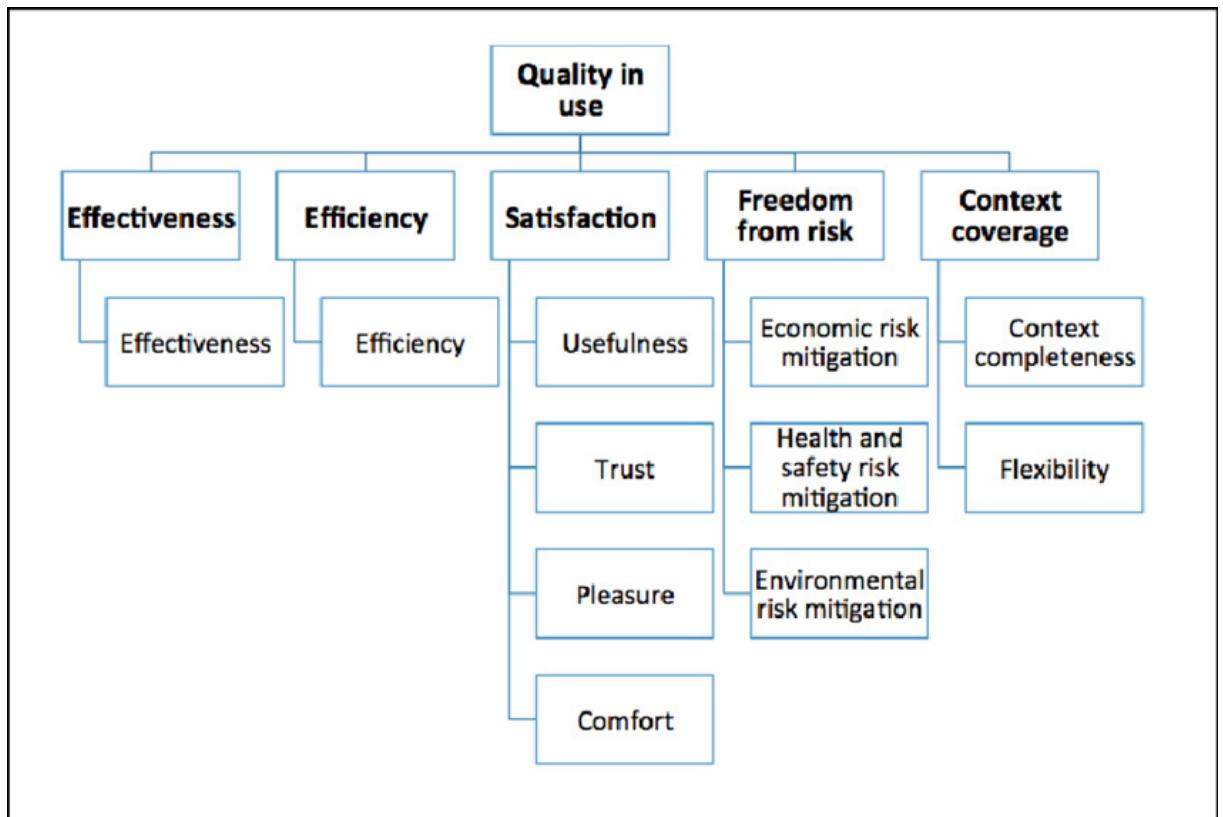


Abbildung 2.1: ISO/IEC 25010:2011

# 3 Testplan & Testkonzept

Nachdem wir im ersten Kapitel den Qualitätsplan betrachtet haben, gilt es nun auf den Testplan und die Teststrategie einzugehen. Hierbei wird darauf eingegangen, wie wir konkret in unserem Projekt vorgegangen sind.

Wichtige Aspekte sind das Vorgehensmodell, unser Dokumentations-Ansatz, die Teststufen und die Test-Ende-Kriterien. Außerdem wird darauf eingegangen, wie wir die Unabhängigkeit durch Reviews gewahrt haben. Darüber hinaus stellt auch unser Merge-Konzept auf Github ein wichtigen Punkt dar.

Das Kapitel schließt mit einem Ausblick auf mögliche Weiterentwicklungen und Verbesserungsansätze.

## 3.1 Teststrategie

Innerhalb unseres Projektes haben wir uns für das *Kanban Modell* entschieden. Dies hat für uns den Vorteil, dass der Fluss der Arbeit klar visualisiert wird. Dadurch werden Diskussionen vereinfacht und man vermeidet Missverständnisse. Darüber hinaus gibt das Board einen klaren Überblick über das Projekt.

Während des Projektes findet keine direkte Rollentrennung statt. Innerhalb eines Studenten-Projektes sind wir zeitlich klar begrenzt und der Lerneffekt soll für jedes Mitglied ähnlich sein. Dadurch ist jeder gleichermaßen für die Qualität des Projektes verantwortlich.

Zu Beginn des Projektes wurde mittels eines UseCase Diagramms eine Menge von abzudeckenden UseCases aufgestellt. Diesen UseCases wurde anschließend jeweils eine Kurzbeschreibung der erwarteten Funktionalität zugeordnet. Die erarbeiteten UseCases werden in der Grafik 3.1 verdeutlicht.

Um die UseCases in das Kanban Board einzuarbeiten, wurden diese aufgespalten. Als Maßstab galt die Maßeinheit Arbeitstage. Ein Ticket sollte in einem Aufwand von etwa 2 *Arbeitstagen* liegen. Diese Aufspaltung wird im Kapitel Testschätzung ausgeführt.

Das Kanban Board haben wir wie folgt aufgebaut. Am linken Rand des Boards befindet sich eine *Not-Started Spalte*. Diese kann als Backlog aufgefasst werden. Pro Sprint nimmt sich je ein Team-Mitglied ein Element aus diesem Backlog heraus, um es im Sprint zu bearbeiten.

Anschließend wandert das Ticket zu der *Konzeptions-Spalte*. In dieser Phase werden die Tests konzeptioniert. Hierfür existiert eine Vorlage, welche im Kapitel Test-Dokumentation erläutert wird. Falls das Ticket dem Frontend zugeordnet ist, wird neben den Tests auch ein grobes Mockup der Benutzeroberfläche entworfen. Abschließend findet pro Ticket ein Review der Konzeption statt. Der Review Prozess wird im Kapitel Reviews behandelt.

In der nächsten Phase, der *Implementierungs-Phase*, wird die Funktionalität des Tickets implementiert. Die konkreten Tests werden hier noch nicht verfasst.

### 3 Testplan & Testkonzept

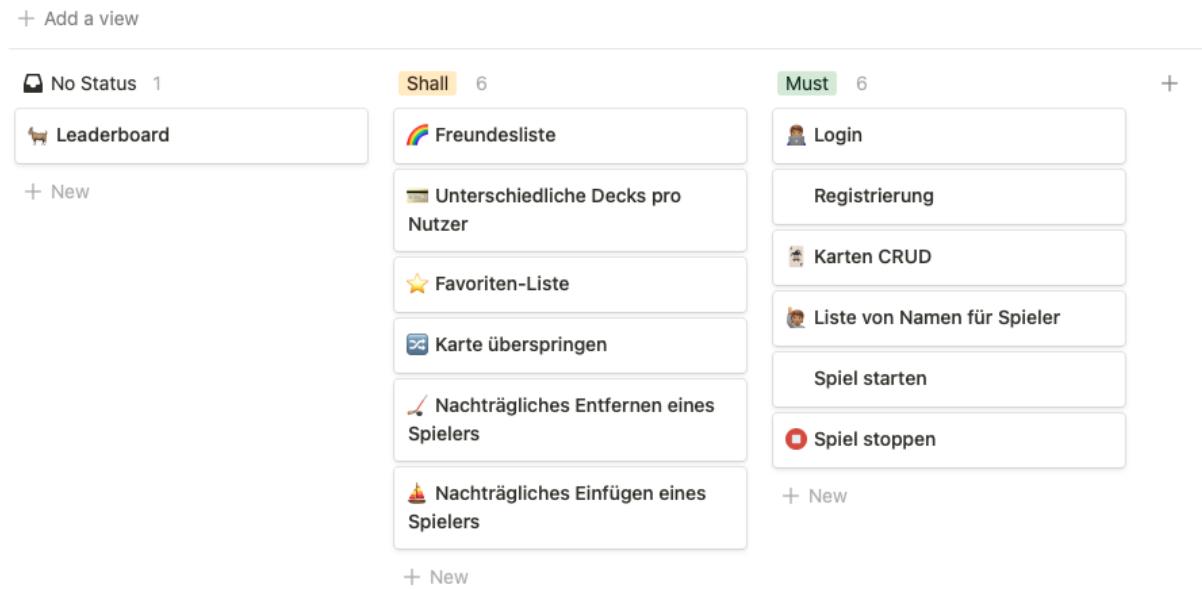


Abbildung 3.1: Die von uns initial erarbeitete Use Caseses

Ist die Funktionalität implementiert, wandert sie in die *Test-Phase*. In dieser werden die zuvor konzeptionierten Tests umgesetzt. All diese Tests müssen umgesetzt und grün sein. Die Unabhängigkeit wird dabei gewahrt, obwohl derselbe Entwickler Tests konzeptuiert und implementiert. Hierfür sorgt der Review Prozess und die zeitliche Trennung von Konzeption und Implementierung der Tests. Auf die Unabhängigkeit wird ebenfalls im gleichnamigen Kapitel eingegangen.

Sind alle zuvor konzeptionierten Tests umgesetzt und grün, wird nach dem Merge-Konzept vorgegangen, um die Funktionalität in den Master zu integrieren. Dieses wird ebenfalls in einem eigenen Kapitel behandelt.

Ist die Funktionalität im Master integriert, und alle Tests sind weiterhin grün, wandert das Ticket in die *Done Spalte* und ist abgearbeitet. Während dieses Vorgangs wird stets sichergestellt, dass sich das Projekt in einem funktionierenden Zustand befindet und inkrementell um Funktionalitäten erweitert wird. Das Board wird in Grafik 3.2 abgebildet. Zur Verwaltung des Boards haben wir das Tool *Notion* genutzt.

Da unser Projekt zeitlich klar terminiert ist, finden End-2-End Tests gesondert am Ende des Projektes statt. Hierdurch vermeiden wir ein regelmäßiges Abändern der e2e-Tests und sparen damit Zeit und Ressourcen.

Der fundamentale Testprozess ist in unserem konkreten Kanban-Vorgehen integriert. Die Testplanung findet dabei zu Beginn des Projektes statt. Hier werden die UseCases in Tickets aufgeteilt und letztere den Personen zugeordnet. Die Testanalyse und das Design haben wir aufgespalten und auf jedes Ticket verteilt. Hierdurch kann sich gesondert auf eine Funktionalität konzentriert werden. Darüber hinaus werden die Tests mittels eines Review-Prozesses validiert. Die Testrealisierung und Auswertung findet dabei erst nach der eigentlichen Implementierung statt. Ein Testabschluss und die Auswertung kann entweder zum Ende jedes Tickets stattfinden, oder global am Ende des gesamten Projektes.

## 3.2 Testspezifikation

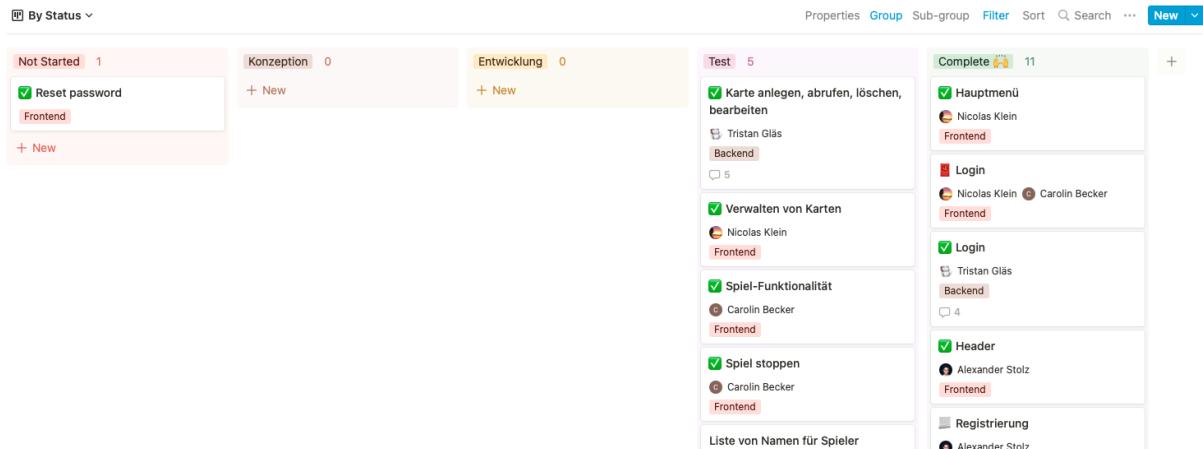


Abbildung 3.2: Ein Snapshot unseres Kanban-Boards

Unser Test-Strategie kann als *Service-First Ansatz* verstanden werden. Wir definieren die Tests vor der eigentlichen Implementierung. Allerdings werden die Tests erst nach der Implementierung der fachlichen Funktionalität im Code umgesetzt. Bedingt durch die fehlende Erfahrung mit Angular und Nest, sowie den zeitlichen Rahmenbedingungen, haben wir uns gegen Test-Driven Development entschieden.

## 3.2 Testspezifikation

Zu Beginn des Projektes haben wir gemeinsam im Team eine Liste von UseCases ermittelt. Diese finden sie im Anhang. Aus diesen UseCases heraus haben wir Tickets erarbeitet, welche anschließend im Kanban Board organisiert wurden.

Die für das jeweilige Ticket zugehörigen Tests wurden in der Konzeptionsphase spezifiziert. Anschließend wurden diese, pro Ticket, mit jeweils einer weiteren Person reviewed. Die dadurch entstandenen Testspezifikationen finden sie im Anhang.

## 3.3 Testdurchführung und Dokumentation

Zur Dokumentation der Tests wurde zu Beginn des Projektes eine Vorlage definiert. Für die Verwaltung des Teams nutzten wir das Tool [Notion](<http://notion.so>). Dieses bietet uns die Möglichkeit Vorlagen zu erstellen und an den passenden Stellen automatisch anwenden zu lassen.

In unserem Fall können wir jedes Ticket des Kanban-Boards öffnen. Anschließend erhalten wir eine leere Vorlage zum Erfassen von Testfällen für das jeweilige Ticket.

Dies bietet uns die Möglichkeit, die Dokumentation und unser Kanban-Board zusammenzuführen. Bei einer Teamgröße von unter 10 Personen bietet dies organisatorische Vorteile. Es kann Zeit gespart und die Übersicht bewahrt werden.

Die von uns für jedes Ticket genutzte Vorlage finden sie unterhalb dieses Abschnitts in Tabelle 3.1. Dabei wird die Tabelle pro Testfall dupliziert. Platz für eine allgemeine Ticket Beschreibung und das Mockup der zugehörigen Oberfläche finden sich ebenfalls in der Vorlage.

Wir haben ebenfalls eine Vorlage zur Fehlererfassung eingeführt. Diese wird in Tabelle 3.2 gezeigt. Im Umfang unserer Projektes hat sich diese allerdings nicht als notwendig herausgestellt. Kleinere Fehler wurden stets in der Test-Phase gefunden und behoben. Sobald das Ziel erreicht wurde, dass alle Tests grün sind, wurden auch keine umfangreichen

### 3 Testplan & Testkonzept

<b>Testfall (Nr.)</b>	Frontend   Backend-[Name der Komponente/Modul]-[Testnummer] (z.B. Frontend-Login-2)
<b>Name, Zusammenfassung</b>	
<b>Setup, Voraussetzungen</b>	
<b>Testdaten</b>	
<b>Testschritte</b>	
<b>Erwartetes Ergebnis</b>	
<b>Automatisiert</b>	Ja, halb-automatisiert oder manuell

Tabelle 3.1: Die Testspezifikations-Vorlage

Fehler mehr gefunden.

<b>Fehler-ID</b>	Error-Frontend   Backend-[Fehler-Nummer] (z.B. Error-Frontend-2)
<b>Test-ID</b>	
<b>Zusammenfassung</b>	
<b>Fehlerklasse</b>	
<b>Verwendete Testdaten</b>	
<b>Reproduzierbarkeit</b>	
<b>Ablaufbeschreibung</b>	

Tabelle 3.2: Die Fehlerspezifikations-Vorlage

## 3.4 Metriken

In diesem Abschnitt werden Metriken des Projektes vorgestellt. Diese sind in Produktmetriken und Prozessmetriken gegliedert. Aufgrund mangelnder Erfahrung wurden die Metriken erst am Ende des Projektes aufgestellt. Daher gibt es keine Vergleichswerte mit Zwischenständen und die Darstellung der Metriken von Backend und Frontend erfolgt am Ende des Projektes.

Auf Grund der überschaubaren Anzahl an Projektanforderungen, wurde von Spezifikationsmetriken wie z.B. Function Points abgesehen. Die einzelnen Arbeitspakete wurden so klein gehalten, dass ein Ticket einen Aufwand von ca. zwei Arbeitstagen hatte.

Im nächsten Abschnitt werden zuerst die verwendeten Produktmetriken beschrieben.

### 3.4.1 Codemetriken

Um die Lines of Code (LOC) überwachen zu können, wurde das Plugin „Statistic“ in IntelliJ verwendet. Mit diesem Plugin war es möglich, die Lines of Code der einzelnen Klassen zu überwachen. Außerdem ergab sich somit auch einen Überblick über den Anteil der Kommentare und der Leerzeilen pro Klasse. Mit dem Plugin ist eine gute Übersicht über die Anzahl der Zeilen je Klasse zu sehen. Falls die Anzahl der LOC zu groß ist, ist dies ein Hinweis darauf, dass diejenige Klasse besser in zwei Klassen unterteilt werden sollte.

Die Übersicht der Kommentare ermöglicht eine Überprüfung, ob der Code ausreichend kommentiert ist.

Beispielsweise ist im Backend zu sehen, dass eine TypeScript-Klasse aus nicht mehr als 64 Code Zeilen besteht. Dies weist darauf hin, dass die Anzahl an Klassen gut gewählt wurde, sodass keine zu großen Klassen existieren.

Auch im Frontend ist eine solche Überwachung gelungen. Hier besitzt keine TypeScript-Klasse mehr als 89 Code Zeilen, sodass die Klassen ebenfalls nicht zu groß ausfallen.

Allgemein ist das Überwachen der Lines of Code in größeren Projekten hilfreicher, da die Anzahl an verschiedenen Klassen dort größer und unübersichtlicher ist. Dies könnte mit dem Plugin gut überwacht werden.

### 3.4.2 Testmetriken

Nachfolgend sind die Testabdeckungen vom Frontend sowie Backend zu sehen. Diese werden im Server mit „npm run test:cov“ und im Client mit „ng test –code-coverage“ ausgegeben. Allgemein sind die Testabdeckungen für Backend und Frontend in untenstehender Tabelle zu sehen.

Bei genauerer Betrachtung der Testabdeckungen ist im Backend das Ergebnis in Tabelle 3.4 zu sehen.

	<b>Backend</b>	<b>Frontend</b>
<b>Statements</b>	10.90%	78.03%
<b>Branches</b>	59.09%	61.53%
<b>Functions</b>	8.57%	70.27%

Tabelle 3.3: Testmetriken 1

<b>Backend</b>	<b>Statements</b>
<b>cards-logic.ts</b>	100%
<b>register-user.dto.ts</b>	100%
<b>user-logic.ts</b>	100%

Tabelle 3.4: Testmetriken 2

Hier wird deutlich, dass die Anweisungsüberdeckung bei den verschiedenen Klassen jeweils bei 100 Prozent liegt. Bei den nicht dargestellten Klassen liegt die Anweisungsüberdeckung bei unter 100 Prozent, denn diese wurden z.B. mit Postman getestet wodurch eine Dokumentation, wie oben dargestellt, nicht möglich war.

Dies ist auch unter anderem der Grund, warum die Werte der allgemeinen Tabelle im Backend wie z.B. die Anweisungsüberdeckung mit 10,09 Prozent gering sind. Ein weiterer Grund dafür ist, dass in die Statistik generierter Code, Services und auch Controller mit einberechnet sind, die aber nicht von den Unit Tests abgedeckt werden.

In der Tabelle 3.5 ist ein Auszug aus dem Frontend zu sehen.

Im Frontend liegt die Anweisungsüberdeckung ebenfalls nicht bei 100 Prozent, da nicht nur automatisierte Tests durchgeführt wurden. Stattdessen wurden auch manuelle Tests verwendet.

### 3 Testplan & Testkonzept

Frontend	Statements
src/app/modules/auth/register	85.71%
src/app/modules/auth/login	88.46%
src/app/core/main_menu/menu	77.77%
src/app/core/gameFunctionality/gameSettings	75.47%
src/app/core/gameFunctionality/game	67.74%

Tabelle 3.5: Testmetriken 3

Die Branch Coverage, auch Zweigabdeckung/Pathüberdeckung genannt, entspricht dem Verhältnis der beim Test durchlaufener Zweige bzw. Branches zu der Gesamtanzahl der Zweige. Dieses beträgt im Backend 59.09 Prozent und im Frontend 62.5 Prozent.

Die Function Coverage berechnet sich aus der Anzahl an Funktionen, die aufgerufen werden. Diese beträgt im Backend 8.57 Prozent und im Frontend 57.14 Prozent.

#### 3.4.3 Prozessmetriken

Wie bereits erwähnt, wurde allgemein darauf geachtet, dass pro Anforderung bzw. pro Ticket ein Aufwand von etwa zwei Arbeitstagen festgelegt wurde. Dadurch konnten die Tickets zügig bearbeitet werden und es kam nicht zu großen Verzögerungen auf Grund von aufwändigen Tickets, die den Arbeitsfluss ins Stocken bringen würden.

Bei Auftreten von Fehlern z.B. bei Reviews oder bei den abschließenden Tests wurden die Fehler möglichst schnell wieder behoben. Diese Zeit betrug maximal ein bis zwei Stunden nach Feststellung der Fehler, sodass es eine stetige Weiterentwicklung gab und keine Verzögerungen auf Grund der identifizierten Fehler auftraten.

Durch die verschiedenen Metriken werden Risikofaktoren wie z.B. eine zu hohe Bearbeitungszeit einzelner Tickets minimiert.

Durch die LOC – Metrik wird überwacht, ob das Verhältnis zwischen Code und Kommentaren sowie die Anzahl der Code Zeilen in einer Klasse angemessen ist. Dadurch werden Risiken wie eine zu große Klasse verhindert, die besser in zwei Klassen aufgeteilt werden sollte.

Durch die Testüberdeckungen wird eine anhaltende Funktionalität gewährleistet. Beispielsweise wird mit Integrationstests die Funktionalität im ganzen Projekt nochmals überprüft und dadurch das Fehlerrisiko minimiert.

## 3.5 Test-Stufen

Unser Ansatz für die Teststufen war eine Mischung aus *TopDown* und *Big Bang*. Hierbei ist der Umfang und das Zeitfenster des Projektes zu beachten. Die meisten der gebauten Komponenten ließen sich klar voneinander abtrennen. Sowohl im Frontend, als auch im Backend, konnte parallel und inkrementell vorgegangen werden.

Konkret heißt das im Falle des Frontends, dass die Services zunächst gemocked wurden. Vergleichbar mit einem *TopDown* Ansatz wurden Platzhalter für die Services verwendet, und die UI erstellt. Allerdings wurde pro Ticket jeweils eine gesamte Funktionalität realisiert. Beispielsweise wurde der Login für ein Ticket realisiert. Hier wurde zunächst der

AccountService gemocked und die dazu nötige Benutzeroberfläche gebaut. Im nächsten Schritt wurde anschließend der Mock durch die eigentliche Implementierung des AccountService ersetzt. Erst zu diesem Zeitpunkt war das Login Ticket abgeschlossen. Es hat also keine Spaltung in verschiedene Schritte oder Arbeitspakete stattgefunden. Daher lässt sich das Vorgehen auch mit einem BigBang vergleichen.

### 3.5.1 Eingangskriterien

Die Eingangs-Kriterien ergeben sich aus unserem Vorgehensmodell. Dabei haben Unit-Tests als Anforderung, dass zuvor deren Konzeption verfasst und die zugehörige Funktionalität implementiert wurde.

Komponenten-Integrationstest können durchgeführt werden, sobald die Unit-Tests der Funktionalität erfolgreich durchlaufen wurden. Und erst, wenn alle geplanten Tickets sich in der Done-Phase befinden, werden die e2e-Tests verfasst und durchlaufen.

### 3.5.2 Testendekriterien

Vor dem Start der Implementierung wurden für das Projekt „Test-Ende Kriterien“ definiert.

Ein Kriterium ist, dass alle Kern-Tests für die funktionalen Anforderungen innerhalb eines Branches erfolgreich durchlaufen und somit auf „grün“ stehen. Dadurch kann sichergestellt werden, dass alle fachlichen Anforderungen an das Testobjekt vollständig und korrekt umgesetzt wurden.

Aufgrund des überschaubaren Projektes wurde beschlossen, dass die Fehlerrate beim Testen gleich 0 sein sollte. Um dieses Kriterium zu überprüfen, wurden beispielsweise Fehleingaben getestet.

Bei Auftreten von Mängeln oder Bugs wie z.B., dass vorher definierte Erwartungen oder Anforderungen nicht erfüllt sind, muss der Code bzw. die Funktionalität erneut bearbeitet werden. Nachdem die Mängel beseitigt wurden, wurde das Testen erneut durchgeführt.

Nachdem alle Testkriterien erfolgreich waren, wurde der Branch mit den neuen Funktionalitäten in den Master-Branch gegerget. Danach wurde das System erneut getestet. Dieser Test wurde nach Erfüllung der Test-Ende Kriterien beendet. Die Funktionalität des Merge-Konzeptes wird später noch genauer erläutert.

### 3.5.3 Unit-Test

Als unterste Teststufe haben wir Unit-Tests im Frontend, bzw. die Tests mit Postmen im Backend umgesetzt. Für die Tests im Frontend wurden Tests mit Jasmin und Chai bzw. Mokka definiert. Die Requests nach außen hin wurden gemocked, um eine Unabhängigkeit zum Backend zu erhalten.

### 3.5.4 System-Test

System-Tests wurden von uns nicht umgesetzt. Im Verlauf des Projektes waren diese bezüglich Zeit und Umfang für uns nicht rentabel. Allerdings hätte man diese mit dem Tool json-Server umsetzen können. Dieser Server erstellt eine REST-API basierend auf einer Json-Datei und stellt statische Informationen zur Verfügung. Dadurch hätten wir

### 3 Testplan & Testkonzept

die Anfragen des Frontends nach außen testen können, ohne auf das eigentliche Backend zuzugreifen.

#### 3.5.5 Komponenten-Integrationstests

Komponenten-Integrationstests haben wir durchgeführt, indem wir das Frontend mit dem tatsächlichen Backend erneut getestet haben. Hier wurden die zugehörigen Services und deren Anfragen nicht mehr nach außen gemocked.

#### 3.5.6 System-Integrationstests

Abschließend wurden System-Integrationstests bzw. End-2-End Tests durchgeführt. Bei diesem wurde direkt auf die UI-Zugriffe und die gesamten Abläufe der Anwendung durchgespielt. Um False Negatives zu vermeiden, wurden Custom-ids genutzt, um die Elemente der DOM zu adressieren.

### 3.6 Eingangskriterien

Die Eingangs-Kriterien ergeben sich aus unserem Vorgehensmodell. Dabei haben Unit-Tests als Anforderung, dass zuvor deren Konzeption verfasst und die zugehörige Funktionalität implementiert wurde. Komponenten-Integrationstest können durchgeführt werden, sobald die Unit-Tests der Funktionalität erfolgreich durchlaufen wurden.

Und erst, wenn alle geplanten Tickets sich in der Done-Phase befinden, werden die e2e-Tests verfasst und durchlaufen.

### 3.7 Test-Ende-Kriterien

Vor dem Start der Implementierung wurden für das Projekt *Test-Ende Kriterien* definiert. Ein Kriterium ist, dass alle Kern-Tests für die funktionalen Anforderungen innerhalb eines Branches erfolgreich durchlaufen und somit auf „grün“ stehen. Dadurch kann sichergestellt werden, dass alle fachlichen Anforderungen an das Testobjekt vollständig und korrekt umgesetzt wurden.

Aufgrund des überschaubaren Projektes wurde beschlossen, dass die Fehlerrate beim Testen gleich null sein sollte. Um dieses Kriterium zu überprüfen, wurden beispielsweise Fehleingaben getestet.

Bei Auftreten von Mängeln oder Bugs wie z.B., dass vorher definierte Erwartungen oder Anforderungen nicht erfüllt sind, muss der Code bzw. die Funktionalität erneut bearbeitet werden. Nachdem die Mängel beseitigt wurden, wurde das Testen erneut durchgeführt. Nachdem alle Testkriterien erfolgreich abgeschlossen waren, wurde der Branch mit den neuen Funktionalitäten in den Master-Branch gemerged. Danach wurde das System erneut getestet. Dieser Test wurde nach Erfüllung der Test-Ende Kriterien beendet. Die Funktionalität des Merge-Konzeptes wird später noch genauer erläutert.

### 3.8 Reviews

In unserem Projekt haben wir Reviews eingesetzt um die Testspezifikation zu überprüfen. In dem von uns eingesetzten Kanban Modell läuft jedes Ticket durch eine Test-Konzeptions-Phase. In dieser werden die Tests für die Komponente festgelegt. Wir sehen

diese Phase als kritisch an. Denn hier müssen bereits alle benötigten Tests definiert werden. Sollte hier ein Testfall falsch spezifiziert sein, oder fehlen, wirkt sich das direkt auf die nachfolgenden Phasen aus.

Unser Ziel war es, durch Reviews fehlerhafte oder unvollständig definierte Testfälle zu finden und gemeinsam daraus zu lernen.

Als Review Art haben wir uns für eine Stellungnahme entschieden. Für unser Projekt galt es, die zeitliche Komponente im Fokus zu halten. Ein schnelles, informelles Feedback ist ausreichend. Hierdurch kann ein Lerneffekt erzielt werden, ohne zu viel Zeit in die Reviews zu investieren. Das Feedback ist intern. Zwischen zwei Personen im Frontend oder Backend Bereich.

Das Review-Team besteht dabei aus folgenden Personen:

- Die für das Ticket zuständigen Person ist der Autor
- Einer weiteren Person aus dem jeweiligen Team (Frontend oder Backend) ist der Gutachter

Im Folgenden werden die Phasen unserer Reviews aufgestellt und erläutert:

#### Planung

In der Planungsphase wird ermittelt, welche konkreten Testfälle ein Review benötigen. Sollte konkret ein fehlender oder falscher Testfall aufgefallen sein, ist dieser hier aufzuarbeiten.

#### Reviewsitzung

Start des eigentlichen Reviews. Hier werden gemeinsam die betreffenden Testfälle erarbeitet und eine verbesserte Konzeption abgeleitet.

#### Überarbeitung

...

## 3.9 Statische Analyse

Für die statische Code-Analyse gibt uns TypeScript eine grundfunktionalität mit ESLint. Letzteres kann bereits Endlosschleifen erkennen und diese hervorheben. Auch schlechter Stile, wie zum Beispiel unnötige Leerzeichen und fehlende Absätze nach dem Methodenname, werden erkannt. Sollte eine Variable mit var oder let definiert sein, obwohl diese auch den const Identifier nutzen könnte, wird darauf ebenfalls hingewiesen. Toter-Code kann ebenfalls durch ESLint gefunden werden.

Da die fachliche Komplexität unserer Anwendung sich im unteren Mittelfeld befindet, wurde keine weiteren statischen Analysen genutzt. In einem späteren Abschnitt wird auf die genutzten Metriken eingegangen.

- ESLint
- TypeScript

## 3.10 Unabhängigkeit

Wir stellen die Unabhängigkeit dadurch sicher, dass wir die Testkonzeption und die Testimplementierung voneinander trennen. Die Testkonzeption beschränkt sich auf die Definition der Tests. Unabhängig von der konkreten Implementierung.

### **3 Testplan & Testkonzept**

Zusätzlich wird die Testfall-Konzeption jedes Tickets durch ein Review geprüft. Hierdurch haben zwei Personen die Testfälle überprüft.

Sollte letztlich dieselbe Person, welche auch die Testfälle ursprünglich verfasst hat, diese Implementieren, ist trotzdem eine Unabhängigkeit gewahrt. Denn die Definition hat bereits stattgefunden und wurde von mindestens einer anderen Person geprüft.

Dadurch wird eine Personenunabhängigkeit erreicht.

#### **3.11 Rollen**

Während unserer Projektes haben alle Mitglieder die Rolle des Tester:innen und Testmanager:innen eingenommen. Bedingt durch den Umfang und die zeitlichen Rahmenbedingungen war es nicht zielführend einzelne Personen zu den jeweiligen Rollen zuzuordnen. Auch sollte der Lerneffekt dadurch gesteigert werden, dass jedes Mitglied jede der Rollen kennengelernt hat.

Explizite Testdatenmanager:innen oder Defektmanager:innen wurden ebenfalls nicht festgelegt.

#### **3.12 Testschätzung**

Zentral für unser Projekt war die Spaltung der einzelnen Tickets in der ursprünglichen Planungsphase. Unser Ziel war es, jedem der Tickets den etwa selben Aufwand zuzuordnen. Dabei sollte die Implementierung in etwa einem Arbeitstag umsetzbar sein. Zusammen mit der Testfallkonzeption und der Testimplementierung sollten nicht mehr als zwei Arbeitstage benötigt werden, um ein Ticket über das gesamte Board zu bewegen.

Dieses Vorgehen könnte als Einzelschätzung betrachtet werden. Denn sie war schnell und in formal. Ferner sind wir uns der hohen Subjektivität und großen Schätzungsunsicherheit bewusst.

#### **3.13 Testfortschrittsüberwachung**

Bedingt durch den Aufbau unseres Kanban-Modells ergibt sich ein klarer Ablauf für die Tests. Noch vor der Umsetzung des Tickets in der Implementierung werden die Tests konzeptioniert. Erst nach der Implementierung werden die Tests umgesetzt und ausgeführt.

Als Testfortschrittsüberwachung gilt für uns, wie viele der konzeptionierten Tests umgesetzt wurden und erfolgreich sind.

Ein Vergleich anhand von expliziteren Metriken erscheint uns wenig sinnvoll. Denn die Testkosten sind für nahezu jedes Modul identisch. Während der Testschätzung wurden die Tickets so formuliert, dass jedes den etwa gleichen Umfang von 2 Personen-Tagen aufweist.

Auch Metriken wie Datenbanknutzung und Schnittstellenbreite führen zu keinen wertvollen Erkenntnissen. Jedes Ticket im Frontend setzt eine Kommunikation mit dem Backend

um. Genauso setzt jedes Ticket des Backends eine Kommunikation mit der Datenbank um. Auch die Testpfadkomplexität nährt sich in jedem Ticket einem Mittelwert an. Die Testbarkeit jedes Tickets ist folglich sehr ähnlich. Die benutzen Testmetriken werden im Kapitel 3.4 beschrieben.

## 3.14 Testdatenmanagement

Alle Testdaten werden mit Hilfe von Skripten vor den Testausführungen in die Datenbank geschrieben. Die eigentlichen Testdaten liegen in SQL Befehlen vor, wobei jeder Befehl mit einem Kommentar zu dem entsprechenden Testfall verlinkt ist. Die Skripte sind alle in Git eingepflegt.

## 3.15 Merge-Konzept

Wir nutzen zur Verwaltung unseres Codes Git. Den Code haben wir auf Github gehostet. Eine ausführliche Diskussion über die Vor- und Nachteile von Git im Vergleich zu zentralisierten Quellcode-Kontrollsystmen finden sich im Internet. Für uns war das Ziel, die Arbeit möglichst gut parallelisieren zu können. Jedes Teammitglied sollte unabhängig von den anderen Entwickeln und Testen können. Dies spiegelt sich auch in unserem Merge-Konzept wider. Eine grafische Darstellung finden sie unter diesem Abschnitt.

Das Repository-Setup, das wir verwenden und das gut mit diesem Verzweigungsmodell funktioniert, ist das mit einer zwei zentralen Bausteinen. Wir entschieden uns das Frontend und Backend klar voneinander zu trennen. Dies wird uns bei der Testautomatisierung helfen. Letzteres wird später erläutert.

Der Master ist dabei die zentrale Wahrheit jedes Repositorys. Der Master ist stets in einem funktionierenden Zustand und wird inkrementell um Features erweitert. Er ist dabei immer production-ready.

Als unterstützende Branches führen wir Feature Branches ein. Diese setzen jeweils eines der von uns im Kanban Board definierten Tickets um. Sobald das Ticket die Implementierung und Test Phasen des Kanban Boards durchlaufen hat, nehmen wir an, dass folgende Schritte in diesem Feature Branch durchlaufen wurden:

- Das Feature wurde vollständig implementiert.
- Es wurden alle zuvor konzeptionierten Tests umgesetzt.
- Diese Tests laufen alle ohne Fehler durch.
- Der Master wurde in den Feature-Branch gemerged, und alle Tests laufen weiterhin erfolgreich durch
- Das Feature wurde in den Master gemerged und der Feature-Branch entfernt.

Feature-Banches haben eine begrenzte Lebenszeit und werden gelöscht, sobald das zugehörige Ticket abgeschlossen ist.

Der Vorteil dieses Vorgehens ist, dass die Person auf ihrem lokalen System in einem eigenen Branch arbeiten kann. Dadurch erhält sie schon in ihrem lokalen Branch eine

### 3 Testplan & Testkonzept

Rückmeldung über die Tests. Wird beispielsweise der Master in den Feature-Branch gemerged, können danach die Tests neu ausgeführt werden. Sollten sich jetzt Probleme ergeben, hat noch keine Änderung am Master stattgefunden.

Von einem Prinzip mit Pull-Requests haben wir für das Projekt abgesehen. Dies geht daraus hervor, dass wir auch keine Issues auf Github verwalten. Das Issue Board wird durch unser Kanban-Board auf Notion umgesetzt. Und da sich die Menge an Tickets nicht dynamisch verändert, hatte eine Verwaltung von Issues für uns keine Vorteile.

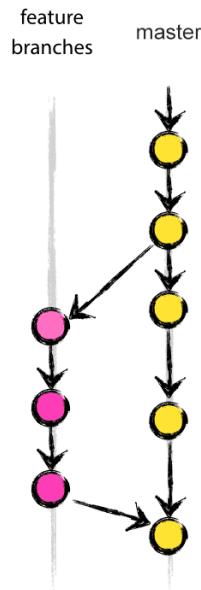


Abbildung 3.3: Ein beispielhafter Feature-Branch und der Master-Branch

#### 3.15.1 Tooling

Als Test-Umgebung verwenden wir das lokale System des jeweiligen Entwicklers und Github Actions. Github-Actions bieten uns die Möglichkeit das Frontend auf einem Github Server zu bauen, auszuführen und zu testen. Dies hat den Vorteil, dass man auch abseits des lokalen Systems eine Rückmeldung über den Status des Projektes erhält.

Dabei haben wir Github mittels der Datei '.github/workflows/main.yaml' im Frontend Repository so konfiguriert, dass die Anwendung automatisch gebaut und getestet wird. Github nutzt hierfür ein Ubuntu System, auf welchem Node 14 und die benötigten Pakete installiert werden. Anschließend wird eine spezielle Version von Chrome namens \*ChromeHeadlessCI\* ausgeführt, welche ohne eine visuelle UI auskommt.

Für Unit-Tests bietet Angular von Haus aus solide Testwerkzeuge. Wenn ein Angular Projekt über die Befehlszeilenschnittstelle erstellt wird, ist es mit einem voll funktionsfähigen Test-Setup für Unit-Tests ausgestattet.

Das Angular-Team hat bereits die Entscheidung getroffen Jasmine als Test-Framework und \*Karma\* als Test-Runner zu integrieren. Der Implementierungs- und Testcode wird mit Webpack gebündelt. Anwendungsteile werden in Angulars *TestBed* getestet. Wir haben diese Voreinstellungen nicht verändert. Es wäre allerdings problemlos möglich, eine eigene Test-Toolkette zusammenzustellen.

Für die End-2-End Tests nutzen wir *Cypress*. Es gibt zwei Kategorien von End-to-End-Test-Frameworks: Diejenigen, die WebDriver verwenden, und diejenigen, die es nicht tun. Das WebDriver-Protokoll definiert eine Möglichkeit, einen Browser mit einer Reihe von Befehlen fernzusteuern. Nicht alle Frameworks bauen auf WebDriver auf. Einige Frameworks integrieren sich direkt in den Browser. Das macht sie zuverlässiger, weniger komplex, aber auch weniger flexibel, da sie nur wenige Browser unterstützen. Für uns haben hier die Vorteile überwogen, da die Flexibilität und Browser-Limitationen keine Rolle spielten.

Es möglich End-2-End-Tests ebenfalls mit *Github Actions* zu automatisieren. Allerdings werden hier sowohl das Frontend als auch Backend benötigt. Wohingegen Unit-Tests das Backend *Mockend*, greifen End-2-End-Tests auf den gesamten Anwendungs-Stack zurück. Dies bringt eine Reihe von Komplikationen mit sich, welche nicht trivial mit Github Actions umsetzbar sind. Bedingt durch den Fokus des Projektes und der Vorlesung, sowie des zeitlichen Rahmens, haben wir uns dagegen entschieden, die e2e-Tests auf Github ausführen zu lassen.

Die API des Backends wird mit Hilfe von *Postman* bzw. *Newman* getestet. Diese Tools bieten eine breite Palette von Möglichkeiten an Test Suites zu implementieren, wobei es auch möglich ist diese zu exportieren.

Des Weiteren kommt das Unit Testframework *Jest* zum Einsatz, welches für die interne Logik des Backends verwendet wird.

### 3.15.2 Agile Prinzipien

In dem Projekt wurden folgende Agile Prinzipien beachtet:

Es wurde z.B. das Prinzip „Flow“ umgesetzt, da wir uns für das Kanban Modell entschieden haben, mit der eine stetig fortlaufende Produktentwicklung möglich ist. Somit war, wie in der Test-Strategie bereits beschrieben, der Fluss der Arbeit bereits visualisiert. Des Weiteren war dadurch ein guter Überblick über das Projekt, sowie der Status der einzelnen Aufgaben der Teammitglieder einsehbar.

Das Prinzip „Lernen Verstärken“ ist ein wichtiger Bestandteil der Agilen Prinzipien. Durch dieses Projekt haben sich die einzelnen Teammitglieder auf Grund neuer Technologien weiterentwickelt sowie weitergebildet. Ein Beispiel einer solchen Technologie ist das Jasmine Test Framework. Die verwendeten Technologien wurden zuvor in Bezug auf ihre Eignung für das Projekt analysiert und bewertet. Nach positiver Bewertung wurde der Umgang mit ihnen erlernt.

Gemäß der „Limitierung der laufenden Arbeiten“ wurden die einzelnen Anforderungen zunächst in Arbeitspakete unterteilt und nach Priorität geordnet. Die laufenden Arbeitspakete wurden den Teammitgliedern zugewiesen, sodass die Mitglieder diese nacheinander abarbeiten konnten. Somit fokussiert sich jedes Teammitglied immer nur auf das gerade von ihm bearbeitete Arbeitspaket.

Begründet auf dem Prinzip „Sagen statt Fragen“ war eine Erreichbarkeit der einzelnen Teammitglieder bei Fragen oder auch Anmerkungen stets gegeben. Somit konnte immer direkt ein Kontakt über Teams oder Notion mit dem Betroffenen hergestellt und infol-

### 3 Testplan & Testkonzept

gedessen unmittelbar an einer Lösung gearbeitet werden. Bei Rückfragen oder Problemen wurde dies in Notion in der entsprechenden Stelle dokumentiert und das jeweilige Teammitglied wurde markiert. Dieser bekam per E-Mail eine Benachrichtigung, sodass er schnellstmöglich darauf antworten konnte.

#### 3.15.3 Issue-Tracking und Bug-Tracking

Bugs sind Abweichungen der Software. Bugs zu entfernen bedeutet nicht gleich, dass im Anschluss die Software wieder fehlerfrei funktioniert. Die geringe Komplexität des entwickelten Systems und die komponentenbasiert Entwicklung, samt des bestehenden und schlanken Qualitätssicherungsplans, führten bei späteren e2e Tests und Analysen zu keinen Bugs. Zumindest keine, die offen dokumentiert wurden - oftmals wurden sie direkt beim Auftreten behoben.

Auch hier kann der Fall betrachtet werden, dass Fehler nicht direkt behoben werden. Wie kann mit Bug-Tracking umgegangen werden?

Eine Überlegung wäre die Integration von Jira (von Atlassian). Als zusätzliche Drittanbieter Applikation baut Jira auf agile Projekt Management Tools auf und verwendet, wie auch aktuell im Projekt benutzt, KanBan Boards. Bug & Issue tracking. Von Burndown-Diagrammen bis hin zu Sprint-Berichten bietet Jira dutzende von sofort einsetzbaren Reports mit Echtzeit-insights in die Leistung des Teams. Dies völlig automatisiert.

Die Integration von Jira würde auf jeden Fall Sinn ergeben, falls das Projekt weiter entwickelt werden soll. Die bisherige Dokumentation in Notion reicht allerdings für die Dokumentation aus.

#### 3.15.4 Bewusste Defects

Passwort-Recovery ist ein bewusster Defekt. Als die Anforderungen erfasst worden sind, wurde die Passwort-Recovery mit einer niedrigen Priorität erfasst und als nicht notwendig für das Basis-System eingestuft. Aus Zeitgründen wurde dann nicht mehr mit der Entwicklung begonnen.

Ein weiter Bewusster Defect ist die Eingabe der Mitspieler. Eine Kommaseparierte Eingabe ist nicht optimal für nicht-Informatiker. Aber auch hier haben wir die Priorität einer besseren Mitspieler-Erstellung als niedrig bewertet.

## 3.16 Test-Arten

### 3.16.1 Risikobasiertes Testen

In dem Projekt wurde der Testaufwand so geteilt, dass man mehr Testaufwand investiert, wo man am ehesten Fehler erwartet hat. Dagegen wurde der Testaufwand an Stellen reduziert, wo es keine \*Anforderungskritikalität geben kann.\*

Zum Beispiel, wurde mehr Testaufwand im Backend mit API-Tests investiert als der Test von dem Design in Frontend.

Also sollte der Fokus des Testentwurfsverfahrens auf der Entdeckung von Fehlern liegen, die große Auswirkungen haben.

D.h. vor dem Erstellen der Testspezifikation, wird basierend auf dem identifizierten Risiko entschieden, welche Tests erstellt und durchgeführt werden.

### 3.16.2 Automatisiertes Testen

Da manuelle Tests dauerhaft wiederholende Tests sind, werden sie mit dem Aufbau des Projekts immer teurer. Daher wurde in dem Projekt bei End-2-End-Tests automatisierte Tests erstellt und durchgeführt. Damit kann man die Software in kurzer Zeit mit weniger Aufwand und geringeren Kosten testen kann.

Darüber hinaus ist der Test sorgfältiger und zuverlässiger geworden. Man kann die Fehler damit schnell finden und beheben.

Aber man kann sagen, dass die automatisierte Tests nicht immer die gute Wahl sind, weil sich die Funktionen häufig ändern können und somit müssen die Testfälle geändert werden. Deswegen haben wir entschieden, manuelle und automatisierte Tests zu verwenden, um gute Qualität der Tests zu bekommen (Aufwand von Änderungen von Tests wurden hier in Betracht gezogen).

### 3.16.3 Sicherheits-Tests

Der Zugang zu den Benutzern des Backends ist passwortgeschützt. Ein Benutzer kann sich nur mit seinem bei der Registrierung angegebenen Passwort anmelden und darf nur auf seine eigenen Karten zugreifen. Dieser Mechanismus wurde mit API-Tests abgedeckt. Jedoch wurden keine weiteren sicherheitsrelevanten Tests durchgeführt, denn es werden keine personenbezogenen oder sicherheitsrelevanten Daten erfasst.

Des Weiteren ist die Anwendung (ein Spiel) und das Schlimmste was passieren könnte, wäre der Verlust von den Spielkarten. Daher halten wir die Entscheidung keine Sicherheits-Tests durchzuführen für vertretbar.

### 3.16.4 Regressions-Tests

Es wurde im Front- und Backend darauf geachtet, dass jedes Inkrement eine bestimmte Menge von wieder verwendbaren Tests aufweist, die nach jeder Iteration durchlaufen werden können. Das sollen gewährleisten, dass schon implementierte Funktionen durch die Ergänzungen des neuen Inkrements immer noch korrekt sind. Es wurde dabei auch auf einen hohen Automatisierungsgrad geachtet.

### 3.16.5 Last und Performance-Tests

Last und Performance-Tests sind Tests von nicht funktionalen Anforderungen. Sie bestimmen die Geschwindigkeit, Stabilität und Reaktionsfähigkeit der Software. Diese Tests helfen uns Beispielweise bei dem Projekt festzustellen, wie viele Benutzer das Programm parallel nutzen können. Außerdem könnte man mit dem Performance-Test überprüfen, wie schnell das entspannte Programm reagiert. Da unsere Programm ein kleines Programm ist, haben wir auf die Performance-Tests verzichtet und die Lasttests nur für Backend umgesetzt.

Für Login und Registrierung wurden keine Lasttests implementiert, da diese Funktionen nicht so oft aufgerufen werden.

Um die Lasttests von dem Aufrufen, Anlegen, Bearbeiten und Löschen von Karten haben wir das Tool JMeter von Apache verwendet.

### 3 Testplan & Testkonzept

JMeter ist ein Apache Performance-Testing Tool, das Lasttests für Web- und andere Anwendungen erzeugen kann.

Es wurde per Test-Setup im Backend 100 Karten durch SQL-Anfragen angelegt. In JMeter wurden die Voraussetzungen für jeden Test definiert. Z.b. die Anzahl der Threads (Users), die Ramp-up Periode, die Dauer des Tests (in Sekunden) sowie die geforderten Antwortzeiten.

#### 3.16.6 Usability Tests

Neben den bereits genannten Methoden führten wir Usability Tests durch. Hier stellte sich für uns die Frage, wie neue Nutzer auf unsere Anwendung reagieren. Dabei sind für uns folgende Aspekte wichtig:

- Ist unsere Anwendung leicht verständlich und selbsterklärend?
- Können Nutzer ein Spieldurchlauf ohne Vorkenntnisse durchführen?
- Können neue Nutzer innerhalb eines vorgegebenen Zeitrahmens in den Spiele-Bildschirm gelangen?
- Finden Nutzer keine Bugs oder Fehler, an welche wir nicht gedacht haben?

In einem nächsten Schritt stellt sich die Frage, welche Nutzer wir für die Usability Tests auswählen. Da sich die Anwendung nicht an ältere Menschen richtet, haben wir uns auf Jugendliche konzentriert. Dabei ist der Altersrahmen zwischen 18 und 30 Jahren.

Das Ziel der Usability-Tests ist für uns zu bestätigen, dass unser Produkt die Erwartungen erfüllt. Was wir nicht erreichen wollen, sind A/B-Tests. Den Nutzern wird die finale Version unserer Anwendung präsentiert. Es gibt keine Abwandlungen oder unterschiedliche Versionen zum Testen.

##### 3.16.6.1 Ablauf

Im Rahmen des Tests wird jeder Person die gleiche Aufgabe gestellt. Zunächst gilt es ein neuen Account anzulegen. Hierfür werden dem Nutzer keine Vorgaben hinsichtlich des Passworts oder Nutzernamen gemacht. Anschließend soll sich die Person, in den gerade erstellen Account, ein loggen. Nun werden ihr eine Reihe von Personen vorgegeben, die mit ihr das Spiel spielen. Die Aufgabe ist es für die Menge an Personen ein neues Spiel zu konfigurieren und zu starten. Anschließend werden 3 Runden gespielt. Nun soll das Spiel gestoppt und der Nutzer abgemeldet werden.

Während des Tests wird die Zeit gemessen. Der Testbeauftragte greift nicht ein, oder gibt Hinweise.

##### 3.16.6.2 Usability Test A

###### Person A - Zeit 4min

Person A ist 23 Jahre alt und studiert Wirtschaftsinformatik. Als Vorkenntnisse hat diese Person bereits ähnliche Apps genutzt. Hierzu zählt zum Beispiel Picolo.

Das registrieren und anmelden stellte für Person A kein Problem dar. Auch das starten und konfigurieren des Spiels führte zu keinen Schwierigkeiten. Allerdings wollte Person A das Spiel beenden, indem sie auf das Logo in der Header-Leiste klickt. Dieses hat die Person fälschlicherweise zum Login-Bildschirm geführt. Obwohl die Person bereits

eingelogged war. Das von uns vorgesehene Verhalten war, dass der Spiel-Beenden Button genutzt wird.

Person A gab an, gewohnt zu sein, dass das Logo stets zur Startseite führt.

### 3.16.6.3 Usability Test B

*Person B - Zeit 10min*

Testperson B studiert Medizin im Bereich Zahntechnik. Die Person hat zwar schon mal von Picolo gehört, hat eine solche App aber noch nicht selbst genutzt. Das Registrieren und Einloggen stellte auch für Person B kein Problem dar. Allerdings konnte die Person nichts mit einer Kommaseparierten Darstellung von Namen anfangen. Dass hier die Personen durch ein Komma getrennt eingetragen werden sollen, war für Person B nicht selbstverständlich.



## 4 Ausblick

Abschließend werden einige Möglichkeiten aufgezählt, mit denen das Projekt zukünftig noch erweitert werden könnte: Ein Beispiel wäre eine Passwort-Recovery mit dem der Benutzer sein Passwort zurücksetzen, könnte. Aufgrund einer niedrigen Priorität und der zeitlichen Begrenzung konnte dies nicht in diesem Projekt umgesetzt werden.

Mit Hinzufügen des Passwort-Recovery wäre dann somit auch das Speichern von E-Mail-Adressen notwendig, um das gespeicherte Passwort zurücksetzen zu können. Aufgrund der Speicherung von personenbezogenen bzw. sicherheitsrelevanten Daten wäre an dieser Stelle die Durchführung von Sicherheits-Tests sinnvoll. Eine weitere mögliche Erweiterung ist, das Projekt in eine Container-Virtualisierung auszulagern. Der Vorteil dabei besteht beispielsweise darin, dass die Start- sowie Stopnzeiten schneller sind und weniger Speicherplatz benötigt wird.



# Abbildungsverzeichnis

2.1 ISO/IEC 25010:2011 . . . . .	4
3.1 Die von uns initial erarbeitete Use Caseses . . . . .	6
3.2 Ein Snapshot unseres Kanban-Boards . . . . .	7
3.3 Ein beispielhafterer Feature-Branch und der Master-Branch . . . . .	16



# **Anhang**



## **A Testspezifikationen**

Der Anhang enthält die Testspezifikationen, welche wir in der Konzeptionsphase jedes Tickets erarbeitet haben.



# Registrierung

⌚ Created	@November 24, 2021 5:37 PM
✔ Status	Complete 🎉
✖ Priority	P2
👤 Person	
✖ Bereich	Frontend

## Testspezifikation

✓ → Vollautomatisiert

✓ → Halb-Automatisiert

👤 📲 → Manuell

Automatisiert	ID	Name	Setup/Voraussetzungen	Testschritte	Testdaten	Erwartete Ergebnisse	Priorität
	Frontend-Registrierung-1	Username corresponds to specifications	Modul geladen. Input field für Username vorhanden	Eingabe des Users [...] Submit Überprüfung ob eingegebener Nutzernname Spezifikationen entspricht	Userdaten	Falls Username nicht Spezifikationen entspricht, soll ein Alert geworfen werden. Registriervorgang wird nicht durchgeführt Falls Username den Spezifikationen entspricht, (und Password auch) Registriervorgang wird durchgeführt	high
	Frontend-Registrierung-2	Username already present in the system	Server gibt vorhandene Usernamen zurück	Eingabe des Users [...] Submit Überprüfung ob eingegebener Username bereits im System vorhanden ist	keine	Noch vor dem eigentlichen einloggen wird dem Nutzer visuell verdeutlicht, dass er eine falsch formulierte Email eingegeben hat.	high

Automatisiert	ID	Name	Setup/Voraussetzungen	Testschritte	Testdaten	Erwartete Ergebnisse	Priorität
	Frontend-Registrierung-3	Password corresponds to specifications	Modul geladen. Input field für password vorhanden	Eingabe des Users & Password [...] Submit Überprüfung ob eingegebenes password Spezifikationen entspricht	Userdaten	Falls password nicht Spezifikationen entspricht, soll ein ein Alert geworfen werden. Registriervorgang wird nicht durchgeführt Falls password den Spezifikationen entspricht, (und Password auch) Registriervorgang wird durchgeführt	high
	Frontend-Registrierung-4	Password does not corresponds	Password bereits eingegeben	Eingabe des Users & Password, Password [...] Submit Überprüfung ob eingegebenes password mit dem Originall übereinstimmt	Userdaten	Weiterleitung zum Login-Bildschirm unter der URL localhost:8888/login	high
	Frontend-Registrierung-5	Register	Alle Input-fields geladen, gefüllt, überprüft und bereit für den Submit.	Eingabe des Users & Password, Password [...] Submit	Userdaten	Weiterleitung zum Login-Bildschirm unter der URL localhost:8888/login	high
	Frontend-Registrierung-6	Server-response time 10 Sekunden.	registriervorgang	Eingabe des Users & Password, Password [...] Submit	Userdaten	Innerhalb von maximal 10 Sekunden wird der User zur component 'Login' weitergeleitet	medium

## Konzeption

Das Modul/die Komponente 'register' ermöglicht dem Nutzer die Registration:

Eingabefelder sind:

- ▼ Username

Username darf nur Zeichen und nicht aus nicht-numerischen Ziffern bestehen

- ▼ password

Die Mindestlänge des Passwortes besteht aus mind. 8 Zeichen und besteht aus mind. einem Sonderzeichen und einer Zahl

▼ repeat password

Das eingegebene Passwort muss mit der Wiederholung übereinstimmen

Des Weiteren soll das Modul über einen Button für die Übermittlung an den Server enthalten  
Bspw. 'Submit'/'Register'/'Registrieren'

Das Modul soll folgende Alerts werfen:

▼ invalid username

Der eingegebene Username entspricht nicht den Anforderungen

▼ invalid password

Der eingegebene Passwort entspricht nicht den Anforderungen

▼ username already present

Der eingegebene Username existiert bereits im System

## Mockup

...

### Register

---

[Login](#) [Register](#)

---







[Register](#)

---

Alle Tests müssen erfolgreich gewesen sein

Funktionalität ist gemerged

Integrationstests erfolgreich



# Header

⌚ Created	@January 20, 2022 3:18 PM
✔ Status	Complete 🎉
✖ Priority	
👤 Person	
⌚ Bereich	Frontend

## Testspezifikation

- Vollautomatisiert
- Halb-Automatisiert
- Manuell

<b>Testfall-Nummer/ID</b>	Frontend-Header-1
<b>Name/Zusammenfassung</b>	Sofern kein Nutzer angemeldet ist, werden im Header keine User-Menüpunkte (z.B. Ausloggen) angezeigt
<b>Setup/Voraussetzungen</b>	Kein User ist angemeldet
<b>Testdaten</b>	(keine) Nutzerdaten
<b>Testschritte</b>	Seite wird neu geladen oder Nutzer hat sich zuvor abgemeldet.
<b>Erwartetes Ergebnis</b>	Darstellung des Header in Nicht-eingeloggter-User Form
<b>Automatisiert</b>	
<b>Testfall-Nummer/ID</b>	Frontend-Header-2
<b>Name/Zusammenfassung</b>	Sofern ein Nutzer angemeldet ist, wird im Header das User-Menüpunkte (z.B. Ausloggen) angezeigt

<b>Setup/Voraussetzungen</b>	User ist angemeldet
<b>Testdaten</b>	Nutzerdaten
<b>Testschritte</b>	Keine - Nutzer muss angemeldet sein, sofern ist ein Testschritt die Anmeldung
<b>Erwartetes Ergebnis</b>	Darstellung des Header in Eingeloggter-User Form
<b>Automatisiert</b>	
<b>Testfall-Nummer/ID</b>	Frontend-Header-3
<b>Name/Zusammenfassung</b>	Ein Klick auf das Logo, navigiert (falls User eingeloggt) zur Spieleanwahl.
<b>Setup/Voraussetzungen</b>	User ist angemeldet
<b>Testdaten</b>	Nutzerdaten
<b>Testschritte</b>	Klick aufs Logo
<b>Erwartetes Ergebnis</b>	Navigation zu ' <code>'main-menu'</code>
<b>Automatisiert</b>	
<b>Testfall-Nummer/ID</b>	Frontend-Header-4
<b>Name/Zusammenfassung</b>	Ein Klick auf das Logo, navigiert (falls User NICHT eingeloggt) zur Root.
<b>Setup/Voraussetzungen</b>	User ist nicht angemeldet
<b>Testdaten</b>	-
<b>Testschritte</b>	Klick aufs Logo
<b>Erwartetes Ergebnis</b>	Navigation zu ' <code>'main-menu'</code>
<b>Automatisiert</b>	

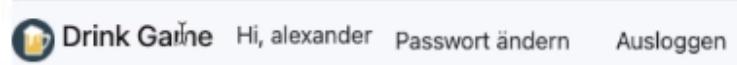
## Konzeption

Sobald der Nutzer angemeldet ist, sollen ihm zusätzliche Menüpunkte im Header (der oberen Navigation) angezeigt werden. Neben der Begrüßung mit "Hi, {{username}}" soll es die Möglichkeit zum Passwort ändern und ausloggen geben.

Ein Klick auf das Logo, navigiert (falls User eingeloggt) zur '`'main-menu'`.

Ein Klick auf das Logo, navigiert (falls User NICHT eingeloggt) zum Ausgangsmenü, mit Möglichkeit zum Registrieren/Anmelden.

## Mockup



item: Hi, {{username}}

item: Passwort ändern (route zu reset-password) VERWORF

item: Ausloggen

Alle tests müssen erfolgreich gewesen sein

Funktionalität ist gemerged

Integrationstests erfolgreich



# Login

⌚ Created	@November 25, 2021 12:39 PM
❖ Status	Complete 🙌
❖ Priority	P2
👤 Person	
❖ Bereich	Backend

## Testspezifikation

Es wird eine Testsuite mit Postman angelegt über die die Tests laufen. Je nach Bedarf wird noch mit Jest Unitests ergänzt. Manuelle Tests sind nicht vorgesehen. Nicht alle Aspekte des Login-Mechanismus können hier getestet werden. Controller, auf die nur nach einem erfolgreichem Login, zugegriffen werden darf müssen diese Tests jeweils selbst implementieren.

## Testfälle

✓ → Vollautomatisiert

✓ → Halb-Automatisiert

👤 → Manuell

Testfall-Nummer/ID	Backend-Login-1
Automatisiert	✓
Name/Zusammenfassung	Es wird der positive Fall eines Login Versuches getestet, bei dem der Benutzer eine gültige Kombination von Benutzername und Passwort angibt.

<b>Setup/Voraussetzungen</b>	1.) Der zu testende Benutzer ist im Backend (in der Datenbank vorhanden, bzw. registriert). 2.) Das Backend ist gestartet.
<b>Testdaten</b>	Benutzername: backendLoginTestUserName, Passwort: backendLoginTestUserPassword Als JSON in Body des HTTP Requests: { "username": "backendLoginTestUserName", "password": "backendLoginTestUserPassword" }
<b>Testschritte</b>	1.) Sende HTTP Post Anfrage mit den Informationen Benutzername und Passwort and den Auth-Controller ( <a href="http://localhost:3000/auth/login">http://localhost:3000/auth/login</a> )
<b>Erwartetes Ergebnis</b>	1.) Die Antwort auf die POST Anfrage hat den Statuscode 201. 2.) Im Body der Antwort ist ein Token enthalten und den Benutzer: { "access_token": "a token here", "user": { "id": "the id of the user", "username": "name of the user" } }

<b>Testfall-Nummer/ID</b>	Backend-Login-2
Automatisiert	✓
<b>Name/Zusammenfassung</b>	Der Benutzer gibt einen vorhandenen Benutzernamen aber ein falsches Passwort an
<b>Setup/Voraussetzungen</b>	1.) Der zu testende Benutzer ist im Backend (in der Datenbank) vorhanden. 2) Das Backend ist gestartet.
<b>Testdaten</b>	Benutzername: backendLoginTestUserName, Passwort: backendLoginTestUserWrongPassword Als JSON in Body des HTTP Requests: { "username": "backendLoginTestUserName", "password": "backendLoginTestUserWrongPassword" }
<b>Testschritte</b>	1.) Sende HTTP Post Anfrage mit den Informationen Benutzername und Passwort and den Auth-Controller ( <a href="http://localhost:3000/auth/login">http://localhost:3000/auth/login</a> )
<b>Erwartetes Ergebnis</b>	1.) Die Antwort auf die POST Anfrage hat den Statuscode 401 (Unauthorized). 2.) Der Body der Antwort enthält kein Token (Der Key access_token ist nicht vorhanden).
<b>Testfall-Nummer/ID</b>	Backend-Login-3
Automatisiert	Halb-Automatisiert

<b>Name/Zusammenfassung</b>	Der Benutzer gibt einen nicht vorhandenen Benutzernamen aber ein vorhandenes Passwort an
<b>Setup/Voraussetzungen</b>	1.) Der Benutzername aus den Testdaten ist nicht vorhanden das Passwort aber schon. 2.) Das Backend ist gestartet.
<b>Testdaten</b>	Benutzername: backendLoginTestWrongUserName, Passwort: backendLoginTestUserPassword Als JSON in Body des HTTP Requests: { "username": "backendLoginTestWrongUserName", "password": "backendLoginTestUserPassword" }
<b>Testschritte</b>	1.) Sende HTTP Post Anfrage mit den Informationen Benutzername und Passwort und den Auth-Controller ( <a href="http://localhost:3000/auth/login">http://localhost:3000/auth/login</a> ) 1.) Sende HTTP Post Anfrage mit den Informationen Benutzername und Passwort und den Login-Controller ( <a href="http://localhost:3000/auth/login">http://localhost:3000/auth/login</a> )
<b>Erwartetes Ergebnis</b>	1.) Die Antwort auf die POST Anfrage hat den Statuscode 401 (Unauthorized). 2.) Der Body der Antwort enthält kein Token (Der Key access_token ist nicht vorhanden).

<b>Testfall-Nummer/ID</b>	<b>Backend-Login-4</b>
Automatisiert	✓
<b>Name/Zusammenfassung</b>	Der Benutzer gibt einen nicht vorhandenen Benutzernamen und ein nicht vorhandenes Passwort an
<b>Setup/Voraussetzungen</b>	1.) Der Benutzername und das Passwort aus den Testdaten sind nicht vorhanden. 2.) Das Backend ist gestartet.
<b>Testdaten</b>	1.) Benutzername: backendLoginTestWrongUserName, Passwort: backendLoginTestWrongUserPassword Als JSON in Body des HTTP Requests: { "username": "backendLoginTestWrongUserName", "password": "backendLoginTestWrongUserPassword" } 2.) Benutzername: Passwort: (Beide leer) Als JSON in Body des HTTP Requests: { "username": "", "password": "" } 3.) Body komplett leer
<b>Testschritte</b>	1.) Sende HTTP Post Anfrage mit den Informationen Benutzername und Passwort und den Auth-Controller ( <a href="http://localhost:3000/login">http://localhost:3000/login</a> )
<b>Erwartetes Ergebnis</b>	1.) Die Antwort auf die POST Anfrage hat den Statuscode 401 (Unauthorized). 2.). Der Body der Antwort enthält kein Token (Der Key access_token ist nicht vorhanden).

## Konzeption

Über den Pfad /auth/login (Login Controller) kann über eine HTTP POST Anfrage ein Login JWT Login-Token angefragt werden, das dann benutzt wird um auf andere Funktionen des Backends zuzugreifen. Dieser Zugriff wird mit einem Guard umgesetzt.

In der POST Anfrage über gibt der Benutzer seinen Benutzernamen und sein Passwort. Mithilfe eines User-Services wird überprüft, ob die Angaben Benutzername und Passwort korrekt sind. Im Fall, dass der Login nicht erfolgreich ist, wird der Statuscode 401 Unauthorized zurückgegeben.

Das Token ist ein Bearer Token.

---

- Alle tests müssen erfolgreich gewesen sein
- Funktionalität ist gemerged
- Integrationstests erfolgreich



# Login

⌚ Created	@November 10, 2021 6:13 PM
✔ Status	Complete 🎉
✖ Priority	
👤 Person	
📍 Bereich	Frontend

Es gibt den User aber Passwort falsch

Es gibt den Nutzer gar nicht.

## Testspezifikation



Die vollautomatisierten Tests werden mithilfe von Jasmine und Karma ausgeführt. Karma führt dabei die Tests regelmäßig im Hintergrund aus und weiß darauf hin, sollten Tests fehlschlagen.

Außerdem wurde mittels Github Actions das Testen weiter automatisiert. Sobald ein neuer Commit auf das Github Repo gepushed wird, baut Github die Applikation und testet sie.

Hierfür wird die Anwendung auf einem Github Server gestartet und mittels Puppeteer eine Chrome-Headless Instanz gestartet. Diese durchläuft anschließend die mittels Jasmin definierten Tests.

✓ → Vollautomatisiert

✓ → Halb-Automatisiert

👤🟡 → Manuell

Automatisiert	ID	Name	Setup/Voraussetzungen	Testschritte	Testdaten	Erwartete Ergebnisse	Priorität
✓	Frontend-Login-1	Benutzer kann sich in bestehenden Account einloggen	Account anlegen	Eingabe der benötigen Zugangsdaten. Auf Erfolg überprüfen.	Benutzer-Daten	Erfolgreiches einloggen und navigation zum nächsten Screen	high
👤🟡	Frontend-Login-2	Benutzer kann sich <b>nicht</b> in ein nicht existierenden Account einloggen	Nicht existierende Nutzernname/Passwort Kombination finden.	Eingabe der Zugangsdaten. Auf Misserfolg überprüfen.	Nicht-existierende Benutzer-Daten	Fehlermeldung, welche den Nutzer darüber informiert, dass die eingegebenen Benutzerdaten ungültig sind.	high

Automatisiert	ID	Name	Setup/Voraussetzungen	Testschritte	Testdaten	Erwartete Ergebnisse	Priorität
	Frontend-Login-3	Benutzer gibt eine fehlerhaft formatierte Email-Adresse ein.	keine	Fehlerhafte Email-Adresse in das dafür vorgesehene Feld eintragen.	keine	Noch vor dem eigentlichen einloggen wird dem Nutzer visuell verdeutlicht, dass er eine falsch formatierte Email eingegeben hat.	low
🟡🟡	Frontend-Login-4	Der Benutzer kann nicht manuell über die Adressleiste zu dem Spiele-Bildschirm gelangen.	keine	Sicherstellen, dass man nicht aktuell eingeloggt ist. Eingabe der URL localhost:8888/game	keine	Weiterleitung zum Login-Bildschirm unter der URL localhost:8888/login	high
🟡🟡	Frontend-Login-5	Einlog-Vorgang dauert nicht länger als 10 Sekunden.	Account anlegen	Eingabe der Nutzerdaten und betätigen der Einlogfunktion.	Benutzer-Daten	Innerhalb von maximal 10 Sekunden wird der Nutzer zum Spiele Bildschirm weitergeleitet.	medium
✓	Frontend-Login-6	Die Eingabe-Felder sollten initial leer sein.	Starten der Anwendung	Navigierung zum Login Bildschirm. Überprüfung der Eingabe-Felder auf Inhalte.	keine	Leere Eingabe-Felder für Benutzername und Passwort.	medium
✓	Frontend-Login-7	Der Login soll mit den eingegebenen Daten aufgerufen werden.	Starten der Anwendung, Navigierung zum Login-Screen	Eingabe der Nutzerdaten, Überwachung der Aufrufe	Benutzer-Daten	Aufruf der Funktion mit eingegebenen Benutzer-Daten	high
✓	Frontend-Login-8	Der Login sollte nur ein Mal eine Anfrage an das Backend schicken	Starten der Anwendung, Navigierung zum Login-Screen	Eingabe der Nutzerdaten, Überwachung der Aufrufe	Benutzer-Daten	Einmaliger Aufruf der Funktion und einmalige Anfrage an das Backend	high
🟡🟡	Frontend-Login-9	Benutzer füllt das Benutzername Feld nicht aus.	Starten der Anwendung, Navigierung zum Login-Screen	Zum Login-Screen navigieren und ein Passwort eingeben. Login-Button betätigen.	Beliebiges Passwort	Das fehlende Feld wird rot hervorgehoben und es wird nicht zu einem neuen Bildschirm navigiert.	low
🟡🟡	Frontend-Login-10	Benutzer füllt das Passwort Feld nicht aus.	Starten der Anwendung, Navigierung zum Login-Screen	Zum Login-Screen navigieren und einen Nutzernamen eingeben. Login-Button betätigen.	Beliebiger Nutzernamen	Das fehlende Feld wird rot hervorgehoben und es wird nicht zu einem neuen Bildschirm navigiert.	low

## Konzeption

Das Login-Feld ermöglicht dem Nutzer sich für die Nutzung der App zu autorisieren.

Hierfür werden ein Nutzernname / Email und ein Passwort verlangt.

Hierfür wird ein bereits angelegter Account benötigt.

Für die Navigation zum nächsten Screen ist ein einloggen zwingend erforderlich.

## Mockup



- Alle tests müssen erfolgreich gewesen sein
- Funktionalität ist gemerged
- Integrationstests erfolgreich



# Hauptmenü

⌚ Created	@December 13, 2021 12:40 PM
❖ Status	Complete 🙌
❖ Priority	
👤 Person	
❖ Bereich	Frontend

## Testspezifikation

- Vollautomatisiert
- Halb-Automatisiert
- Manuell

<b>Testfall-Nummer/ID</b>	Frontend-Hauptmenü-1
<b>Name/Zusammenfassung</b>	Der 'Play'-Button wird angezeigt
<b>Setup/Voraussetzungen</b>	Der Nutzer muss sich erfolgreich angemeldet haben.
<b>Testdaten</b>	Nutzerdaten
<b>Testschritte</b>	Die Option Login im Hauptmenü wählen, den Nutzer erfolgreich einloggen, überprüfen ob der Spielebutton gerendert wird.
<b>Erwartetes Ergebnis</b>	Der Spielebutton befindet sich auf der Seite.
<b>Automatisiert</b>	

<b>Testfall-Nummer/ID</b>	Frontend-Hauptmenü-2
<b>Name/Zusammenfassung</b>	Der 'Play'-Button navigiert zum Spiele-Bildschirm

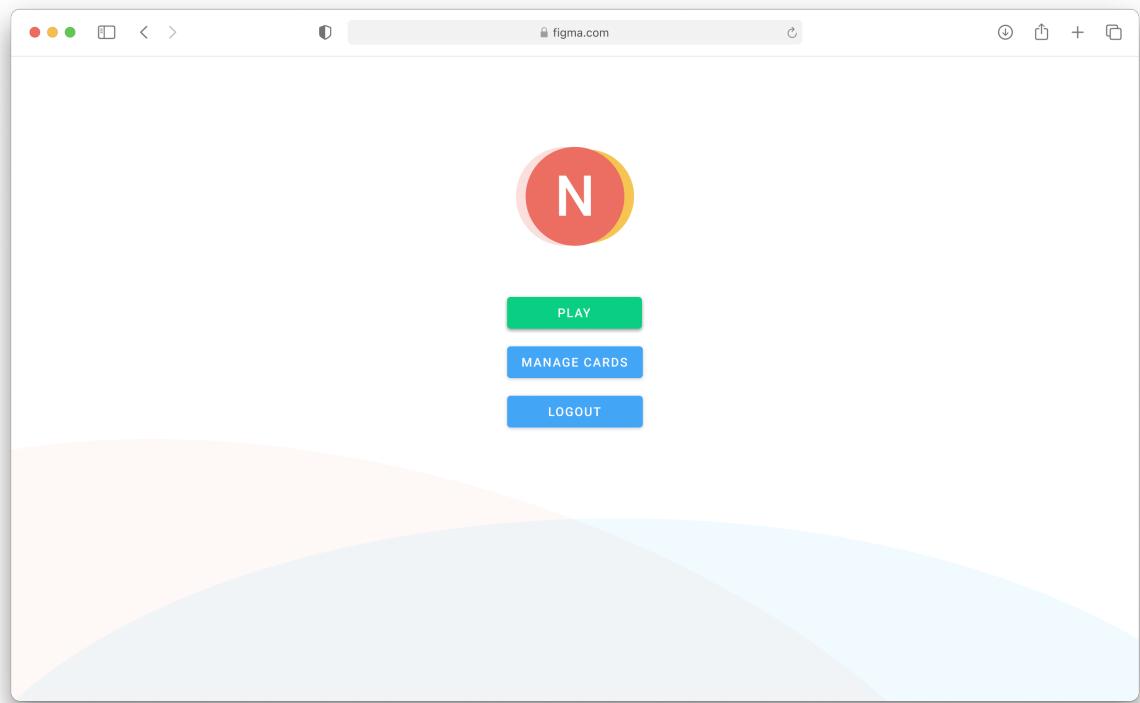
<b>Setup/Voraussetzungen</b>	Der Nutzer muss sich erfolgreich angemeldet haben.
<b>Testdaten</b>	Nutzerdaten
<b>Testschritte</b>	Die Option Login im Hauptmenü wählen, den Nutzer erfolgreich einloggen, auf den Play-Button klicken, überprüfen ob zum Spiele-Screen navigiert wird.
<b>Erwartetes Ergebnis</b>	Es wird zum Spiele-Screen navigiert.
<b>Automatisiert</b>	
<b>Testfall-Nummer/ID</b>	Frontend-Hauptmenü-3
<b>Name/Zusammenfassung</b>	Der 'Karten-Verwalten'-Button navigiert zum Verwaltungs-Bildschirm
<b>Setup/Voraussetzungen</b>	Der Nutzer muss sich erfolgreich angemeldet haben.
<b>Testdaten</b>	Nutzerdaten
<b>Testschritte</b>	Die Option Login im Hauptmenü wählen, den Nutzer erfolgreich einloggen, auf den Karten-Verwalten-Button klicken, überprüfen ob zum Verwalten-Screen navigiert wird.
<b>Erwartetes Ergebnis</b>	Es wird zum Spiele-Screen navigiert.
<b>Automatisiert</b>	
<b>Testfall-Nummer/ID</b>	Frontend-Hauptmenü-3
<b>Name/Zusammenfassung</b>	Der 'Logout'-Button loggt den Nutzer erfolgreich aus.
<b>Setup/Voraussetzungen</b>	Der Nutzer muss sich erfolgreich angemeldet haben.
<b>Testdaten</b>	Nutzerdaten
<b>Testschritte</b>	Die Option Login im Hauptmenü wählen, den Nutzer erfolgreich einloggen, auf den Logout-Button klicken, überprüfen ob die logout() Methode des AuthentifizierungsService aufgerufen wurde.
<b>Erwartetes Ergebnis</b>	Die Nutzerdaten befinden sich nicht mehr im Authentifizierungs-Service
<b>Automatisiert</b>	
<b>Testfall-Nummer/ID</b>	Frontend-Hauptmenü-4
<b>Name/Zusammenfassung</b>	Ein ausgeloggter Nutzer hat kein Zugriff mehr auf die Nutzerdaten des zuvor eingeloggten Nutzers
<b>Setup/Voraussetzungen</b>	Der Nutzer muss sich erfolgreich angemeldet haben.
<b>Testdaten</b>	Nutzerdaten

<b>Testschritte</b>	Die Option Login im Hauptmenü wählen, den Nutzer erfolgreich einloggen, auf den Logout-Button klicken, überprüfen ob weiterhin zum Haupt-Menü gewechselt werden kann, überprüfen ob weiterhin der Verwalten-Screen aufgerufen werden kann.
<b>Erwartetes Ergebnis</b>	Der Nutzer sollte lediglich die Startseite, den Login und Registrierungs-Screen aufrufen können. Es sollten keine weiteren Nutzerdaten einsehbar sein.
<b>Automatisiert</b>	

## Konzeption

Nachdem ein Nutzer sich erfolgreich angemeldet hat, erreicht er das Hauptmenü. Hier kann er unmittelbar ein Spiel starten. Hierdurch navigiert er zu dem eigentlichen Spiel-Bildschirm. Alternativ wird ihm die Möglichkeit geboten, zur Karten-Verwaltung zu navigieren. Abschließend kann er sich auch ausloggen. Hierdurch navigiert er zurück zur Startseite der Anwendung.

## Mockup



- 
- Alle tests müssen erfolgreich gewesen sein
  - Funktionalität ist gemerged
  - Integrationstests erfolgreich



# Karte anlegen, abrufen, löschen, bearbeiten

⌚ Created	@December 13, 2021 12:53 PM
❖ Status	Complete 🎉
❖ Priority	P2
👤 Person	
❖ Bereich	Backend

## Testspezifikation API Tests

- Vollautomatisiert
- Halb-Automatisiert
- Manuell

## Testspezifikation Abrufen von Karten

Testfall-Nummer/ID	Backend-Cards-1.1
Name/Zusammenfassung	Getestet wird ein positiver Fall beim Abrufen aller Karten, wenn mehrere Karten vorhanden sind
Setup/Voraussetzungen	1.) Der Benutzer ist registriert und eingeloggt. 2.) Es sind mehrere Karten für den Benutzer in der Datenbank vorhanden.
Testdaten	URL: <a href="http://localhost:3000/cards/getAllCards/[userID]">http://localhost:3000/cards/getAllCards/[userID]</a>

<b>Testschritte</b>	1.) Sende HTTP GET Anfrage an den Cards-Controller
<b>Erwartetes Ergebnis</b>	1.) Der Body der Antwort enthält ein Array mit mehreren Karten 2.) Status: 200
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-1.2
<b>Name/Zusammenfassung</b>	Getestet wird ein positiver Fall beim Abrufen aller Karten, wenn keine Karten vorhanden sind
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist registriert und eingeloggt. 2.) Es sind KEINE Karten für den Benutzer in der Datenbank vorhanden.
<b>Testdaten</b>	URL: http://localhost:3000/cards/getAllCards/[userID]
<b>Testschritte</b>	1.) Sende HTTP GET Anfrage an den Cards-Controller
<b>Erwartetes Ergebnis</b>	1.) Der Body der Antwort enthält ein leeres Array 2.) Status: 200
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-1.3
<b>Name/Zusammenfassung</b>	Getestet wird ein negativer Fall beim Abrufen aller Karten, wenn der Benutzer nicht eingeloggt ist.
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist nicht eingeloggt.
<b>Testdaten</b>	URL: http://localhost:3000/cards/getAllCards/[userID]
<b>Testschritte</b>	1.) Sende HTTP GET Anfrage an den Cards-Controller
<b>Erwartetes Ergebnis</b>	1.) Der Body der Antwort enthält keine Karten 2.) Status: 401
<b>Automatisiert</b>	✓

## Testspezifikation Löschen von Karten

<b>Testfall-Nummer/ID</b>	Backend-Cards-2.1
<b>Name/Zusammenfassung</b>	Getestet wird ein positiver Fall beim Löschen einer Karte, wenn der Benutzer vorhanden und eingeloggt ist, sowie die Karte auch vorhanden ist.
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist registriert und eingeloggt. 2.) Die Karte ist vorhanden
<b>Testdaten</b>	URL: http://localhost:3000/cards/deleteCard/[cardId]/[userID]
<b>Testschritte</b>	1.) Sende HTTP DELETE Anfrage an den Cards-Controller

<b>Erwartetes Ergebnis</b>	1.) Der Body der Antwort ist leer 2.) Status: 200
<b>Automatisiert</b>	✓

<b>Testfall-Nummer/ID</b>	Backend-Cards-2.2
<b>Name/Zusammenfassung</b>	Getestet wird ein negativer Fall beim Löschen einer Karte, wenn der Benutzer vorhanden und eingeloggt ist, die Karte jedoch nicht vorhanden ist.
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist registriert und eingeloggt. 2.) Die Karte ist nicht vorhanden
<b>Testdaten</b>	URL: <a href="http://localhost:3000/cards/deleteCard/[cardId]/[userId]">http://localhost:3000/cards/deleteCard/[cardId]/[userId]</a>
<b>Testschritte</b>	1.) Sende HTTP DELETE Anfrage an den Cards-Controller
<b>Erwartetes Ergebnis</b>	1.) Status: 404
<b>Automatisiert</b>	✓

<b>Testfall-Nummer/ID</b>	Backend-Cards-2.3
<b>Name/Zusammenfassung</b>	Getestet wird ein negativer Fall beim Löschen einer Karte, wenn der Benutzer nicht eingeloggt ist.
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist nicht eingeloggt.
<b>Testdaten</b>	URL: <a href="http://localhost:3000/cards/deleteCard/[cardId]/[userId]">http://localhost:3000/cards/deleteCard/[cardId]/[userId]</a>
<b>Testschritte</b>	1.) Sende HTTP DELETE Anfrage an den Cards-Controller
<b>Erwartetes Ergebnis</b>	1.) Der Body der Antwort ist leer 2.) Status: 401
<b>Automatisiert</b>	✓

<b>Testfall-Nummer/ID</b>	Backend-Cards-2.4
<b>Name/Zusammenfassung</b>	Getestet wird ein negativer Fall beim Löschen einer Karte, wenn der Benutzer registriert und eingeloggt ist, aber auf eine Karte eines anderen Benutzers zugreifen will.
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist nicht eingeloggt. 2.) Es existiert ein weiterer Benutzer, der mindestens eine Karte besitzt.
<b>Testdaten</b>	URL: <a href="http://localhost:3000/cards/deleteCard/[cardId]/[userId]">http://localhost:3000/cards/deleteCard/[cardId]/[userId]</a>
<b>Testschritte</b>	1.) Sende HTTP DELETE Anfrage an den Cards-Controller
<b>Erwartetes Ergebnis</b>	1.) Der Body der Antwort ist leer 2.) Status: 404
<b>Automatisiert</b>	✓

## Testspezifikation Bearbeiten von Karten

<b>Testfall-Nummer/ID</b>	Backend-Cards-3.1
<b>Name/Zusammenfassung</b>	Getestet wird ein positiver Fall beim Bearbeiten einer Karte, wenn der Benutzer registriert und eingeloggt ist und er erfolgreich eine Karte bearbeitet
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist registriert und eingeloggt. 2.) Der Benutzer hat mindestens eine Karte
<b>Testdaten</b>	URL: <a href="http://localhost:3000/cards/updateCard">http://localhost:3000/cards/updateCard</a> Body: { "id": [cardId], "text": [text of the card] }
<b>Testschritte</b>	1.) Sende HTTP PUT Anfrage an den Cards-Controller
<b>Erwartetes Ergebnis</b>	1.) Der Status der Nachricht ist 200
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-3.2
<b>Name/Zusammenfassung</b>	Getestet wird ein negativer Fall beim Bearbeiten einer Karte, wenn der Benutzer registriert und eingeloggt ist und er eine Karte bearbeitet, die gar nicht existiert (Es gibt keine Karte mit der ID)
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist registriert und eingeloggt. 2.) Die Karte, die er bearbeiten will existiert nicht.
<b>Testdaten</b>	URL: <a href="http://localhost:3000/cards/updateCard">http://localhost:3000/cards/updateCard</a> Body: { "id": [cardId], "text": [text of the card] }
<b>Testschritte</b>	1.) Sende HTTP PUT Anfrage an den Cards-Controller
<b>Erwartetes Ergebnis</b>	1.) Der Status der Nachricht ist 404
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-3.3
<b>Name/Zusammenfassung</b>	Getestet wird ein negativer Fall beim Bearbeiten einer Karte, wenn der Benutzer registriert und eingeloggt ist und er eine Karte bearbeitet, jedoch verbotene Zeichen im Text der Karte verwendet
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist registriert und eingeloggt. 2.) Der Benutzer hat mindestens eine Karte
<b>Testdaten</b>	URL: <a href="http://localhost:3000/cards/updateCard">http://localhost:3000/cards/updateCard</a> Body: { "id": [cardId], "text": [text of the card with forbidden characters] }

<b>Testschritte</b>	1.) Sende HTTP PUT Anfrage an den Cards-Controller
<b>Erwartetes Ergebnis</b>	1.) Der Status der Nachricht ist 403
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-3.4
<b>Name/Zusammenfassung</b>	Getestet wird ein negativer Fall beim Bearbeiten einer Karte, wenn der Benutzer registriert und eingeloggt ist, jedoch eine Karte eines anderen Nutzers bearbeitet.
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist registriert und eingeloggt. 2.) Es existiert ein anderer Benutzer der mindestens eine Karte besitzt
<b>Testdaten</b>	URL: http://localhost:3000/cards/updateCard Body: { "id": [cardId], "text": [text of the card] }
<b>Testschritte</b>	1.) Sende HTTP PUT Anfrage an den Cards-Controller
<b>Erwartetes Ergebnis</b>	1.) Der Status der Nachricht ist 404
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-3.5
<b>Name/Zusammenfassung</b>	Getestet wird ein negativer Fall beim Bearbeiten einer Karte, wenn der Benutzer nicht eingeloggt ist
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist nicht eingeloggt.
<b>Testdaten</b>	URL: http://localhost:3000/cards/updateCard Body: { "id": [cardId], "text": [text of the card] }
<b>Testschritte</b>	1.) Sende HTTP PUT Anfrage an den Cards-Controller
<b>Erwartetes Ergebnis</b>	1.) Der Status der Nachricht ist 401
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-3.6
<b>Name/Zusammenfassung</b>	Getestet wird ein negativer Fall beim Bearbeiten einer Karte, wenn der Inhalt der Karte zu lang ist (mehr als 250 Zeichen hat)
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist registriert und eingeloggt. 2.) Der Benutzer hat mindestens eine Karte
<b>Testdaten</b>	URL: http://localhost:3000/cards/updateCard Body: { "id": [cardId], "text": [text of the card with more than 250 characters] }
<b>Testschritte</b>	1.) Sende HTTP PUT Anfrage an den Cards-Controller

<b>Erwartetes Ergebnis</b>	1.) Der Status der Nachricht ist 403
<b>Automatisiert</b>	✓

## Testspezifikation Erstellen von Karten

<b>Testfall-Nummer/ID</b>	Backend-Cards-4.1
<b>Name/Zusammenfassung</b>	Getestet wird ein positiver Fall beim Erstellen einer Karte, wenn der Benutzer registriert und eingeloggt ist und erfolgreich eine neue Karte erstellt
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist registriert und eingeloggt.
<b>Testdaten</b>	URL: <a href="http://localhost:3000/cards/createCard">http://localhost:3000/cards/createCard</a> Body: { "text": [text of the card] }
<b>Testschritte</b>	1.) Sende HTTP POST Anfrage an den Cards-Controller
<b>Erwartetes Ergebnis</b>	1.) Der Status der Nachricht ist 201
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-4.2
<b>Name/Zusammenfassung</b>	Getestet wird ein negativer Fall beim Erstellen einer Karte, wenn der Benutzer registriert und eingeloggt ist und eine Karte mit verbotenen Zeichen im Text erstellen will
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist registriert und eingeloggt.
<b>Testdaten</b>	URL: <a href="http://localhost:3000/cards/createCard">http://localhost:3000/cards/createCard</a> Body: { "text": [text of the card with forbidden characters] }
<b>Testschritte</b>	1.) Sende HTTP POST Anfrage an den Cards-Controller
<b>Erwartetes Ergebnis</b>	1.) Der Status der Nachricht ist 403
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-4.3
<b>Name/Zusammenfassung</b>	Getestet wird ein negativer Fall beim Erstellen einer Karte, wenn der Benutzer nicht eingeloggt ist
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist nicht eingeloggt
<b>Testdaten</b>	URL: <a href="http://localhost:3000/cards/createCard">http://localhost:3000/cards/createCard</a> Body: { "text": [text of the card] }

<b>Testschritte</b>	1.) Sende HTTP POST Anfrage an den Cards-Controller
<b>Erwartetes Ergebnis</b>	1.) Der Status der Nachricht ist 401
<b>Automatisiert</b>	✓

<b>Testfall-Nummer/ID</b>	Backend-Cards-4.4
<b>Name/Zusammenfassung</b>	Getestet wird ein negativer Fall beim Erstellen einer Karte, wenn der Inhalt der Karte zu lang ist (mehr als 250 Zeichen)
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist registriert und eingeloggt.
<b>Testdaten</b>	URL: <code>http://localhost:3000/cards/createCard</code> Body: <code>{ "text": [text of the card with more than 250 characters] }</code>
<b>Testschritte</b>	1.) Sende HTTP POST Anfrage an den Cards-Controller
<b>Erwartetes Ergebnis</b>	1.) Der Status der Nachricht ist 403
<b>Automatisiert</b>	✓

## Testspezifikation Kombination von Erstellen, Abrufen, Bearbeiten, Löschen

<b>Testfall-Nummer/ID</b>	Backend-Cards-5.1
<b>Name/Zusammenfassung</b>	Getestet wird ein positiver Fall beim dem eine Karte erstellt, dann abgerufen, bearbeitet und schließlich gelöscht wird.
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer ist registriert und eingeloggt
<b>Testdaten</b>	1.) URL: <code>http://localhost:3000/cards/createCard</code> Body: <code>{ "text": [text of the card] }</code> 2.) URL: <code>http://localhost:3000/cards/getAllCards/[userID]</code> 3.) URL: <code>http://localhost:3000/cards/updateCard</code> Body: <code>{ "id": [cardId], "text": [text of the card] }</code> 4.) URL: <code>http://localhost:3000/cards/deleteCard/[cardId]/[userID]</code>
<b>Testschritte</b>	1.) Sende HTTP POST Anfrage an den Cards-Controller 2.) Sende HTTP GET Anfrage an den Cards-Controller 3.) Sende HTTP PUT Anfrage an den Cards-Controller 4.) Sende HTTP DELETE Anfrage an den Cards-Controller
<b>Erwartetes Ergebnis</b>	1.) Der Status der ersten Nachricht ist 201 2.) Der Status der zweiten Nachricht ist 200 3.) Der Status der dritten Nachricht ist 201 4.) Der Status der vierten Nachricht ist 200
<b>Automatisiert</b>	✓

# Testspezifikation Cards Logic

<b>Testfall-Nummer/ID</b>	Backend-Cards-Logic-1.1
<b>Name/Zusammenfassung</b>	Positiver Fall beim Überprüfen der Länge des Text der Karte CuT: UserLogic (src/cards/cards-logic.ts)
<b>Setup/Voraussetzungen</b>	1.) Das CardsLogic Objekt ist erzeugt
<b>Testdaten</b>	Ein String der Länge 1 mit nur gültigen Zeichen
<b>Testschritte</b>	1.) Aufrufen der Funktion checkCardText mit dem Testdaten String
<b>Erwartetes Ergebnis</b>	Die Funktion wirft keine Ausnahme
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-Logic-1.2
<b>Name/Zusammenfassung</b>	Positiver Fall beim Überprüfen der Länge des Text der Karte CuT: UserLogic (src/cards/cards-logic.ts)
<b>Setup/Voraussetzungen</b>	1.) Das CardsLogic Objekt ist erzeugt
<b>Testdaten</b>	Ein String der Länge 250 mit nur gültigen Zeichen
<b>Testschritte</b>	1.) Aufrufen der Funktion checkCardText mit dem Testdaten String
<b>Erwartetes Ergebnis</b>	Die Funktion wirft keine Ausnahme
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-Logic-1.3
<b>Name/Zusammenfassung</b>	Positiver Fall beim Überprüfen der Länge des Text der Karte CuT: UserLogic (src/cards/cards-logic.ts)
<b>Setup/Voraussetzungen</b>	1.) Das CardsLogic Objekt ist erzeugt
<b>Testdaten</b>	Ein String der Länge zwischen 1 und 250 mit nur gültigen Zeichen
<b>Testschritte</b>	1.) Aufrufen der Funktion checkCardText mit dem Testdaten String
<b>Erwartetes Ergebnis</b>	Die Funktion wirft keine Ausnahme
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-Logic-1.4
<b>Name/Zusammenfassung</b>	Negativer Fall beim Überprüfen der Länge des Text der Karte CuT: UserLogic (src/cards/cards-logic.ts)

<b>Setup/Voraussetzungen</b>	1.) Das CardsLogic Objekt ist erzeugt
<b>Testdaten</b>	Ein undefinierter String (undefined)
<b>Testschritte</b>	1.) Aufrufen der Funktion checkCardText mit dem undefinierten String
<b>Erwartetes Ergebnis</b>	Die Funktion wirft eine Forbidden Ausnahme
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-Logic-1.5
<b>Name/Zusammenfassung</b>	Negativer Fall beim Überprüfen der Länge des Text der Karte CuT: UserLogic (src/cards/cards-logic.ts)
<b>Setup/Voraussetzungen</b>	1.) Das CardsLogic Objekt ist erzeugt
<b>Testdaten</b>	Ein leerer String
<b>Testschritte</b>	1.) Aufrufen der Funktion checkCardText mit dem leeren String
<b>Erwartetes Ergebnis</b>	Die Funktion wirft eine Forbidden Ausnahme
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-Logic-1.6
<b>Name/Zusammenfassung</b>	Negativer Fall beim Überprüfen der Länge des Text der Karte CuT: UserLogic (src/cards/cards-logic.ts)
<b>Setup/Voraussetzungen</b>	1.) Das CardsLogic Objekt ist erzeugt
<b>Testdaten</b>	Ein String der Länge 251
<b>Testschritte</b>	1.) Aufrufen der Funktion checkCardText mit dem Testdaten String
<b>Erwartetes Ergebnis</b>	Die Funktion wirft eine Forbidden Ausnahme
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-Logic-1.7
<b>Name/Zusammenfassung</b>	Positiver Fall beim Überprüfen der Länge des Text der Karte CuT: UserLogic (src/cards/cards-logic.ts)
<b>Setup/Voraussetzungen</b>	1.) Das CardsLogic Objekt ist erzeugt
<b>Testdaten</b>	Ein String mit nur einem der erlaubten Zeichen (",",".",",!","?")
<b>Testschritte</b>	1.) Aufrufen der Funktion doesCardTextContainIllegalCharacters mit dem Testdaten String
<b>Erwartetes Ergebnis</b>	Die Funktion wirft kein Ausnahme

<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-Logic-1.8
<b>Name/Zusammenfassung</b>	Positiver Fall beim Überprüfen der Länge des Text der Karte CuT: UserLogic (src/cards/cards-logic.ts)
<b>Setup/Voraussetzungen</b>	1.) Das CardsLogic Objekt ist erzeugt
<b>Testdaten</b>	Ein String mit nur den Zeichen a bis z und A bis Z
<b>Testschritte</b>	1.) Aufrufen der Funktion doesCardTextContainIllegalCharacters mit dem Testdaten String
<b>Erwartetes Ergebnis</b>	Die Funktion wirft kein Ausnahme
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-Logic-1.9
<b>Name/Zusammenfassung</b>	Positiver Fall beim Überprüfen der Länge des Text der Karte CuT: UserLogic (src/cards/cards-logic.ts)
<b>Setup/Voraussetzungen</b>	1.) Das CardsLogic Objekt ist erzeugt
<b>Testdaten</b>	Ein String mit nur den Zeichen “1” bis “9”
<b>Testschritte</b>	1.) Aufrufen der Funktion doesCardTextContainIllegalCharacters mit dem Testdaten String
<b>Erwartetes Ergebnis</b>	Die Funktion wirft kein Ausnahme
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-Logic-1.9
<b>Name/Zusammenfassung</b>	Negativer Fall beim Überprüfen des Texts einer Karte CuT: UserLogic (src/cards/cards-logic.ts)
<b>Setup/Voraussetzungen</b>	1.) Das CardsLogic Objekt ist erzeugt
<b>Testdaten</b>	Ein String mit einem verbotenen Zeichen
<b>Testschritte</b>	1.) Aufrufen der Funktion doesCardTextContainIllegalCharacters mit dem Testdaten String
<b>Erwartetes Ergebnis</b>	Die Funktion wirft eine Forbidden Ausnahme
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-Logic-1.9
<b>Name/Zusammenfassung</b>	Negativer Fall beim Überprüfen des Texts einer Karte CuT:

	UserLogic (src/cards/cards-logic.ts)
<b>Setup/Voraussetzungen</b>	1.) Das CardsLogic Objekt ist erzeugt
<b>Testdaten</b>	Ein String mit nur mehreren verbotenen Zeichen (Als Beispiel: "\$\$%&)
<b>Testschritte</b>	1.) Aufrufen der Funktion doesCardTextContainIllegalCharacters mit dem Testdaten String
<b>Erwartetes Ergebnis</b>	Die Funktion wirft eine Forbidden Ausnahme
<b>Automatisiert</b>	✓
	Backend-Cards-Logic-1.10
<b>Name/Zusammenfassung</b>	Negativer Fall beim Überprüfen des Texts einer Karte CuT: UserLogic (src/cards/cards-logic.ts)
<b>Setup/Voraussetzungen</b>	1.) Das CardsLogic Objekt ist erzeugt
<b>Testdaten</b>	Ein String mit erlaubten und verbotenen Zeichen
<b>Testschritte</b>	1.) Aufrufen der Funktion doesCardTextContainIllegalCharacters mit dem Testdaten String
<b>Erwartetes Ergebnis</b>	Die Funktion wirft eine Forbidden Ausnahme
<b>Automatisiert</b>	✓

## Testspezifikation Lasttests

<b>Testfall-Nummer/ID</b>	Backend-Cards-Performance-Get-1
<b>Name/Zusammenfassung</b>	Die GetAllCards Funktion wird 60 Sekunden lang mit 4 Threads aufgerufen. Dabei werden immer 100 Karten pro Request zurück geliefert. Die Ramp-Up Zeit beträgt 5 Sekunden.
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer muss eingeloggt sein 2.) Ausführen des Performance Test Setup Skript 3.) Backend starten
<b>Testdaten</b>	1.) Ein registrierter Benutzer 2.) 100 Karten mit beliebigem gültigen Inhalt
<b>Testschritte</b>	1.) Ausführen des Lasttests
<b>Erwartetes Ergebnis</b>	Der Status Code der HTTP Response ist 200 und die Antwortzeit ist ≤ 500 Millisekunden (Bei mehr als 90% der Aufrufe)
<b>Automatisiert</b>	✓

<b>Testfall-Nummer/ID</b>	Backend-Cards-Performance-Create-1
<b>Name/Zusammenfassung</b>	Die Create Funktion wird 60 Sekunden lang mit 4 Threads aufgerufen. Bei jedem Request wird eine neue Karte erstellt. Die Ramp-Up Zeit beträgt 5 Sekunden.
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer muss eingeloggt sein 2.) Ausführen des Performance Test Setup Skript 3.) Backend starten
<b>Testdaten</b>	1.) Ein registrierter Benutzer
<b>Testschritte</b>	1.) Ausführen des Lasttests
<b>Erwartetes Ergebnis</b>	Der Status Code der HTTP Response ist 201 und die Antwortzeit ist ≤ 500 Millisekunden (Bei mehr als 90% der Aufrufe)
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-Performance-Update-1
<b>Name/Zusammenfassung</b>	Die Update Funktion wird 60 Sekunden lang mit 4 Threads aufgerufen. Bei jedem Request wird eine vorhandene Karte aktualisiert. Die Ramp-Up Zeit beträgt 5 Sekunden.
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer muss eingeloggt sein 2.) Ausführen des Performance Test Setup Skript 3.) Backend starten
<b>Testdaten</b>	1.) Ein registrierter Benutzer 2.) 100 Karten, die dem Benutzer zugeordnet sind
<b>Testschritte</b>	1.) Ausführen des Lasttests
<b>Erwartetes Ergebnis</b>	Der Status Code der HTTP Response ist 200 und die Antwortzeit ist ≤ 500 Millisekunden (Bei mehr als 90% der Aufrufe)
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Backend-Cards-Performance-Delete-1
<b>Name/Zusammenfassung</b>	Die Delete Funktion wird 30 Sekunden lang mit 3 Threads aufgerufen. Bei jedem Request wird eine vorhandene Karte gelöscht. Die Ramp-Up Zeit beträgt 2 Sekunden.
<b>Setup/Voraussetzungen</b>	1.) Der Benutzer muss eingeloggt sein 2.) Ausführen des Performance Test Setup Skript 3.) Backend starten
<b>Testdaten</b>	1.) Ein registrierter Benutzer 2.) Eine sehr große Menge an Karten, die dem Benutzer zugeordnet sind (Es werden die erstellten Karten des Backend-Cards-Performance-Create-1 Tests verwendet)
<b>Testschritte</b>	1.) Ausführen des Lasttests

<b>Erwartetes Ergebnis</b>	Der Status Code der HTTP Response ist 200 und die Antwortzeit ist ≤ 500 Millisekunden (Bei mehr als 90% der Aufrufe)
<b>Automatisiert</b>	✓

## Konzeption

Für die Karten wird ein neues Modul erstellt. Über einen Karten Controller sollen alle Aktionen auf Karten gemacht werden. Benutzer dürfen nur auf die von ihnen erstellten Karten zugreifen. Die Karten werden in einer eigenen Datenbanktabelle gespeichert. Um zu überprüfen, ob ein Benutzer auf Karten zugreifen darf, wird das Token, das beim Login erstellt wurde benutzt. Dazu speichert man beim Login das Token in der Datenbank.

Jede Karte hat einen Besitzer (userId), einen Text (text) und eine ID (id). Der Text darf nur alphanumerische Zeichen enthalten sowie die Zeichen:

- , (Komma)
- . (Punkt)
- ! (Ausrufezeichen)
- ? (Fragezeichen)

Des Weiteren darf der Text einer Karte nicht leer sein und darf maximal 250 Zeichen enthalten

Funktionalitäten sind:

- Abrufen aller Karten (eines Benutzers)
- Löschen einer bestimmten Karte
- Updaten einer bestimmten Karte
- Erstellen einer neuen Karte

Alle tests müssen erfolgreich gewesen sein

Funktionalität ist gemerged

Integrationstests erfolgreich



# Liste von Namen für Spieler

⌚ Created	@November 10, 2021 6:14 PM
⌄ Status	Test
⌄ Priority	
👤 Person	
⌄ Bereich	Frontend



# Spiel stoppen

🕒 Created	@November 24, 2021 5:37 PM
➕ Status	Test
➕ Priority	
👤 Person	
➕ Bereich	Frontend



# Spiel-Funktionalität

⌚ Created	@December 21, 2021 3:31 PM
❖ Status	Test
❖ Priority	
👤 Person	
❖ Bereich	Frontend

## Testspezifikation

- Vollautomatisiert
- Halb-Automatisiert
- Manuell

<b>Testfall-Nummer/ID</b>	Frontend-Spielfunktionalität-1
<b>Name/Zusammenfassung</b>	Spiel kann gestartet werden
<b>Setup/Voraussetzungen</b>	Anwendung gestartet
<b>Testdaten</b>	Nutzerdaten, Spielernamen
<b>Testschritte</b>	Einloggen des Nutzers, Spiel starten im Hauptmenü, Eingabe von Spielerdaten
<b>Erwartetes Ergebnis</b>	Spiel startet, erste Karte wird angezeigt
<b>Automatisiert</b>	

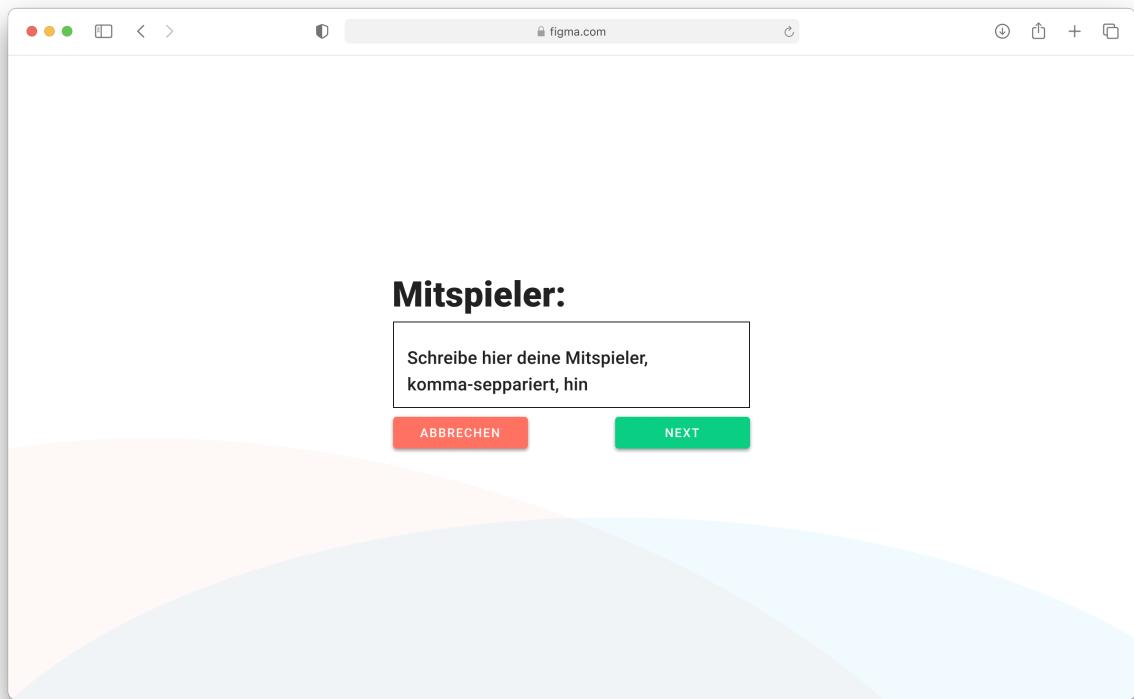
<b>Testfall-Nummer/ID</b>	Frontend-Spielfunktionalität-2
<b>Name/Zusammenfassung</b>	Spiel kann nicht ohne Mitspieler gestartet werden

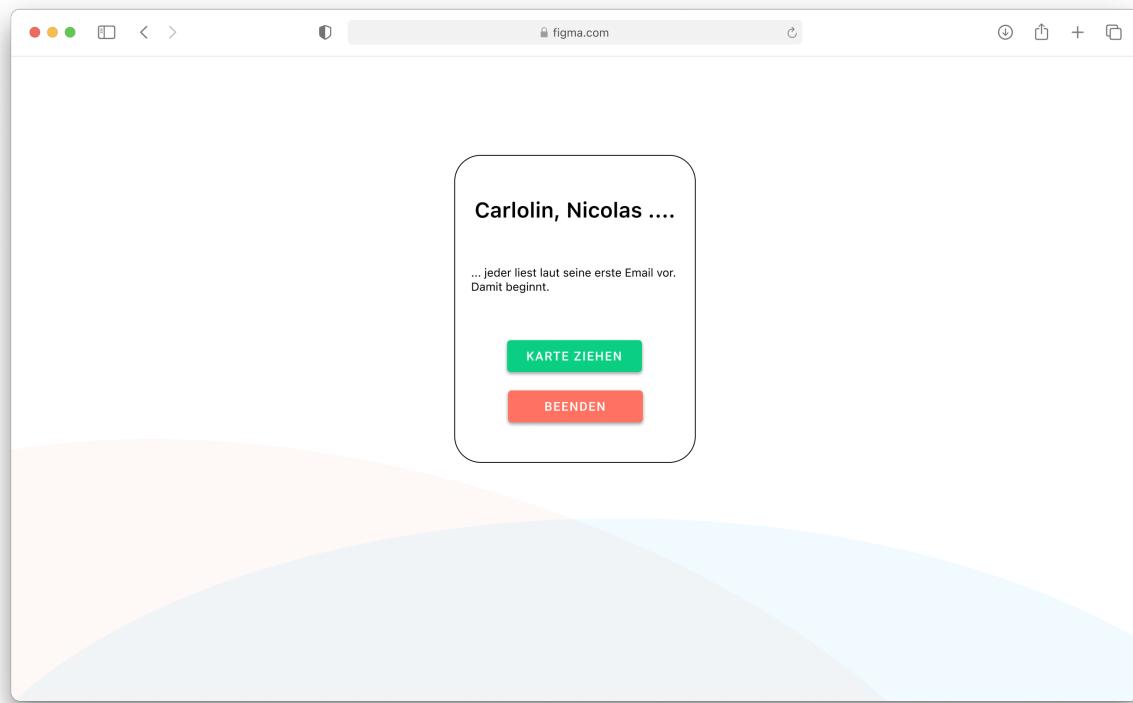
<b>Setup/Voraussetzungen</b>	Anwendung gestartet
<b>Testdaten</b>	Nutzerdaten, Spielernamen
<b>Testschritte</b>	Einloggen des Nutzers, Spiel starten im Hauptmenü, drücken des Weiter-Buttons.
<b>Erwartetes Ergebnis</b>	Feld wird visuell hervorgehoben und es wird nicht zum nächsten Bildschirm navigiert.
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Frontend-Spielfunktionalität-3
<b>Name/Zusammenfassung</b>	Spiel kann beendet werden
<b>Setup/Voraussetzungen</b>	Anwendung gestartet
<b>Testdaten</b>	Nutzerdaten, Spielername
<b>Testschritte</b>	Einloggen des Nutzers, Spiel starten im Hauptmenü, Eingabe von mindestens einem Spielernamen, starten des Spiels, betätigen des Beenden-Buttons
<b>Erwartetes Ergebnis</b>	Navigation zurück zum Hauptmenü
<b>Automatisiert</b>	🟡
<b>Testfall-Nummer/ID</b>	Frontend-Spielfunktionalität-4
<b>Name/Zusammenfassung</b>	Nach dem ziehen einer Karte, wird eine neue Karte angezeigt
<b>Setup/Voraussetzungen</b>	Anwendung gestartet
<b>Testdaten</b>	Nutzerdaten, Spielernamen
<b>Testschritte</b>	Einloggen des Nutzers, Spiel starten im Hauptmenü, Eingabe von mindestens einem Spielernamen, starten des Spiels, betätigen des Karte-Ziehen-Buttons
<b>Erwartetes Ergebnis</b>	Ein neuer Karten-Text wird angezeigt
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Frontend-Spielfunktionalität-5
<b>Name/Zusammenfassung</b>	Es werden Namen angezeigt
<b>Setup/Voraussetzungen</b>	Anwendung gestartet
<b>Testdaten</b>	Nutzerdaten, Spielernamen
<b>Testschritte</b>	Einloggen des Nutzers, Spiel starten im Hauptmenü, Eingabe von mindestens einem Spielernamen, starten des Spiels

<b>Erwartetes Ergebnis</b>	Die Überschrift der Karte sollte mindestens einen Spielernamen anzeigen.
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Frontend-Spielfunktionalität-6
<b>Name/Zusammenfassung</b>	Es wird ein Herausforderungs-Text angezeigt
<b>Setup/Voraussetzungen</b>	Anwendung gestartet
<b>Testdaten</b>	Nutzerdaten, Spielernamen
<b>Testschritte</b>	Einloggen des Nutzers, Spiel starten im Hauptmenü, Eingabe von mindestens einem Spielername, starten des Spiels
<b>Erwartetes Ergebnis</b>	Der Herausforderungs-Text sollte angezeigt werden.
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Frontend-Spielfunktionalität-7
<b>Name/Zusammenfassung</b>	Karte Ziehen Button wird gezeichnet
<b>Setup/Voraussetzungen</b>	Anwendung gestartet
<b>Testdaten</b>	Nutzerdaten, Spielernamen
<b>Testschritte</b>	Einloggen des Nutzers, Spiel starten im Hauptmenü, Eingabe von mindestens einem Spielername, starten des Spiels
<b>Erwartetes Ergebnis</b>	Der Button zum Ziehen einer nächsten Karte sollte zu sehen sein.
<b>Automatisiert</b>	✓
<b>Testfall-Nummer/ID</b>	Frontend-Spielfunktionalität-8
<b>Name/Zusammenfassung</b>	Abbrechen Button wird gezeichnet
<b>Setup/Voraussetzungen</b>	Anwendung gestartet
<b>Testdaten</b>	Nutzerdaten, Spielernamen
<b>Testschritte</b>	Einloggen des Nutzers, Spiel starten im Hauptmenü, Eingabe von mindestens einem Spielername, starten des Spiels
<b>Erwartetes Ergebnis</b>	Der Button zum Abbrechen des Spiels sollte zu sehen sein.
<b>Automatisiert</b>	✓

## Konzeption

# Mockup





- Alle tests müssen erfolgreich gewesen sein
- Funktionalität ist gemerged
- Integrationstests erfolgreich



# Verwalten von Karten

⌚ Created	@January 9, 2022 11:20 AM
❖ Status	Test
❖ Priority	
👤 Person	
❖ Bereich	Frontend

## Testspezifikation

- Vollautomatisiert
- Halb-Automatisiert
- Manuell

Testfall-Nummer/ID	'main-menu'
Name/Zusammenfassung	Das Eingabefeld für eine neue Karte zu erstellen soll angezeigt werden.
Setup/Voraussetzungen	Einloggen eines Nutzers
Testdaten	Nutzerdaten
Testschritte	Navigation zum Karten-Verwalten Screen
Erwartetes Ergebnis	Das Eingabefeld für neue Karten-Texte wird angezeigt
Automatisiert	

Testfall-Nummer/ID	Frontend-Manage_Cards-2
Name/Zusammenfassung	Die Karten des Nutzers werden angezeigt

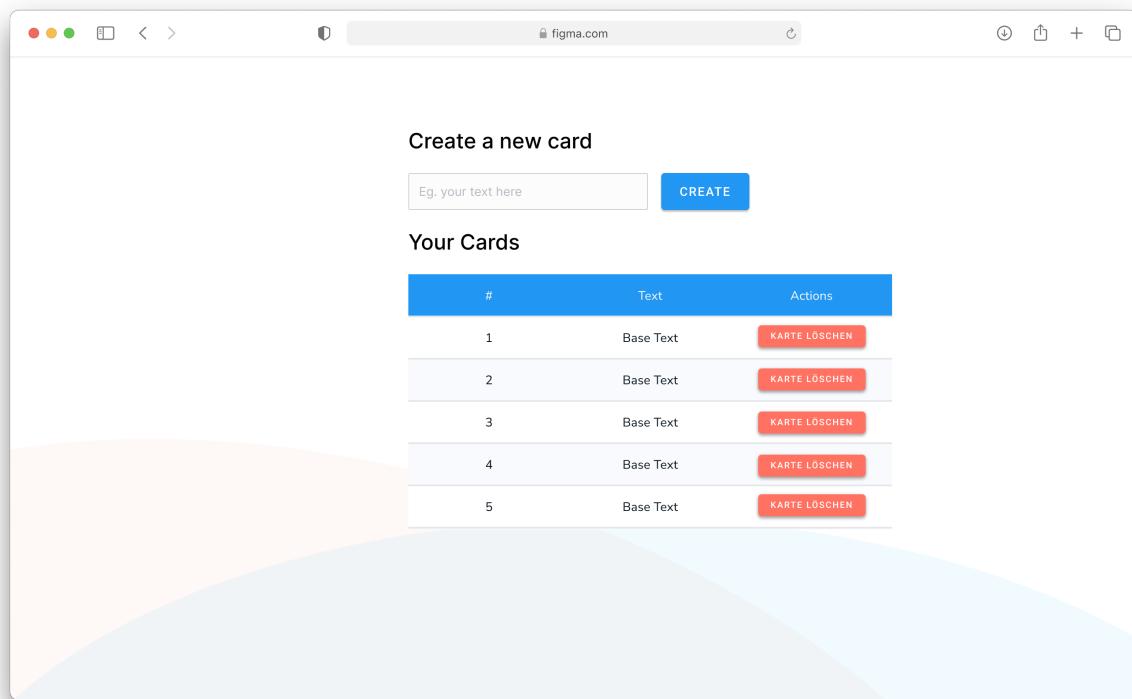
<b>Setup/Voraussetzungen</b>	Einloggen eines Nutzers
<b>Testdaten</b>	Nutzerdaten
<b>Testschritte</b>	Navigation zum Karten-Verwalten Screen, Login des Nutzers in Postmen, Abfrage der Karten des Nutzers, Vergleich ob alle Karten des Nutzers angezeigt werden.
<b>Erwartetes Ergebnis</b>	Dem Nutzer werden all seine Karten in einer Tabelle angezeigt
<b>Automatisiert</b>	
<b>Testfall-Nummer/ID</b>	Frontend-Manage_Cards-3
<b>Name/Zusammenfassung</b>	Löschen einer Karte
<b>Setup/Voraussetzungen</b>	Einloggen eines Nutzers, Erstellung einer Karte für den Nutzer in Postman
<b>Testdaten</b>	Nutzerdaten, Karte
<b>Testschritte</b>	Navigation zum Karten-Verwalten Screen, Betätigung des 'Karte Löschen'-Buttons für die ausgewählte Karte
<b>Erwartetes Ergebnis</b>	Die passende Funktionalität des GameService sollte aufgerufen werden
<b>Automatisiert</b>	
<b>Testfall-Nummer/ID</b>	Frontend-Manage_Cards-4
<b>Name/Zusammenfassung</b>	Erstellen einer neuen Karte
<b>Setup/Voraussetzungen</b>	Einloggen eines Nutzers
<b>Testdaten</b>	Nutzerdaten, Karten Text
<b>Testschritte</b>	Navigation zum Karten-Verwalten Screen, Eingabe des Textes im passenden Input-Feld, Bestätigung des Textes durch den 'Erstellen'-Button.
<b>Erwartetes Ergebnis</b>	Die passende Funktionalität des GameService sollte aufgerufen werden
<b>Automatisiert</b>	
<b>Testfall-Nummer/ID</b>	Frontend-Manage_Cards-5
<b>Name/Zusammenfassung</b>	Erstellen einer neuen Karte ohne Text
<b>Setup/Voraussetzungen</b>	Einloggen eines Nutzers
<b>Testdaten</b>	Nutzerdaten, Karten Text

<b>Testschritte</b>	Navigation zum Karten-Verwalten Screen, Eingabe des Textes im passenden Input-Feld, Bestätigung des Textes durch den 'Erstellen'-Button.
<b>Erwartetes Ergebnis</b>	Der 'Erstellen'-Button sollte ausgegraut und nicht klickbar sein.
<b>Automatisiert</b>	✓

## Konzeption

Der Nutzer soll dem Spiel eigene Karten hinzufügen können. Hierfür benötigt er ein Verwaltungs-Bildschirm. Auf diesem kann er bereits vorhandene Karten anzeigen, diese löschen oder neue Karten hinzufügen.

## Mockup



Alle tests müssen erfolgreich gewesen sein

Funktionalität ist gemerged

Integrationstests erfolgreich



## **Kolophon**

Dieses Dokument wurde mit der L<sup>A</sup>T<sub>E</sub>X-Vorlage für Abschlussarbeiten an der htw saar im Bereich Informatik/Mechatronik-Sensortechnik erstellt (Version 2.1). Die Vorlage wurde von Yves Hary und André Miede entwickelt (mit freundlicher Unterstützung von Thomas Kretschmer, Helmut G. Folz und Martina Lehser). Daten: (F)10.95 – (B)426.79135pt – (H)688.5567pt