

# Verbesserte Gruppenorganisation mit Hilfe einer Gamification-App

---

## ***Masterarbeit***

Marko Dehmel-Dethloff

Betreuer: Dr.-Ing Guido Rößling

Darmstadt, 13.02.2020



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

## **Erklärung zur Master-Thesis**

Hiermit versichere ich, Marko Dehmel-Dethloff, die vorliegende Master-Thesis gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Datum:

Unterschrift:

---

(Marko Dehmel-Dethloff)

## **Zusammenfassung**

Die vorliegende Masterthesis beschäftigt sich damit, wie Gamification eingesetzt werden kann, um die Organisation und Produktivität von Gruppen zu verbessern. Hierfür wurde eine iOS-App entwickelt, mit der Aufgaben innerhalb einer Gruppe erstellt und anschließend von Gruppenmitgliedern bearbeitet werden können. Um die Motivation bei den Beteiligten zu steigern, Aufgaben zu bearbeiten, werden Gamification-Mechaniken eingesetzt. Neben der Vorstellung der eigentlichen Implementierung dieser App soll zudem untersucht werden, welchen Einfluss Gamification schon heute auf die betroffenen Personen hat und welche Auswirkungen insbesondere die hier vorgestellte App auf die Benutzer hat. Hierfür werden zum einen bereits durchgeführte Studien zum Thema Gamification betrachtet und zum anderen wird selbst eine Benutzerstudie durchgeführt, durch die die Auswirkungen der App auf die Benutzer ermittelt werden sollen. Mit Hilfe des erhaltenen Feedbacks wird darüber hinaus diskutiert, um welche Features die App in Zukunft erweitert werden könnte.

## **Abstract**

This master thesis deals with how gamification can be used to improve the organization and productivity of groups. For this purpose, an iOS-App was developed, with which tasks can be created within a group and then edited by the group members. To increase the motivation to work on tasks of those involved gamification mechanisms are used. In addition to the presentation of the implementation of this app, it should also be investigated what influence gamification already has on the people and what effects in particular the presented app has on the users. For this purpose, on the one hand already carried out studies on the topic of gamification are considered and on the other hand a user study itself is carried out to determine the effects of the app on users. The feedback received will also be used to discuss which features of the app could be expanded to in the future.

## Inhaltsverzeichnis

<b>1. Einleitung .....</b>	<b>1</b>
<b>2. Verwendete Methoden und verwandte Arbeiten.....</b>	<b>3</b>
<b>2.1 Gamification .....</b>	<b>3</b>
2.1.1 Begriffserklärung.....	3
2.1.2 Begriffsabgrenzung .....	3
2.1.3 Spielertypen .....	4
2.1.4 Auswirkungen von Gamification .....	5
2.1.5 Chancen und Gefahren .....	5
<b>2.2 Die Entwicklungsumgebung Xcode.....</b>	<b>7</b>
<b>2.3 Die Programmiersprache Swift .....</b>	<b>8</b>
<b>2.4 Das Framework CloudKit.....</b>	<b>8</b>
<b>2.5 Verwandte Arbeiten.....</b>	<b>9</b>
2.5.1 Marktanalyse: erfolgreiche Umsetzung von Gamification in Apps.....	9
2.5.2 Marktanalyse: Apps mit ähnlichen Zielen .....	11
<b>3. Technische Grundlagen von Swift .....</b>	<b>14</b>
<b>3.1 Variablen, Konstanten und Zuweisungen .....</b>	<b>14</b>
3.1.1 Variablen und Konstanten .....	14
3.1.2 Die Schlüsselwörter „static“ und „weak“.....	14
3.1.3 IBOutlet und IBAction .....	15
<b>3.2 Datentypen und Eigenschaften .....</b>	<b>16</b>
3.2.1 Die Datentypen Array, Boolean, Integer und Strings .....	16
3.2.2 Der Datentyp CKRecord .....	16
3.2.3 Der Datentyp Any.....	16
3.2.4 Die Eigenschaft self .....	17
3.2.5 Die Eigenschaften Optional “?” und Implicitly Unwrapped Optional “!” .....	17
3.2.6 Typumwandlung (as-Operator).....	18
<b>3.3 Kontrollstrukturen .....</b>	<b>18</b>
3.3.1 If-Else Anweisungen .....	18
3.3.2 Schleifenkonstrukte .....	19
<b>3.4 Vererbung .....</b>	<b>19</b>
3.4.1 UIViewController.....	19
3.4.2 UITextFieldDelegate .....	20
3.4.3 UITableViewDelegate .....	20
3.4.4 UITableViewDataSource .....	21
<b>3.5 Funktionen .....</b>	<b>21</b>
3.5.1 viewDidLoad() und viewWillAppear() .....	22
3.5.2 CloudKit-Funktionen .....	22
3.5.3 Funktionen aus anderen Klassen aufrufen.....	24
<b>4. Design.....</b>	<b>26</b>
<b>4.1 Anforderungen an das Frontend der App .....</b>	<b>26</b>
4.1.1 Grundsätzliche Anforderungen an das User Interface.....	26
4.1.2 Bilder und Icons .....	27
4.1.3 Navigieren durch die App.....	27

4.1.4	Anlegen von Ereignissen .....	28
4.2	<b>Gamification-Mechaniken</b> .....	28
4.3	<b>Speicherverwaltung</b> .....	29
<b>5.</b>	<b>Implementierung</b> .....	<b>30</b>
5.1	<b>Implementierung der App unter Berücksichtigung der Anforderungen aus Kapitel 4</b> .....	30
5.1.1	Implementierung des User Interfaces.....	30
5.1.2	Verwendete Bilder und Icons.....	32
5.1.3	Eingesetzte Navigation durch die App .....	33
5.1.4	Anlegen von Ereignissen .....	34
5.1.5	Präsentation der Auszeichnungen .....	34
5.1.6	Benutzte Gamification-Mechaniken .....	35
5.1.7	Eingesetzte Speicherverwaltung.....	39
5.1.8	Resümee des Entwicklungsprozesses .....	42
5.2	<b>Herausforderungen</b> .....	43
5.2.1	Direktes Anzeigen der angelegten Ereignisse .....	43
5.2.2	Verwaltung der Gruppenprofile eines Benutzers .....	45
5.2.3	Dark Mode durch das iOS 13er-Update .....	46
5.3	<b>Implementierung einer iOS-Beispiel-App</b> .....	46
5.3.1	Umfang der Beispiel-App .....	47
5.3.2	Anlegen eines neuen Projektes.....	47
5.3.3	Befüllen der Screens mit Inhalten.....	48
5.3.4	Implementierung der Logik der App .....	49
<b>6.</b>	<b>Aufbau und Ergebnisse der Benutzerstudie</b> .....	<b>54</b>
6.1	<b>Aufbau, Durchführung und Teilnehmer der Benutzerstudie</b> .....	54
6.1.1	Aufbau der App für die Benutzerstudie .....	54
6.1.2	Aufbau und Ablauf der Benutzerstudie .....	55
6.1.3	Anonymisierung der Ergebnisse.....	55
6.1.4	Teilnehmer der Benutzerstudie .....	56
6.1.5	Aufbau des Fragebogens.....	56
6.2	<b>Ergebnisse der Benutzerstudie</b> .....	57
6.2.1	Demographische Eigenschaften der Teilnehmer .....	58
6.2.2	Bedienbarkeit der App .....	58
6.2.3	Verwendete Gamification-Mechaniken.....	59
6.2.4	Auswirkungen der App auf die Benutzer .....	60
6.3	<b>Interpretation der Ergebnisse der Benutzerstudie</b> .....	61
<b>7.</b>	<b>Zusammenfassung und Ausblick</b> .....	<b>63</b>
	Abbildungsverzeichnis.....	65
	Literatur.....	67
	Anhang.....	70

## 1. Einleitung

34,33 Millionen US-Dollar – diese Summe wurde 2019 bei einem in Shanghai, China ausgetragenen eSport-Turnier an die Teilnehmer ausgezahlt [20]. Hinter dem Begriff eSport verbirgt sich der sogenannte elektronische Sport. Bei dieser Art von Sport messen sich Menschen mit Hilfe von Videospielen in unterschiedlichen Disziplinen. Waren es 2016 noch etwa 281 Millionen Menschen weltweit, die sich für diese recht neue Form des Sports interessierten, sollen es im Jahr 2022 bereits nahezu 650 Millionen sein [43]. Es wird prognostiziert, dass im gleichen Zeitraum der Gesamtumsatz des eSport-Marktes von 493 Millionen auf fast 1,8 Milliarden US-Dollar ansteigen wird [44]. Zum Vergleich: Die 36 Fußballclubs der 1. und 2. Bundesliga haben in der Saison 2016/17 einen Umsatz von 4,01 Milliarden Euro erwirtschaftet [16]. Zwar übertrifft der deutsche Profifußball die eSport-Szene in dieser Hinsicht noch deutlich, dennoch zeigen diese Zahlen eindrucksvoll auf, mit welcher Geschwindigkeit sich der eSport gesellschaftlich und wirtschaftlich entwickelt.

Wurden Videospieler früher noch meistens als übergewichtige, sozial unverträgliche Außenseiter abgestempelt, ist das Videospielen heutzutage in der Mitte der Gesellschaft angekommen. Mit 34,3 Millionen Gamern und einem Durchschnittsalter von 36 Jahren untermauert Deutschland im Jahr 2019 diese These eindrucksvoll [42]. Weder in Deutschland noch sonst wo auf der Welt gibt es wohl ein vergleichbares Phänomen, das junge und alte Menschen, männlich oder weiblich, unabhängig von ihrem sozialen Stand für sich begeistern kann. Diese Tatsache haben viele Unternehmen längst erkannt und setzen spieltypische Elemente ein, um mehr Produkte zu verkaufen oder die Produktivität ihrer Mitarbeiter zu steigern. Der Einsatz von Spielmechaniken in einem spielfremden Kontext wird in der Literatur als Gamification bezeichnet. Gamification verfolgt dabei im Grunde genommen nur ein Ziel: die Motivation der beteiligten Personen soll für unbeliebte oder monotone Aufgaben gesteigert werden. Häufig resultiert der Einsatz von Gamification darin, dass Ziele effizienter erreicht werden und die Ergebnisse positiver ausfallen.

In dieser Masterthesis sollen die Auswirkungen von Gamification auf alltägliche Aufgaben untersucht werden. Aus diesem Grund wurde im Rahmen dieser Thesis eine Smartphone-Applikation, kurz App, entwickelt, die den Gamification-Ansatz verfolgt. Bei der App handelt es sich um eine reine iOS-App, welche in der Programmiersprache Swift implementiert wurde. Sie ermöglicht den Benutzern, Gruppen zu erstellen, in denen die Teilnehmer wiederum Aufgaben erstellen können. Mit Hilfe von Gamification sollen die Benutzer motiviert werden, diese Aufgaben anschließend zu bearbeiten. An diesem Punkt stellt sich natürlich die Frage, ob die App positive und wenn ja, wie starke Auswirkungen auf die Teilnehmer hat. Diese Frage soll im Verlauf dieser Masterthesis beantwortet werden. Hierfür wurde eine Benutzerstudie durchgeführt, bei der die Teilnehmer die App testen konnten und anschließend ihre Eindrücke und Erfahrungen, mit Hilfe eines Fragebogens, dokumentiert wurden.

Am Anfang dieser Arbeit werden die verwendeten Methoden beschrieben und der aktuelle App-Markt wird im Hinblick auf andere Gamification-Apps untersucht. Anschließend wird im dritten Kapitel auf die technischen Grundlagen eingegangen. Hier werden einerseits bestimmte Eigenheiten der Programmiersprache Swift aufgezeigt, andererseits aber auch wichtige Funktionen beschrieben. Wie mit der verwendeten Datenbank kommuniziert wird, wird ebenfalls in diesem Kapitel erklärt. Im folgenden Kapitel werden die Designentscheidungen besprochen, welche bei der Entwicklung der App zu treffen waren. Das beginnt bei optischen Designentscheidungen, etwa welche Farben verwendet werden sollen, setzt sich fort bei Gamification-Designentscheidungen, welche spieltypischen Elemente eingesetzt werden sollen, und endet bei der Wahl, wie und wo die Daten gespeichert werden sollen. Das fünfte Kapitel befasst sich mit der eigentlichen Implementierung der App. Hier soll anhand einiger Beispiele skizziert werden, wie die App letztendlich programmiert wurde. Im sechsten Kapitel wird der Aufbau und die Durchführung der bereits oben erwähnten Benutzerstu-

die beschrieben und die so gewonnenen Ergebnisse diskutiert. Im letzten Kapitel erfolgt eine Zusammenfassung der wichtigsten Punkte dieser Arbeit. Zudem wird ein Ausblick gegeben, wie es mit der hier entwickelten App weitergehen könnte und welche Auswirkungen Gamification im Allgemeinen auf die Gesellschaft in Zukunft haben könnte.

## 2. Verwendete Methoden und verwandte Arbeiten

In diesem Kapitel werden zu Beginn die verwendete Methode Gamification, sowie die Entwicklungstools Xcode, Swift und CloudKit vorgestellt und im Anschluss verwandte Arbeiten untersucht. Bei der Betrachtung von verwandten Arbeiten werden zum einen allgemein Apps vorgestellt, die Gamification verwenden. Zum anderen wird der aktuelle App-Markt nach Apps durchsucht, die gleiche oder ähnliche Ziele verfolgen, wie die hier entwickelte App.

### 2.1 Gamification

Dieser Abschnitt befasst sich mit der Vorstellung von Gamification. Anfangs findet eine grundlegende Einführung des Begriffs Gamification statt. Anschließend wird der Begriff zu verwandten Begriffen abgegrenzt, unterschiedliche Spielertypen werden vorgestellt und die positiven Auswirkungen von Gamification durch Studien belegt. Am Ende dieses Unterkapitels wird auf die Chancen und Gefahren von diesem Ansatz eingegangen.

#### 2.1.1 Begriffserklärung

Bei dem Begriff Gamification handelt es sich um ein englisches Kunstwort, welches sich aus dem Wort *Game* und dem Suffix *ification* zusammensetzt. Ins Deutsche übersetzt werden kann es mit Spielfizierung. Diese Übersetzung wird allerdings auch im deutschsprachigen Raum so gut wie nie verwendet, weshalb auch in der folgenden Arbeit das Wort Gamification gebraucht wird. Wie die deutsche Übersetzung bereits vermuten lässt, beschreibt Gamification den Vorgang, einen spielfremden Kontext mit Spielementen zu infizieren. Es werden also spieltypische Elemente, wie Erfahrungspunkte, Ranglisten oder Auszeichnungen in einem anderen Zusammenhang eingesetzt [15]. Beispielsweise könnte der Besitzer einer Autofertigungsstraße seinen Mitarbeitern für jedes produzierte Fahrzeug ein paar Punkte geben. Die verschiedenen Fertigungsteams hätte somit einen spielerischen Anreiz, produktiver zu arbeiten als die anderen Teams, damit ihr Team möglichst weit vorne in einer geführten Rangliste zu finden ist. Durch diese recht simple Maßnahme könnte der Besitzer der Fertigungsstraße die Produktivität erhöhen, wodurch alle Beteiligten finanziell profitieren würden.

Natürlich bergen solche Maßnahmen ebenfalls Gefahren in sich, auf die in Abschnitt 2.1.5 näher eingegangen wird. Dennoch ist anhand dieses Beispiels das grundsätzliche Ziel von Gamification gut zu erkennen: die Motivation der Betroffenen soll durch die eingesetzten Spielemente erhöht werden [15], insbesondere in Situationen, in denen unbeliebte oder monotone Aufgaben bearbeiten werden müssen. Zwar ist der Begriff Gamification gerade erst dabei, sich bei der breiten Bevölkerung zu etablieren, jedoch wurde er bereits vor fast 40 Jahren das erste Mal gebraucht. Im Jahr 1978 bezeichnete der britische Computerspielautor und Forscher Richard Allan Bartle mit diesem Wort das Spiel, das sein damaliger Kommilitone programmiert hatte. Er war der Meinung, dass es sich bei dem Programm nicht um ein klassisches Spiel handeln würde, sondern vielmehr um ein Chatprogramm mit spieltypischen Elementen [36]. Diesen Umstand beschrieb er mit dem Wort Gamification.

#### 2.1.2 Begriffsabgrenzung

Häufig wird Gamification mit den verwandten Begriffen Edutainment und Serious Games vertauscht oder vermischt. Deshalb findet im Folgendem eine kurze Abgrenzung zu diesen Begriffen statt. Genauso wie bei Gamification handelt es sich bei Edutainment um ein englisches Kunstwort, das sich aus den Wörtern Education und Entertainment zusammensetzt. Es beschreibt also die Verschmelzung von Lernen (Education) und Unterhaltung (Entertainment), wie es beispielsweise bei Lernspielen oder Lernfernsehserien wie der Sesamstraße der Fall ist [34]. Im Gegensatz zu

Gamification soll bei Edutainment das Spiel oder eine andere Unterhaltungsform selbst die Motivation des Benutzers erhöhen. Gamification hingegen setzt zwar auch auf spieltypische Mechaniken, allerdings ausdrücklich in einem spielfremden Bereich [15]. Darüber hinaus konzentriert sich Edutainment nur auf das Vermitteln von Lerninhalten, Gamification dagegen kann in nahezu jedem Kontext eingesetzt werden.

Serious Games ist das einzige hier genannte Genre, hinter dem sich ausschließlich Spiele verbergen. Übersetzt man den Begriff ins Deutsche „ernsthafte Spiele“, wird schnell klar, dass es sich auch hier nicht um normale Spiele handelt. Zwar steht das eigentliche Spielerlebnis bei Serious Games durchaus im Vordergrund, jedoch sollen zudem Informationen und Wissen durch sie vermittelt werden [8]. Pedro M. Latorre Andrés et al. bezeichnen Serious Games in ihrem Paper „TimeMesh: Producing and Evaluating a Serious Game“ passend als Spiele, die speziell entwickelt wurden, um die mentalen Fähigkeiten und Fertigkeit des Spielers weiterzuentwickeln [25]. Somit ist im Prinzip jede Simulation unter anderem ein Serious Game, egal, ob der Spieler einen Traktor steuert oder eine Operation am offenen Herzen durchführt. In beiden Fällen eignet er sich durch das Spielen des Spiels neue Kompetenzen an oder verbessert bereits vorhandene. Ähnlich wie es bei Edutainment bereits der Fall war, unterscheiden sich Gamification und Serious Games in der Herangehensweise. Gamification nutzt Spielemente, Serious Games hingegen sind Spiele. Die drei Ansätze verfolgen also alle das gleiche Ziel: sie wollen die Motivationen bei den betroffenen Personen steigern. Sie unterscheiden sich lediglich in der Art und Weise, wie sie dies erreichen wollen und in den Gebieten, in denen sie eingesetzt werden können.

### **2.1.3 Spielertypen**

Beim Einsatz von Gamification müssen einige Punkte berücksichtigt werden. Dem Anwender von Gamification muss bewusst sein, dass sich nicht jeder Mensch im gleichen Maße oder überhaupt von diesem Ansatz motivieren lässt. Bei Menschen, die kein Interesse an Videospielen haben, wird Gamification in der Regel nichts bewirken. Darüber hinaus lässt sich nicht jeder Spieler gleichermaßen von Ranglisten und anderen Spielementen motivieren. Die einen bevorzugen das Sammeln von Auszeichnungen und andere den Wettkampf gegen andere Spieler. In der Literatur wird prinzipiell zwischen den vier Spielertypen Achiever, Competitor, Explorer und Socialiser unterschieden [14], wobei an dieser Stelle erwähnt werden muss, dass jeder Spieler auch mehreren oder gar allen Typen zugeordnet werden kann. Die vier Spielertypen werden im Folgenden vorgestellt.

#### **Achiever**

Die Spieler, die dem Spielertyp Achiever zugeordnet werden, zeichnen sich nach ihrem Drang aus, möglichst viele Auszeichnungen zu erhalten. Häufig stecken sie sich selbst spielbezogene Ziele, etwa das Erhalten von allen Auszeichnungen einer bestimmten Kategorie, die sie erreichen wollen [14]. Für Auszeichnungen nehmen sie in Kauf, mit anderen Spielern zusammen zu spielen oder sich mit ihnen zu messen. Ansonsten ist für sie die Interaktion mit anderen Spielern eher von geringerer Bedeutung.

#### **Competitor**

Hinter dem Spielertyp Competitor stecken Spieler, die sich gerne mit anderen Spielern in Wettkämpfen messen. Sie lieben es, ihre Mitspieler zu dominieren und ziehen ihre Motivation hauptsächlich aus der Demütigung ihrer Mitspieler [14]. Sie lassen sich in der Regel besonders gut durch Ranglisten oder direkten Konkurrenzkampf motivieren. An einer Zusammenarbeit mit anderen Spielern sind sie nur dann interessiert, wenn sie sich selbst davon einen Vorteil erhoffen. Für diesen Spielertyp ist es somit zwingend erforderlich, dass weitere Spieler am Spiel teilnehmen.

## Explorer

Bei Explorern handelt es sich um Spieler, die große Freude daran haben, ihre Umwelt zu erkunden [14]. Im Gegensatz zu den zwei bereits genannten Spielertypen verspürt der Explorer überhaupt kein Druck, irgendetwas möglichst schnell oder gut zu erledigen. Vielmehr interessiert diese Spieler, was sich hinter der nächsten Ecke befindet oder wie weit in eine Richtung gelaufen werden kann, bis der Rand der Spielwelt erreicht wird. Für monotone, sich ständig wiederholende Aufgaben hat dieser Spielertyp nicht viel übrig.

## Socialiser

Die Socialiser bilden das Gegenstück zu den Competitor-Spielertyp. Für sie steht die Zusammenarbeit mit anderen Spielern an erster Stelle. Der Austausch mit ihren Mitspielern ist für sie nicht nur Mittel zum Zweck, sondern sie empfinden große Freunde daran, in Chats und Foren miteinander zu kommunizieren oder anderen Spieler zu helfen [14]. Das Wohlergehen der Gruppe stellen sie dabei stets über das eigene. Genauso wie Competitors kann dieser Spielertyp nur dann auftreten, wenn mehrere Spieler beteiligt sind.

### 2.1.4 Auswirkungen von Gamification

Nachdem der Gamification-Ansatz bis zu diesem Punkt ausschließlich theoretisch betrachtet wurde, soll nun aufgezeigt werden, welche Auswirkungen Gamification schon heutzutage in der Praxis hat. An der Hochschule für angewandte Wissenschaften Augsburg wurde dieses Thema bereits untersucht [23]. Zu diesem Zweck wurden insgesamt 56 Studien in acht verschiedenen Anwendungskontexten betrachtet. Bei den Kontexten handelt es sich um die Bereiche Arbeit, Bildung, Crowdsourcing, Datenerhebungen und Umfragen, Gesundheit, Online Communities und soziale Netzwerke, Marketing sowie Umweltschutz. Mit 17 Studien war der Bereich Bildung am prominentesten und Marketing mit lediglich zwei Studien am schwächsten vertreten. Die Ergebnisse der einzelnen Studien wurden einer der drei Kategorien „positive Wirkung“, „gemischte Wirkung“ oder „negative Wirkung“ zugeordnet.

Die Auswertung der Untersuchung ergab, dass von den 56 betrachteten Studien 41 Gamification eine positive Auswirkung attestieren. In den Bereichen Gesundheit und Umweltschutz wurde sogar jede Studie in diese Kategorie eingeordnet. 15 der betrachteten Studien wurden in die Kategorie gemischte Wirkung eingeordnet. Keine einzige hingegen kam zum Ergebnis, dass Gamification negative Auswirkungen auf die Testpersonen hatte.

Weitere interessante Befunde der Untersuchung sind einerseits die verwendeten spieltypischen Elemente, die in den Studien eingesetzt wurden. So wurden Auszeichnungen, der Einsatz eines Punktesystems und Bestenlisten mit Abstand am häufigsten verwendet. Team-Bestenlisten und Zeitdruck wurden eher selten eingesetzt. Darüber hinaus kommt der Urheber der Untersuchung zu dem Ergebnis, dass sich sowohl das Alter als auch der Grad der Vertrautheit im Umgang mit Videospielen auf die Nutzung von Gamification auswirkt. Insgesamt wird durch die Untersuchung also deutlich, dass Gamification in vielen verschiedenen Kontexten positive Auswirkungen auf die beteiligten Personen hat. Besonders bemerkenswert ist zudem die Tatsache, dass nicht eine einzige Studie Gamification negative Auswirkungen zuschreibt.

### 2.1.5 Chancen und Gefahren

Im letzten Unterkapitel zu Gamification wird auf die Chancen und Gefahren eingegangen, die diese in sich birgt. Begonnen wird mit den Chancen.

## Chancen

Wie die gerade genannte Untersuchung unter anderem zeigt, wirkt sich Gamification auf die betroffenen Personen vorrangig positiv aus. So fällt es Schülern leichter, sich fürs Lernen zu motivieren oder Arbeiter sind plötzlich viel produktiver. Der wohl größte Vorteil von Gamification ist die Tatsachen, dass sich so viele Menschen von ihr motivieren lassen. Mit einem Verhältnis von 51% zu 49% gibt es unter den deutschen Gamern gerade einmal 2% mehr männliche Videospiele als weibliche [46]. Wird das Alter der Gamer betrachtet, so ist festzustellen, dass der Anteil in allen Altersklassen von 18 Jahren bis 54 Jahren nahezu gleich groß ist. Lediglich die Altersklasse 25 bis 34 Jahre liegt mit 29% knapp zehn Prozent über dem Schnitt [45]. Diese Zahlen zeigen, dass per se keine Gesellschaftsgruppe in Deutschland von Gamification ausgeschlossen wird.

Ein weiterer Vorteil besteht darin, dass Gamification im Grunde genommen in fast jedem Kontext eingesetzt werden kann. Es spielt dabei keine Rolle, ob sie Leute zum Lernen, Sport oder Arbeiten motivieren soll. Richtig eingesetzt wird sie sich in jedem Bereich positiv auswirken können. Darüber hinaus ist die Umsetzung von Gamification in der Regel vergleichsweise simpel und kostengünstig. Wie im eingangserwähnten Beispiel mit der Fertigungsstraße kann eine einfache Rangliste bereits ausreichen, um die Produktivität der Angestellten zu erhöhen. Natürlich kann der Gamification-Ansatz auch beliebig komplex implementiert werden; dies ist bei jedem Einsatz neu zu beurteilen. In den meisten Fällen wird allerdings der Einsatz von rudimentären Spielementen bereits positive Auswirkungen mit sich bringen.

Durch die Anwendung von Gamification ist es dem Betreiber zudem möglich, das Verhalten der Benutzer spielerisch zu seinem Vorteil zu beeinflussen. Die beeinflussten Personen werden in den meisten Fällen vermutlich nicht einmal merken, dass sie manipuliert werden, da sie der Einsatz von Gamification an ein Spiel erinnern wird und sie damit etwas positives verbinden. Die Arbeiter in der Fertigungsstraße werden sich durch Spielemente wahrscheinlich besser motivieren lassen, als durch eine Ansage vom Chef, der ihnen befehlt, härter zu arbeiten. Eine nicht zu unterschätzende Gegebenheit von Gamification ist außerdem der Umstand, dass von ihr alle beteiligten Personen gleichbehandelt werden. Dadurch, dass die Spieler häufig durch virtuelle Avatare repräsentiert werden, spielen Eigenschaften wie das Geschlecht, die Hautfarbe und Herkunft keine Rolle. Alle müssen sich an die gleichen aufgestellten Regeln halten. So kann durch Gamification im Normalfall niemand bevorzugt werden. Auf den ersten Blick spricht also eigentlich nichts dagegen, Gamification in jeder erdenklichen Situation einzusetzen. Wird Gamification jedoch falsch eingesetzt, so kann dies durchaus negative Folgen für die beteiligten Personen haben. Welche Gefahren dann drohen, wird im nächsten Abschnitt beschrieben.

## Gefahren

Wie im Abschnitt Chancen bereits erwähnt, kann Gamification in vielen Bereichen eingesetzt werden. Es gibt allerdings Situationen, in denen sie besser nicht eingesetzt werden sollte, da sie in diesen Fällen von der Ernsthaftigkeit der Situation ablenken kann. Beispielsweise bei kritischen Militäreinsätzen oder Operationen am Menschen sollte der Einsatz von Gamification vermieden werden. Wird in solchen Fällen dennoch Gamification verwendet, kann es vorkommen, dass Entscheidungen nicht rational, sondern im Sinne des Spiels getroffen werden. Dies könnte fatale Auswirkungen haben. Des Weiteren sollte dem Anwender von Gamification bewusst sein, dass zwar viele, aber eben nicht alle, Spaß an Spielen haben und sich somit nicht jeder durch Gamification motivieren lässt. Deshalb sollte Gamification stets in einem freiwilligen Rahmen stattfinden. Für Personen, die sich nicht an ihr beteiligen wollen, sollten sich daraus keine Nachteile ergeben.

Der wohl schwerwiegendste Kritikpunkt an Gamification ist das Sammeln von sensiblen Daten [1]. Einerseits ist es gerade bei Gamification-Apps in einem sportlichen Kontext erforderlich, dass Da-

ten über die Gesundheit und Fitness der Benutzer gesammelt werden. Andererseits können diese Daten, wenn sie in die falschen Hände gelangen, durchaus auch gegen diese Benutzer verwendet werden. Zum Beispiel könnten Krankenkassen anhand dieser Daten die Höhe der Beiträge ermitteln oder der Betreiber verkauft die gesammelten Daten höchstbietend an andere Unternehmen weiter.

Weiterhin ist zu beachten, dass die durch Gamification hervorgerufene Motivation schnell in Demotivation umschlagen kann. Wird aus der in der Fertigungsstraße eingesetzten Rangliste beispielsweise ersichtlich, welcher Mitarbeiter wie viel in einem bestimmten Zeitraum gearbeitet hat, kann dies massiven Druck auf alle Mitarbeiter ausüben, vor allem auf die, die sich am Ende der Rangliste befinden. In diesem Fall kann Gamification nicht nur negative Folgen für die betroffenen Personen haben, sondern auch für den Betreiber selbst. Die Mitarbeiter könnten aufgrund des so verursachten Drucks anfangen, ihre Aufgaben zwar schneller, aber dafür weniger gut zu erledigen, oder sich aus Scham über ihre schlechte Platzierung gar nicht mehr in die Firma trauen. Ein solches Szenario wäre wohl der Worst Case, der durch den Einsatz von Gamification eintreten kann und sollte unter allen Umständen vermieden werden.

Wie das gerade genannte Beispiel zeigt, muss mit dem Gamification-Ansatz sehr sensibel umgegangen werden. Eine Blaupause für den Einsatz von Gamification kann es deshalb nicht geben. In dem genannten Fall würde eine teilweise Anonymisierung der Rangliste eventuell helfen. Der Grad der Ausprägung von Gamification muss für jede Situation neu bewertet und angepasst werden.

## 2.2 Die Entwicklungsumgebung Xcode

Xcode [6] ist die von Apple bereitgestellte Entwicklungsumgebung für macOS, mit der sich Apps für macOS, iOS, tvOS und watchOS entwickeln lassen. Sie wurde im Jahr 2003 im Rahmen der World Wide Developers Conference (WWDC) vorgestellt und ersetzt seitdem den bis dato verwendeten Project Builder als Entwicklungsumgebung. In erster Linie werden die Programmiersprachen Swift, welche im folgenden Kapitel vorgestellt wird, und Objective-C von Xcode unterstützt. Es ist allerdings auch möglich, andere Sprachen zu verwenden, wie etwa Java, Ruby oder Perl. Apple möchte der breiten Masse ermöglichen, Apps für ihre Produkte zu entwickeln, weshalb die Nutzung von Xcode kostenlos ist. Für die Veröffentlichung von Apps in Apples App-Store wird dann allerdings ein sogenannter Developer-Account benötigt, welcher kostenpflichtig ist.

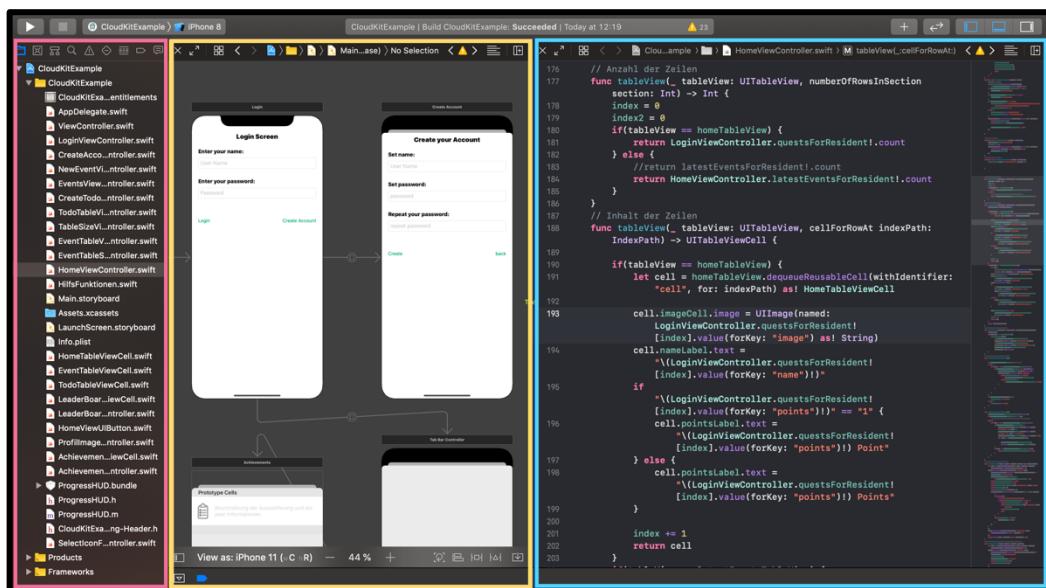


Abbildung 1: Apples Entwicklungsumgebung Xcode ist unterteilt in drei Bereiche: Links: Der Package Explorer.

Mitte: Das Main Storyboard. Rechts: Der Quellcode.

Wie in Abbildung 1 zu sehen ist, lässt sich das Interface von Xcode grundsätzlich in drei Bereiche aufteilen. Diese wurden zur besseren Veranschaulichung pink, gelb und blau umrahmt. Ähnlich wie bei der Entwicklungsumgebung Eclipse, handelt es sich bei dem pink gekennzeichneten linken Bereich um den Package Explorer. Dort werden alle vom Entwickler erstellten Projekte aufgelistet. Mit Hilfe von Drop-Down Menüs können alle verwendeten Dateien und Klassen angezeigt werden. Hauptsächlich dient dieser Bereich dazu, den Überblick über die Projekte zu behalten.

Der im gelb markierten mittleren Bereich zusehende Ausschnitt wird für die meisten Eclipse-Benutzer hingegen neu sein. Dort kann mittels Drag and Drop das Aussehen der App erstellt werden. Egal, ob Buttons, Labels oder Tabellen, alle Elemente können auf diese Weise platziert werden. Im Main Storyboard, so der Name dieses Bereiches, wird jeder App-Screen erstellt, der im fertigen Produkt zu sehen sein soll. In Abbildung 1 wurde beispielsweise das Aussehen eines Login-Screens und eine Account-Erstellung designt. Auch die Übergänge zwischen den einzelnen App-Screens können dort festgelegt werden.

Die eigentliche Logik der App wird anschließend im blauen rechten Abschnitt programmiert. Bei diesem Ausschnitt handelt es sich um eine Swift-Klasse, die einem App-Screen zugeordnet wurde. In dieser Klasse wird das komplette Verhalten des zugehörigen Screens implementiert. Die Logik jedes Buttons, jedes Übergangs und jeder Tabelle muss in diesen Klassen programmiert werden, ansonsten wird sich die App nicht wie gewünscht verhalten. Jedem Screen wird auf diese Weise sozusagen Leben eingehaucht.

Damit die Entwickler ihre implementierten Apps testen können, bietet Xcode zudem die Möglichkeit, diese auf Simulatoren zu testen. Die Palette der Simulatoren umfasst dabei Apples komplette Produktreihe. Mit Xcode bietet Apple seinen Entwicklern also ein Komplettpaket, mit dem Apps designt, die Logik implementiert und mit Hilfe von Simulatoren realitätsnah getestet werden kann.

### 2.3 Die Programmiersprache Swift

Als Apple im Jahr 2014 auf der hauseigenen Konferenz WWDC die neue Programmiersprache Swift mit den Worten schnell, modern, sicher und interaktiv vorstellte, konnten die meistens Zuschauer die Begeisterung der vor Ort anwesenden Journalisten und Entwickler wohl nicht nachvollziehen. Swift gilt als Nachfolger von Objective-C [5] und ist eine Programmiersprache für die Betriebssysteme iOS, iPadOS, macOS, tvOS und watchOS. Wie im Abschnitt 2.2 bereits angedeutet, können durch diese Sprache also sämtliche Apple-Apps entwickelt werden. Bei Swift handelt es sich um eine Hochsprache, die laut Chris Lattner, dem Chefarchitekten von Swift, durch die Programmiersprachen Objective-C, Rust, Haskell, Ruby, Python, C# und viele weitere inspiriert wurde [26]. Aus diesem Grund bietet sie unter anderem alle gängigen Datentypen, sowie Kontrollstrukturen, Vererbung und Klassen. Auf den gesamten Umfang von Swift wird in Kapitel 3 genauer eingegangen. Doch warum sah sich Apple gezwungen, einen Nachfolger für die fast 20 Jahre alte Programmiersprache Objective-C zu entwickeln?

Eines von Apples Markenzeichen ist die Einfachheit, mit der sich die Produkte bedienen lassen. Diese Eigenschaft kann Objective-C im Hinblick auf das Programmieren von Apps allerdings nicht nachgesagt werden. Folglich hatte Apple Angst, dass der Strom an für ihre Betriebssysteme entwickelten Apps in Zukunft mehr und mehr abreissen würde [10]. Aus diesem Grund sah sich Apple genötigt, den Entwicklern eine Programmiersprache zur Verfügung zu stellen, die genauso einfach zu bedienen ist wie ihre Geräte.

### 2.4 Das Framework CloudKit

Bei jeder App-Entwicklung stellt sich früher oder später die Frage, an welchem Ort die verwendeten Daten gespeichert werden sollen. Hierfür gibt es heutzutage eine große Auswahl an Lösungsansätzen. Einer davon nennt sich CloudKit [3]. Bei CloudKit handelt es sich um ein von Apple bereit-

gestelltes Framework, mit dem es möglich ist, Daten in der iCloud zu speichern. Es wurde im Jahr 2014 auf Apples World Wide Developers Conference vorgestellt. Durch CloudKit erhält jede App einen eigenen Container, der wiederum aus den drei Datenbanken public Database, private Database und shared Database besteht. Auf die public Database kann jeder zugreifen, auf die private Database nur der Benutzer selbst und die shared Database kann genutzt werden, um Daten mit einer ausgewählten Gruppe zu teilen.

In den einzelnen Datenbanken können sogenannte Datentypen angelegt werden. Datentypen können durch eine Vielzahl von Attributen beschrieben werden und stellen somit das Äquivalent zu den Datentabellen anderer Datenbankstrukturen dar. Den Datentypen können wiederum Records zugeordnet werden. Die Records repräsentieren die eigentlichen Objekte. So könnte ein Datentyp beispielsweise ein Fahrzeug beschreiben und die dazugehörigen Records konkrete Automodelle.

Die Kosten für den Einsatz von CloudKit fallen im Normalfall sehr gering aus. Apple stellt je nach Anzahl der Benutzer, die die entsprechende App verwenden, mehr Speicherplatz und ein höheres Limit von Anfragen pro Sekunden zur Verfügung. Mehr Benutzer bedeuten also mehr kostenlosen Speicherplatz und Anfragen. Erst wenn diese Limits überschritten werden, fallen Kosten für den Entwickler an. Mit CloudKit stellt Apple ein solides Framework zum Speichern von Daten zu einem fairen Preis bereit. Es muss jedoch beachtet werden, dass dieses Framework ausschließlich in Kombination mit Apples iCloud verwendet werden kann. Eine ausführliche Einführung in CloudKit erfolgt in Kapitel 3.5.2.

## 2.5 Verwandte Arbeiten

In diesem Abschnitt findet eine Marktanalyse statt. Am Anfang werden die zwei aktuell erfolgreichsten Apps vorgestellt, die Gamification verwenden. Anschließend wird der Markt nach Apps untersucht, die ähnliche Ziele verfolgen, wie die im Rahmen dieser Masterthesis entwickelten App.

### 2.5.1 Marktanalyse: erfolgreiche Umsetzung von Gamification in Apps

In den verschiedenen App-Stores sind heutzutage bereits viele Apps zu finden, die auf Gamification setzen, um die Benutzer für unterschiedliche Tätigkeiten zu motivieren. Runtastic [41] und Pokémon Go [33] sind wohl die zwei erfolgreichsten Apps, im Bezug auf die Downloadzahlen, wenn es um das Thema Gamification geht. Anhand dieser beiden Beispiele soll dem Leser nähergebracht werden, welches Potenzial und welche Auswirkungen Gamification-Apps auf die Benutzer haben.

#### Runtastic

Bei der Smartphone-App Runtastic [41] dreht sich alles um das Thema Fitness und Laufen. Die App wurde im Jahr 2009 vom gleichnamigen österreichischen Unternehmen Runtastic GmbH entwickelt, und zählt mit mehr als 300 Millionen Downloads, laut eigenen Aussagen, zu den am meisten heruntergeladenen Apps im Bereich Fitness überhaupt [40]. Bei der App handelt es sich um eine Sport-App, die ihre User mit Hilfe des Gamification-Ansatzes zum Sport motivieren möchte. In dieser Arbeit wird dabei ausschließlich auf den Gamification-Part der App eingegangen. Weitere Funktionen, die Runtastic bietet, werden hier nicht betrachtet. Mit Ranglisten bietet die App ein sehr beliebtes Spielement, das in vielen Gamification-Umsetzungen nicht fehlen darf. Abbildung 2 zeigt eine solche Rangliste, in der der Benutzer sehen kann, wie viele Kilometer er im Vergleich zu seinen Freunden in einem bestimmten Zeitraum gejoggt ist. Wie in Kapitel 2.1.3/Competitor bereits erwähnt wurde, eignen sich solchen Ranglisten gut, um einen Wettkampf unter Freunden zu entfachen und somit die Motivation fürs Joggen bei jedem Teilnehmer zu erhöhen.

Durch ein kleines integriertes Netzwerk ist der Benutzer zudem in der Lage, die sportlichen Aktivitäten seiner Freunde zu kommentieren und zu liken. Bei Bedarf kann er sich sogar während seines Laufes von ihnen anfeuern lassen. Eine weitere Spielmechanik, der sich Runtastic bedient, ist das

Erzählen von Geschichten während eines Laufes. Der Benutzer kann zwischen verschiedenen Szenarien wählen, in die sein Lauf eingebunden wird. Beispielsweise handelt ein Szenario von einer Zombie-Invasion. Der Benutzer muss in diesem Fall vor einer Zombiehorde fliehen. Durch die Geschichten wird er einerseits von der eigentlichen Anstrengung abgelenkt, andererseits wird er auf diese Weise viel mehr Spaß am Laufen haben, wodurch er vermutlich lieber und häufiger joggen geht. Für die Competitor-Spieler wird darüber hinaus noch ein weiteres Feature geboten. So ist es dem Benutzer möglich, Strecken, die von einem anderen Benutzer bereits absolviert wurden, selbst zu laufen. Bei dieser Gelegenheit hat er direkt die Möglichkeit herauszufinden, ob er die Distanz in einer kürzeren Zeit zurücklegen kann.

Runtastic ist in erster Linie wohl so erfolgreich, weil die Entwickler verstanden haben, Gamification gepaart mit sozialen Interaktionen sinnvoll einzusetzen. Dies ist wie in Kapitel 2.1.5/Gefahren beschrieben nicht so trivial, wie man meinen könnte, weshalb es zwar viele andere Apps für diesen Anwendungskontext gibt, aber nur wenige, die ähnlich erfolgreich sind wie Runtastic.

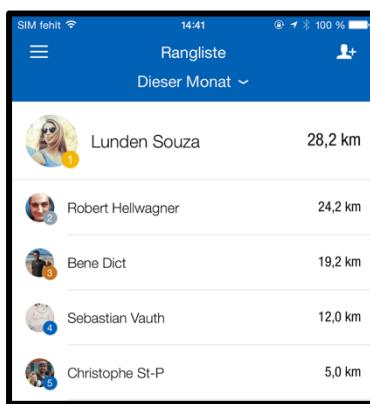


Abbildung 2: Zeigt eine Rangliste in der App Runtastic.

## Pokémon Go

Als Pokémon Go [33] am 6. Juli 2016 vom Entwicklerstudio Niantic, Inc. veröffentlicht wurde, ahnten wohl nur die wenigsten, welchen Erfolg und welche Konsequenzen diese App haben würde. Komplette Städte wurden weltweit von Pokémon Go-Spielern lahmgelegt, auf Autobahnen kam es zu kilometerlangen Staus verursacht durch Unfälle, die auf Grund von Pokémon Go passierten und Parkbesucher wunderten sich über die plötzlich auftretenden Menschenhorden. Mit über einer Milliarde Downloads und einem Umsatz von mehr als 2.65 Milliarden US-Dollar weltweit kann diese App mit Recht als sehr erfolgreich bezeichnet werden [28]. Dabei handelt es sich bei Pokémon Go im Prinzip um nichts anderes als um eine App, die den Benutzer zum Laufen motivieren möchte. Diese eigentliche Intension verschleiert Pokémon Go jedoch gekonnt mit Hilfe von Gamification und Augmented Reality, kurz AR. So werden die Pokémon durch den Einsatz von AR einfach in die reale Welt projiziert. Diese wird dadurch also zur Spielwelt.

Dem Spieler wird gleich zu Beginn des Spiels mitgeteilt, dass er von nun an ein Pokémon-Trainer ist und es somit seine Aufgabe ist, möglichst viele Pokémon zu fangen. Dafür muss er lediglich in seiner Umgebung herumlaufen und stößt so an jeder Ecke über neue Pokémon. Zusätzlich sollen sogenannte Pokémon-Eier den Spieler zum Laufen motivieren. Findet er ein solches Ei in der Spielwelt, muss eine bestimmte Distanz zu Fuß zurückgelegt werden, damit das Ei ausgebrütet wird. Anschließend können aus diesen Eiern seltene Pokémon schlüpfen.

Darüber hinaus sind überall in der Spielwelt Arenen verteilt, die von den drei im Spiel befindlichen Fraktionen erobert werden können. Jeder Spieler kann sich am Anfang einer der drei Fraktionen

anschließen und kämpft von diesem Zeitpunkt an im Namen dieser Fraktion. Niantic setzt also gleich auf mehrere Spielmechaniken, um die Spieler weg vom Sofa und raus in die Natur zu bringen. Mit Erfolg: nach gerade einmal fünf Monaten hatten alle Pokémon Go-Spieler gemeinsam bereits eine Strecke von mehr als 8,7 Milliarden Kilometer zurückgelegt. Diese Distanz entspricht einer 200000-fachen Erdumrundung [37]. Durch das Beispiel Pokémon Go wird erneut deutlich, dass es für den Erfolg einer Gamification-Umsetzung von großer Bedeutung ist, in welchem Setting und in welchem Umfang diese eingesetzt wird. So hat das selbe Entwicklerstudio mit Ingress eine nahezu gleiche App, jedoch in einem anderen Setting, zuvor veröffentlicht. Diese App war allerdings nicht ansatzweise so erfolgreich wie Pokémon Go.



Abbildung 3: Links: Pokémon-Eier, die ausgebrütet werden. Mitte: ein Kampf gegen ein Pokémon. Rechts: Informationen zu einem ausgewählten Pokémon.

Sowohl Runtastic als auch Pokémon Go beweisen, dass Gamification, sinnvoll eingesetzt, schon heute in der Lage ist, die breite Masse zu begeistern und zu motivieren. Nachdem nun zwei sehr erfolgreiche Gamification-Apps vorgestellt wurden, soll es im nächsten Abschnitt um Apps gehen, die ähnliche Ziele verfolgen, wie die im Rahmen dieser Masterthesis implementierte App.

### 2.5.2 Marktanalyse: Apps mit ähnlichen Zielen

Da es sich bei der hier entwickelten App um einen Organizer für WGs und Gruppen handelt, wurden die verschiedenen App-Stores noch einmal explizit nach Apps durchsucht, die Gamification, Todo-Listen oder ähnliche Mechaniken verwenden. Sowohl in Apples App-Store als auch in Googles Play-Store sind viele Apps zu finden, die sich mit diesen Themen beschäftigen. Zur besseren Übersicht wurden die untersuchten Apps in drei Gruppen eingeteilt. In die erste Gruppe werden Apps eingeordnet, die sich mit dem Organisieren von WGs und Gruppen befassen. Der zweiten Gruppe werden Apps zugeordnet, die sich allgemein mit Putzplänen beschäftigen und in der dritten Gruppe befinden sich Apps, bei denen es um Todo-Listen geht. In jeder Gruppe werden zwar die bekanntesten und erfolgreichsten Apps namentlich genannt, allerdings wird keine App im Einzelnen beschrieben, da dies den Umfang der Arbeit sprengen würde. Vielmehr soll kurz erwähnt werden, welche Eigenschaften die einzelnen Gruppen auszeichnen.

#### Apps zum Organisieren von WGs und Gruppen

Flatastic [21], Haus-Segen – Der WG Planer [32], Hubble [49] und Wunderlist [51] sind nur einige von vielen Apps, die ein Komplettpaket fürs WG-Leben bieten und als Organizer für Gruppen dienen wollen. Alle gerade genannten Apps sind zumindest in ihren Grundversion kostenfrei und für

die Betriebssysteme iOS und Android verfügbar. Der Umfang fällt bei den meisten Apps mehr oder weniger gleich aus. So können Putzpläne erstellt, die Finanzen innerhalb der Gruppe verwaltet, Einkaufslisten geteilt und über eine Pinnwand mit den anderen Benutzern kommuniziert werden. Auf Gamification verzichten dabei alle Apps weitestgehend, lediglich Flatastic bietet den Benutzern die Möglichkeit, einer Aufgabe Punkte zuzuordnen. Anhand der Punkte wird anschließend ermittelt, wer als nächstes für ein WG-Todo eingeteilt wird. Wunderlist ist zudem die einzige App, mit der neben einer WG auch beliebige andere Gruppen sinnvoll gemanagt werden können. In dieser App können Todo-Listen erstellt und anschließend mit anderen Benutzern geteilt werden. Jedes Mitglied der Gruppe kann die Liste dann bearbeiten.

### Apps zum Erstellen von Putzplänen

Mit Hilfe der Apps Tody [29] und CleanMyHouse [50] lassen sich Putzpläne erstellen, die den Benutzer automatisch ans Putzen erinnern. Hierfür kann er Putz-Todos anlegen und jeweils angeben, in welchem Intervall er dieses bearbeiten möchte. Die einzelnen Todos können außerdem einem Ort, beispielsweise der Küche oder dem Bad, zugeordnet werden. So sieht der Benutzer auf einem Blick, welche Todos im aktuellen Zimmer bearbeitet werden müssen. Allerdings verzichten auch diese Apps fast komplett auf Gamification. Zwar verwendet die App Tody Fortschrittsbalken, um zu visualisieren, wann die einzelnen Todos wieder bearbeitet werden sollen, weitere Spielelemente werden jedoch nicht eingesetzt. Zudem ist bei beiden Apps keine Interaktion mit anderen Benutzern möglich.

### Apps zum Anlegen von Todo-Listen

Apps, mit denen Todo-Listen angelegt werden können, gibt es zahlreiche. Viele von ihnen setzen dabei jedoch weder auf Gamification, noch bieten sie dem Benutzer die Möglichkeit, mit anderen Usern zusammen zu arbeiten. Eine Ausnahme macht hier die App Habitica [24]. Sie setzt Gamification in hohem Maße ein, um den Benutzer für seine Todos zu motivieren. Nachdem der Benutzer einen Avatar erstellt hat, kann er für diesen Ausrüstung und Erfahrungspunkte sammeln, indem er seine Todos erledigt. So wird sein Avatar mit der Zeit immer stärker und er kann alleine oder gemeinsam mit anderen Spielern gegen Monster kämpfen. Allerdings hat der Benutzer nur Zugriff auf seine eigene Todo-Liste. Es ist nicht möglich, Listen mit anderen Spielern zu teilen.

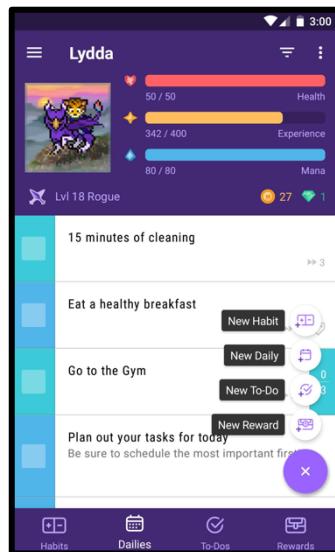


Abbildung 4: Charakteransicht eines Avatars in der App Habitica.

---

In den App-Stores von Apple und Google gibt es also schon viele Apps, die sich mit ähnlichen Themen auseinandersetzen, wie die App, die vom Verfasser dieser Arbeit entwickelt wurde. Es besteht also durchaus Interesse an solchen Apps. Allerdings existiert keine App, die die Themen Todo-Listen, Gamification und das Interagieren mit anderen Benutzern vereint.

Dieses Kapitel diente zur Einführung des Begriffs Gamification sowie der Entwicklungsumgebung Xcode, der Programmiersprache Swift und dem Framework CloudKit. Des Weiteren wurde der derzeitige App-Markt nach verwandten Apps durchsucht und die in dieser Arbeit vorgestellte App in diesen eingeordnet. Das nächste Kapitel dient dazu, dem Leser die Grundlagen und Eigenheiten von Swift näher zu bringen. So wird beispielsweise erklärt, was sich hinter den Operatoren „!“ und „?“ verbirgt. Dieses Wissen ist für den Leser notwendig, um den darauffolgenden Kapiteln besser folgen zu können.

### 3. Technische Grundlagen von Swift

Für die Kapitel 4 und 5 dieser Arbeit werden grundlegende Kenntnisse der Programmiersprache Swift vorausgesetzt, dies gilt insbesondere für die in diesen Kapiteln gezeigten Code-Ausschnitte. Da es sich bei Swift um eine Sprache handelt, von der zwar bereits viele gehört, mit der aber nur wenige in Berührung gekommen sind, werden die Grundlagen dieser Programmiersprache dem Leser in diesem Kapitel nähergebracht. Es ist jedoch von Vorteil, wenn der Leser bereits mit mindestens einer objektorientierten Programmiersprache gearbeitet hat, etwa Java oder C++. Im Übrigen soll dieses Kapitel dabei nicht als Nachschlagewerk für Swift betrachtet werden. Es werden lediglich die Konstrukte und Eigenheiten vorgestellt, die für die Implementierung der App von Relevanz waren. Eine ausführlichere Einführung in die Sprache Swift bietet beispielsweise das Buch „iOS-Apps programmieren mit Swift“ von Christian Bleske [10]. Dieses stellt zudem die Grundlage für dieses Kapitel dar.

#### 3.1 Variablen, Konstanten und Zuweisungen

In diesem Unterkapitel werden mit Variablen, Konstanten und Zuweisungen grundlegende Konstrukte vieler Programmiersprache erläutert.

##### 3.1.1 Variablen und Konstanten

Auch Swift kommt nicht ohne Konstanten und Variablen aus. Der Abbildung 5 kann entnommen werden, wie diese angelegt werden. So wird bei der ersten Codezeile, die blau umrahmt wurde, eine Konstante angelegt und dieser der Wert „4“ zugeordnet. In Swift werden Konstanten mit dem Schlüsselwort `let` erstellt. Anschließend folgt der Name, der Datentyp und der eigentliche Wert, den die Konstante annehmen soll. Nach dem gleichen Schema werden Variablen angelegt, zu sehen in der zweiten Box. In diesem Fall heißt das Schlüsselwort allerdings `var` und nicht `let`. Anhand der dritten Box werden zwei weitere Eigenschaften von Swift ersichtlich. Erstens ist Swift casesensitive, bei den Variablen „name“ und „Name“ handelt es sich also nicht um die selbe Variable, und zweitens muss der Datentyp bei Variablen und Konstanten nicht explizit angegeben werden. Wird dieser nicht mit angegeben, so erkennt Swift diesen eigenständig. Zur besseren Lesbarkeit des Codes sollte der Datentyp jedoch immer mit angegeben werden. Im Unterschied zu Java oder C++ kann zudem auf das Semikolon am Ende der Zeile verzichtet werden.

```
let counter: Int = 4
var name: String = "Liam"
var Name = "Liam"
```

Abbildung 5: Anlegen von einer Konstanten und zwei Variablen in Swift.

##### 3.1.2 Die Schlüsselwörter „static“ und „weak“

In Swift können unter anderem die Schlüsselwörter `static` und `weak` vor Variablen stehen. Genauso wie bei Java weist das Schlüsselwort `static` darauf hin, dass von einer Variablen nur eine Instanz existiert. Alle Klassen und Methoden greifen also mittels „*Klassename*.*Variablename*“ auf die selbe Instanz zu. Somit kann der Wert einer statischen Variablen aus jeder Klasse innerhalb eines Projektes aufgerufen und verändert werden. Das Präfix `weak` wird hingegen immer dann verwendet, wenn es vorkommen kann, dass einer Variablen auch kein Wert zugeordnet wird, sie also den Wert „nil“

(in Java null) annehmen kann. Aus diesem Grund müssen weak-Variablen immer als optional deklariert werden. Was sich hinter der Eigenschaft optional verbirgt, wird in Kapitel 3.2.5 beschrieben. Darüber hinaus kann das Präfix weak auch verwendet werden, um den Speicherplatz des Hauptspeichers optimal zu nutzen. Im Gegensatz zu herkömmlichen Variablen werden weak-Variablen nämlich aus dem Hauptspeicher entfernt, wenn sie nicht benötigt werden, selbst wenn sich ihre Klasse noch in diesem befindet. So schafft Swift mehr Speicherplatz im Hauptspeicher für weiteren Code.

### 3.1.3 IBOOutlet und IBAction

Wie in Kapitel 2.2 bereits beschrieben wurde, wird die grafische Oberfläche der App mit Hilfe des Main Storyboards designt und anschließend die Logik mittels Swift-Klassen implementiert. Doch woher weiß das entsprechende Objekt, dass es gerade von dem Swift-Code angesprochen wird? An dieser Stelle kommen die sogenannten IBOOutlets und IBActions ins Spiel. Das Präfix „IB“ steht dabei für Interface Builder. Mit diesen lassen sich in Xcode Oberflächen und Menüs modellieren. Die IBOOutlets und IBActions sind also eine Art Bindeglied zwischen dem Main Storyboard und den Swift-Klassen. Bei Xcode werden alle Objekte, deren Verhalten im Swift-Code beschrieben wird, wortwörtlich in diesen per Drag and Drop reingezogen und somit eine Referenz zu den entsprechenden Objekten erstellt, zu sehen in Abbildung 6 im oberen Kasten. Anschließend können die Objekte über diese Referenzen angesprochen werden. So kann ein Label auf diese Weise mit einem Wert initialisiert werden, der dann in der App angezeigt wird, wie der unteren Box in Abbildung 6 zu entnehmen ist. In diesem Fall wurde dem Label namens „Label“ der String „Hallo Welt!“ zugeordnet. IBOOutlets können für Labels, Bilder, Textfelder und Tabellen erstellt werden, also für alle Objekte, die irgendeine Art von Inhalt anzeigen.

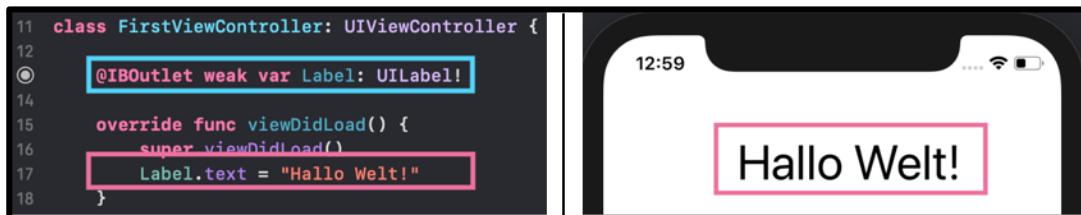


Abbildung 6: Links: Ein Label wurde als IBOOutlet angelegt und ein String wurde diesem zugeordnet. Rechts: Das Label zu sehen in der App.

Für Buttons hingegen werden IBActions benötigt. Diese funktionieren im Grunde genommen nach dem gleichen Prinzip wie IBOOutlets, nur handelt es sich bei IBActions um Methoden, die immer dann ausgeführt werden, wenn der dazugehörige Button gedrückt wurde, siehe Abbildung 7. In dem hier gezeigten Beispiel wurde ein Button mit dem Namen „Back“ angelegt und die dazugehörige Methode bewirkt, dass der Benutzer auf einen anderen Screen der App zurück wechselt.

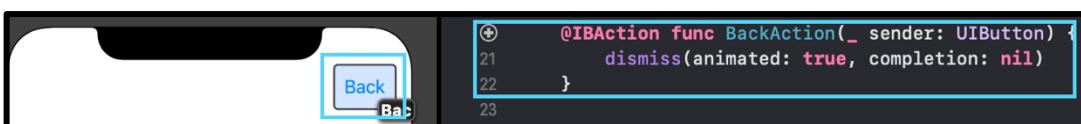


Abbildung 7: Ein Button mit der dazugehörigen Methode.

Natürlich können sowohl bei den IBOOutlets als auch den IBActions noch viel mehr Einstellungen vorgenommen werden. So können durch die IBOOutlet-Referenzen beispielsweise die Schriftart und Schriftgröße festgelegt oder ein Rahmen um Bilder gezogen werden. Da all diese Eigenschaften

ebenfalls direkt in Xcode eingestellt werden können, wird auf eine ausführlichere Beschreibung der IBOutlets und IBActions an dieser Stelle verzichtet.

### 3.2 Datentypen und Eigenschaften

Nachdem sich der letzte Abschnitt mit Konstanten und Variablen befasst hat, wird in diesem Abschnitt auf die Datentypen und Eigenschaften der Instanzen eingegangen, die bei der Implementierung der App verwendet wurden. Allgemeinbekannte Datentypen, wie Strings und Booleans, werden zwar zu Beginn vollständigkeitshalber kurz gezeigt, jedoch liegt der Fokus dieses Kapitels auf den Swift-spezifischen Datentypen.

#### 3.2.1 Die Datentypen Array, Boolean, Integer und Strings

Wie nahezu jede objektorientierte Programmiersprache unterstützt auch Swift die Standard-Datentypen Array, Boolean, Integer und Strings. Das Anlegen von Integer und Strings wurde bereits in Abbildung 5 gezeigt, weshalb diese in Abbildung 8 nicht noch einmal aufgeführt werden. Generell unterscheiden sich die hier aufgezählten Datentypen in ihrer Handhabung kaum gegenüber anderen Programmiersprachen. Lediglich das Deklarieren von Arrays besitzt im Vergleich zu Java und C++ eine leicht veränderte Syntax. So steht der Datentyp eines Arrays bei Swift innerhalb der Klammern und nicht davor (Abbildung 8, blauer Kasten).

```
var Namen: [String] = []
var wahrheitswert: Bool = true
```

Abbildung 8: Deklarieren eines Arrays und Initialisieren einer Booleschen-Variablen.

#### 3.2.2 Der Datentyp CKRecord

Bei CKRecord handelt es sich um einen Datentyp, der nicht zu den Standarddatentypen von Swift gehört, sondern erst nach Importieren des CloudKits verwendet werden kann. Das Präfix „CK“ steht für CloudKit und befindet sich vor jedem Schlüsselwort, das durch dieses Kit importiert wird. Dieser Datentyp repräsentiert die in Kapitel 2.4 beschriebenen Records. Bei Records handelt es sich um die tatsächlichen Objekte der CloudKit-Datenbanktabellen. Dementsprechend ist ein CKRecord eine Sammlung von Schlüssel-Werte Paaren, wobei jeder Schlüssel einem Attribut der Datenbanktabelle und der Wert jeweils der konkreten Ausprägung dieses Attributes entspricht. Mit Hilfe des Befehls *CKRecordName.value(forKey: „AttributName“)* kann auf das Attribut des gewünschten Records zugegriffen werden. Des Weiteren umfasst dieser Datentyp die ID, den Namen und den Zeitpunkt der letzten durchgeföhrten Modifikation des jeweiligen Records. Darüber hinaus stellen CKRecords dem Entwickler noch weitere Optionen zur Verfügung, auf die an dieser Stelle jedoch nicht näher eingegangen wird, da sie bei der Entwicklung der hier programmierten App keine Rolle spielen.

#### 3.2.3 Der Datentyp Any

Streng genommen verbirgt sich hinter dem Schlüsselwort *Any* gar kein Datentyp. Any kann vielmehr als Joker betrachtet werden, der immer dann zum Einsatz kommt, wenn im Vorhinein nicht klar ist, welchen Wert eine Variable annehmen wird. Wird das Schlüsselwort Any anstatt eines konkreten Datentyps bei der Erstellung einer Variablen verwendet, so können dieser Werte eines beliebigen Datentyps zugeordnet werden. Beispielsweise kann eine Variable vom Typ Any zuerst mit einem String initialisiert und anschließend durch einen Integer-Wert überschrieben werden. Auch selbsterstellte Datentypen können einer solchen Variablen zugeordnet werden. Any geht dabei so-

gar so weit, dass Arrays, welche als Any deklariert wurden, Objekte von verschiedenen Datentypen speichern können. So können in einem einzigen Array Objekte vom Typ String, Integer und Boolean gespeichert werden, wie in Abbildung 9 im blauen Kasten zu sehen ist. Es ist allerdings nicht zu empfehlen, aus Bequemlichkeitsgründen jedes Objekt mit dem Schlüsselwort Any zu versehen, da dies zu ungewünschten Seiteneffekten führen kann. Beispielsweise kann in einer Variablen ein String gespeichert worden sein, obwohl diese eigentlich einen Integer enthalten sollte. Über dieses Fehlverhalten des Codes würde der Entwickler in diesem Fall nicht informiert werden. Das Schlüsselwort Any kann sich also als sehr nützlich erweisen, jedoch sollte der Einsatz gut überlegt werden.

```
var MyArray: [Any] = []
let Boolean: Bool = false
MyArray.append(123)
MyArray.append("Hallo Welt!")
MyArray.append(Boolean)

[123, "Hallo Welt!", false]
```

Abbildung 9: Anlegen und Befüllen eines Any-Array in Swift.

### 3.2.4 Die Eigenschaft self

Hinter dem Schlüsselwort *self* versteckt sich eine Eigenschaft, welche das Äquivalent zu Javas *this* bildet. Christian Bleske beschreibt diese Eigenschaft passend in seinem Buch „iOS-Apps programmieren mit Swift“ wie folgt: Jede Instanz eines Typs hat eine Eigenschaft, die den Namen *self* trägt. Über die *self*-Eigenschaft hat der Programmierer Zugriff auf die Methoden und Eigenschaften der Instanz und kann diese aufrufen [10]. In der Regel muss dieses Schlüsselwort jedoch nicht explizit mit angegeben werden. Swift geht dann automatisch davon aus, dass der Programmierer auf Eigenschaften oder Methoden dieser Instanz zugreifen möchte. Zwingend notwendig wird die Angabe von *self* erst, wenn es mehrere Instanzen gibt, die gleich heißen und in unterschiedlichen Scopes angelegt wurden.

### 3.2.5 Die Eigenschaften Optional “?” und Implicitly Unwrapped Optional “!”

In Swift kann beim Anlegen von Variablen und Konstanten direkt hinten dem Datentyp wahlweise ein „?“, ein „!“ oder gar nichts stehen. Diese Zeichen haben Einfluss auf die Eigenschaften der jeweiligen Variablen und Konstanten. Wenn nichts hinter dem Datentyp steht, handelt es sich um eine ganz normale Variable oder Konstante. Befindet sich hingegen ein Fragezeichen hinter dem Datentyp, wie es bei der Variable in Abbildung 10 im ersten Kasten der Fall ist, so weist dies auf eine sogenannte *optional* Variable hin. Optionale Variablen können einen Wert enthalten, müssen dies jedoch nicht zwingend. Sie dürfen also auch den Wert „nil“ annehmen. Beispielsweise soll im zweiten Kasten ein String zu einem Integer gecastet und dieser der Integer-Variable „eineIntVariable“ zugeordnet werden. Wenn es sich bei dem String um eine Zahlenfolge handelt, enthält die Variable im Anschluss den entsprechenden Wert. Enthält der String jedoch eine Zeichenkette, die nicht in einen Integer umgewandelt werden kann, so kann der Variable kein Wert zugeordnet werden. Bei einer normalen Variablen oder Konstante würde dies zu einer Fehlermeldung führen. Der *?*-Operator sollte also immer dann verwendet werden, wenn bei der Deklaration nicht klar ist, ob die Variable oder Konstante zur Laufzeit auch tatsächlich einen Wert enthalten wird.

Der *!*-Operator, auch *Implicitly unwrapped optional* genannt, stellt das Gegenstück zum *?*-Operator dar. Besitzt eine Variable zu Beginn beispielsweise noch keinen Wert, der Entwickler ist sich jedoch

sicher, dass die Variable sobald sie verwendet wird einen Wert enthält, so kann diese mit dem implicitly unwrapped optional-Operator versehen werden. Aus diesem Grund ist es möglich, eine implicitly unwrapped optional Variable ebenfalls mit dem Wert „nil“ zu initialisieren (dritter Kasten).

```
var name: String? = nil  
var eineIntVariable: Int? = Int("123")  
var nachname: String! = nil
```

Abbildung 10: Anlegen von optional und implicitly unwrapped optional Variablen in Swift.

### 3.2.6 Typumwandlung (as-Operator)

Durch den *as*-Operator ist es möglich, die Sichtweise auf eine Variable in Swift zu ändern. So kann beispielsweise ein String in einen Integer umgewandelt werden. Der as-Operator wird dabei häufig in Kombination mit den im letzten Abschnitt vorgestellten optional oder implicitly unwrapped optional Operator verwendet. Je nach eingesetzter Kombination besitzt das Schlüsselwort „as“ andere Eigenschaften. Wird es mit dem ?-Operator versehen, so wird versucht, eine Typumwandlung durchzuführen. Ist diese jedoch nicht möglich, verändert sich der Datentyp der Variable nicht und das Programm wird ohne Fehlermeldung weiter ausgeführt. Anders sieht es aus, wenn der as-Operator mit dem !-Operator erweitert wird. Dann muss in jedem Fall ein Cast erfolgen; wenn das nicht der Fall sein sollte, wird das Programm in der Regel mit einer Fehlermeldung beendet. Wann immer ein solcher Cast von Swift also vorausgesetzt wird, muss der Entwickler sicher gehen, dass dieser auch durchgeführt werden kann.

## 3.3 Kontrollstrukturen

Die allgemein bekannten Kontrollstrukturen, wie Fallunterscheidungen mit if/else oder while-Schleifen, sind auch bei Swift vertreten. Im Folgenden wird erneut nur auf diejenigen Kontrollstrukturen eingegangen, welche bei der Entwicklung der App eingesetzt wurden.

### 3.3.1 If-Else Anweisungen

Analog zu den *If-Else* Anweisungen, die aus anderen Programmiersprachen bekannt sind, verhalten sich diese Anweisungen in Swift. Anhand eines booleschen Ausdrucks wird bestimmt, ob der Code im If- oder Else-Teil ausgeführt wird. Im Gegensatz zu Java oder C++ muss der boolesche Ausdruck jedoch nicht in Klammern geschrieben werden. Die If- und Else-Zweige hingegen müssen jeweils mit geschweiften Klammern umschlossen werden. Ein simples Beispiel einer If-Else Anweisung ist in Abbildung 11 zu sehen.

```
if name == "Liam" {  
    // Führe Anweisung 1 aus.  
} else {  
    // Führe Anweisung 2 aus.  
}
```

Abbildung 11: Eine If-Else Anweisung in Swift. Übrigens: Im Gegensatz zu anderen Sprachen ist es in Swift möglich, zwei Strings mit Hilfe des ==-Operators zu vergleichen.

### 3.3.2 Schleifenkonstrukte

Bei *while*-Schleifen handelt es sich um eine von insgesamt drei Schleifenarten, die von Swift zur Verfügung gestellt werden. Bei dieser wird vor jedem Schleifendurchgang ein boolescher Ausdruck evaluiert. Wird dieser zu true ausgewertet, so wird der Code innerhalb der Schleife erneut ausgeführt. Erst wenn der Ausdruck nicht mehr erfüllt wird, springt das Programm aus der Schleife und bricht diese somit ab.

Die *for*-Schleife stellt den zweiten Schleifentyp dar, der von Swift angeboten wird. Bei diesem wird vor oder nach jedem Schleifendurchlauf eine Variable im Schleifenkopf inkrementiert oder dekrementiert. Anschließend wird überprüft, ob die Bedingung, welche ebenfalls im Schleifenkopf zu finden ist, noch erfüllt wird. Wenn ja, wird die Schleife erneut durchlaufen, ansonsten wird diese abgebrochen.

Die dritte und somit letzte Schleifenvariante wird als *repeat...while*-Schleife bezeichnet. Da diese in dieser Arbeit nicht verwendet wurde, wird auf eine Beschreibung dieser Schleifenvariante an dieser Stelle verzichtet.

## 3.4 Vererbung

Das Rad muss und sollte auch in Swift nicht bei jedem Projekt neu erfunden werden. So gibt es viele Problemstellungen, die häufiger auftreten und für die bereits gute Lösungen entwickelt wurden. Die grundlegende Struktur dieser Lösungen werden in sogenannten Protokollen festgehalten, welche Java-Benutzern besser unter dem Namen Interfaces bekannt sind. Egal ob Protokolle oder Interfaces, hinter beiden Begriffen stecken klassenähnliche Konstrukte, die selbst keine Eigenschaften und ausführbaren Code beinhalten, jedoch von anderen Klassen geerbt werden können. In den erbenden Klassen wird dann der Code des jeweiligen geerbten Protokolls überschrieben, wodurch dessen Methoden ausführbar werden. Um eine Klasse von einem solchen Protokoll erben zu lassen, muss hinter dem Klassennamen ein Doppelpunkt und anschließend eine Liste der Namen der Protokolle folgen, wie in Abbildung 12 gezeigt.

Mit der Mechanik Vererbung stellt Swift den Entwicklern zwar eine zentrale Eigenschaft der objektorientierten Programmierung zur Verfügung, jedoch existiert eine Mehrfachvererbung, wie sie beispielsweise bei C++ möglich ist, bei Swift nicht. Für die Implementierung der App wurden insgesamt vier Protokolle verwendet. Diese werden im Folgenden kurz vorgestellt.

```
class HomeViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {
```

Abbildung 12: Signatur der Klasse HomeViewController, welche von einigen Protokollen erbt.

### 3.4.1 UIViewController

Bei dem UIViewController handelt es sich um ein Protokoll, das bei fast jeder Klasse zum Einsatz kommt. Es verwaltet unter anderem die Views innerhalb der einzelnen Szenen, wobei eine Szene wiederum einem Screen der App entspricht. Mit Hilfe der Abbildung 13 soll diese Funktion des Protokolls genauer erklärt werden. Im rechten Kasten ist die fertige Szene einer App zu sehen. Die einzelnen Bestandteile dieser Szene (Views, Labels, Buttons etc.) wurden im linken Kasten von dem Programmierer hierarchisch aufgelistet. Ist ein Element eingerückt, so bedeutet dies, dass es ein Kind des übergeordneten Elements ist und somit alle Eigenschaften von diesem Element übernimmt. Das Einordnen der Elemente in eine solche Hierarchie erleichtert dem Programmierer das Erstellen einer Szene, da die Eigenschaften nicht für jedes Element einzeln festgelegt werden müssen. Das UIViewController-Protokoll verwaltet an dieser Stelle beispielsweise die Größe und Aus-

richtungen der Inhalte innerhalb einer Szene. So ist es möglich, dass die App auf allen Geräten gleich aussieht, selbst wenn deren Bildschirme unterschiedlich groß sind. Darüber hinaus steuert das Protokoll die Übergänge zu anderen App-Screens, also zu anderen Views.

Des Weiteren stellt dieses Protokoll die Methoden `viewDidLoad()` und `viewWillAppear()` bereit, welche im Abschnitt 3.5.1 genauer vorgestellt werden. An dieser Stelle deshalb nur so viel: diese Methoden sind für den Lebenszyklus einer Anwendung verantwortlich. Sie legen also fest, wann welches Element gerendert beziehungsweise neu geladen wird. Um von `UIViewController`-Protokoll erben zu können, muss die Bibliothek `UIKit` importiert werden.

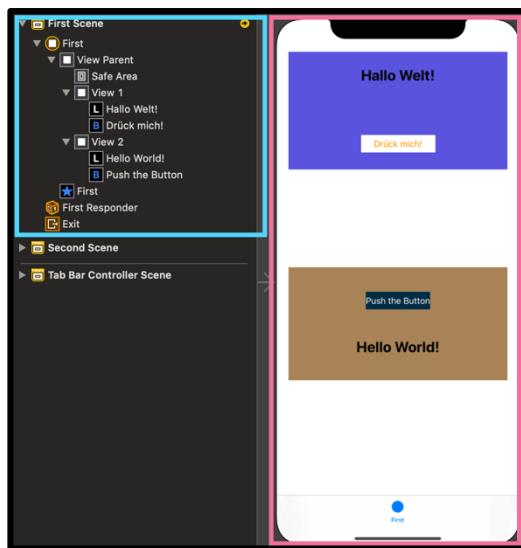


Abbildung 13: Links: Die einzelnen Elemente einer Szene hierarchisch dargestellt. Rechts: Eine fertige Szene.

### 3.4.2 UITextFieldDelegate

Das `UITextFieldDelegate`-Protokoll stellt einige nützliche Methoden zum Thema Textfelder bereit. Während der Entwicklung der App wurde jedoch lediglich die Methode `textFieldShouldReturn` genutzt. Diese ermöglicht es dem Benutzer in der fertigen App, die Tastatur auszublenden, indem dieser auf die Return-Taste klickt. Gerade bei Geräten mit kleineren Bildschirmen kann die Tastatur die Sicht auf wichtige Komponenten verhindern, weshalb diese Funktionalität immer angeboten werden sollte, wenn die Tastatur angezeigt wird.

### 3.4.3 UITableViewDelegate

`UITableViewDelegate` ist eines von zwei Protokollen, die für die Modifikation von `TableViews` verantwortlich sind. Hinter diesen `TableViews` verbergen sich einfache Tabellen, die in verschiedenen Ausprägungen in fast jeder App eingesetzt werden. Durch das Protokoll `UITableViewDelegate` kann unter anderem die Höhe der einzelnen Tabellenzellen eingestellt oder der Header von Sektionen formatiert werden. Des Weiteren bietet das Protokoll eine Methode, durch die Swift weiß, welche Zelle vom Benutzer ausgewählt wurde. Da diese Funktion häufig verwendet wird, ist dieses Protokoll sehr wichtig für Apps, in denen Tabellen eingesetzt werden.

Insgesamt kann festgehalten werden, dass das `UITableViewDelegate`-Protokoll sich primär mit dem optischen Erscheinungsbild von Tabellen beschäftigt. Für den inhaltlichen Part ist hingegen das `UITableViewDataSource`-Protokoll verantwortlich, auf welches im folgenden Abschnitt eingegangen wird.

### 3.4.4 UITableViewDataSource

Bei dem UITableViewDataSource-Protokoll handelt es sich um das zweite in dieser Arbeit vorgestellte Protokoll, das Methoden für Tabellen bereitstellt. Wie im vorangegangenen Abschnitt bereits erwähnt, wird dieses Protokoll hauptsächlich benötigt, um die Tabellen mit Inhalten zu füllen. Hierfür werden zwei Methoden angeboten (siehe Abbildung 14), die implementiert werden müssen, wenn von diesem Protokoll geerbt wird. Die erste Methode gibt eine Zahl zurück (oberer Kasten), welche aussagt, wie viele Zellen die Tabelle besitzen soll. Oder anders ausgedrückt: durch den Rückgabewert der ersten Methode wird bestimmt, wie häufig die zweite Methode (unterer Kasten) ausgeführt werden soll. Durch die zweite Methode wird anschließend nämlich festgelegt, welche Inhalte in den Zellen jeweils angezeigt werden sollen. In dem Beispiel aus Abbildung 14 soll die fertige Tabelle (rechter Kasten) insgesamt fünf Zellen umfassen, wobei in jeder Zelle das Wort „Inhalt“ stehen und darauffolgend eine Zahl von Null hochgezählt werden soll. Dieses Beispiel soll das Zusammenspiel der zwei Methoden verdeutlichen. Analog zu diesen Methoden liefert das Protokoll zwei weitere Methoden, durch die eine Tabelle in Sektionen unterteilt werden kann. Auch bei diesen wird durch eine Methode ermittelt, wie viele Sektionen benötigt werden und durch die zweite Methode, welche Inhalte die Header haben sollen. Durch die Protokolle UITableViewDelegate und UITableViewDataSource wird es Swift-Entwicklern ermöglicht, Tabellen individuell nach ihren Bedürfnissen anzupassen.

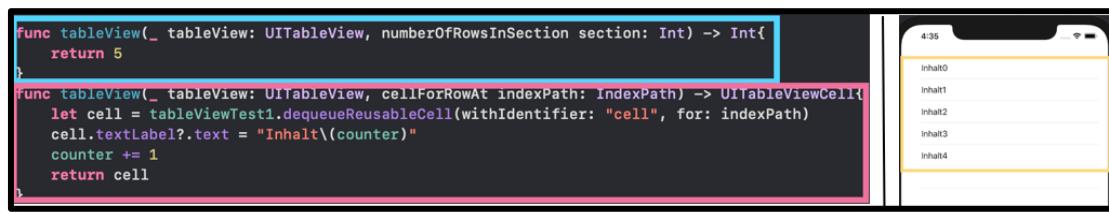


Abbildung 14: Links: Der Quellcode um die Tabelle zu formatieren. Rechts: Die fertige Tabelle.

### 3.5 Funktionen

Der grundsätzliche Aufbau von Funktionen in Swift wurde bereits in einigen Abbildungen zuvor gezeigt. In diesem Kapitel soll zu Beginn jedoch noch einmal explizit auf den Aufbau und die Syntax von Swift-Funktionen eingegangen und anschließend einige wichtige Funktionen vorgestellt werden.

Die Deklarierung einer Funktion wird in Abbildung 15 gezeigt. Funktionen werden immer mit dem Schlüsselwort `func` eingeleitet. Darauf folgt der Name der Funktion und ein Klammernpaar. Durch eine anschließende öffnende geschweifte Klammer endet der Funktionskopf und der Funktionskörper beginnt. Im Fall der Funktion aus Abbildung 15 werden dieser zudem zwei Parameter übergeben, ein String und ein Integer. Die Typen der Parameter müssen genauso wie in Java mit im Funktionskopf angegeben werden. Da die Funktion außerdem einen Wert zurückgeben soll, muss nach den Klammern, in denen die Parameter stehen, ein Pfeil und der Typ des Rückgabewerts eingefügt werden. An dieser Stelle unterscheidet sich die Swift-Syntax stark von der Java-Syntax. Die eigentliche Rückgabe eines Wertes erfolgt dann jedoch wieder analog zu Javas Notation, durch das Schlüsselwort `return` und den Wert, der zurückgegeben werden soll. So soll die gezeigte Funktion einen Boolean zurückgeben und macht dies auch, indem sie den Wert „true“ zurückgibt.

```

func BeispielMethode(parameter1: String, parameter2: Int) -> Bool{
    beispielString = parameter1
    beispielInteger = parameter2

    return true
}

```

Abbildung 15: Eine Funktion mit zwei Parametern und einem Rückgabewert in Swift.

Vor einigen Funktionen kann zudem das Präfix *override* stehen. Dieses weißt auf Funktionen hin, die aus einem Protokoll übernommen und überschrieben wurden. Solche Funktion sind beispielsweise die im nächsten Abschnitt beschriebenen *viewDidLoad*- und *viewWillAppear*-Funktionen. Diese und weitere für die App wichtige Funktionen, die aus Protokollen übernommen wurden, werden im Folgenden vorgestellt.

### 3.5.1 ViewDidLoad() und viewWillAppear()

Sowohl bei *viewDidLoad()* als auch bei *viewWillAppear()* handelt es sich um Funktionen, die durch das *UIViewController*-Protokoll bereitgestellt werden. Beide Funktionen werden benötigt, um den Lebenszyklus eines App-Screens zu steuern. Durch ihren Funktionskörper wird bestimmt, wann welche Komponente (z.B. ein Label oder eine Tabelle) geladen beziehungsweise neugeladen wird. Die zwei Funktionen unterscheiden sich im Grunde genommen nur im Zeitpunkt, zu denen sie aufgerufen werden. Die *viewDidLoad*-Funktion wird einmalig zu Beginn aufgerufen, sobald die dazugehörige Swift-Klasse in den Hauptspeicher geladen wird. In dieser Funktion sollten deshalb alle Komponenten mit den Werten befüllt werden, die sie zum Start besitzen sollen. Beispielsweise könnte innerhalb dieser Methode einen Punktteststand auf den Wert „0“ gesetzt werden.

Die Funktion *viewWillAppear()* wird hingegen immer dann aufgerufen, wenn das entsprechende App-Fenster erneut angezeigt wird. Diese Funktion ist somit für das Aktualisieren aller Komponenten einer App verantwortlich. Der Punktteststand könnte durch diese Funktion also stets aktualisiert werden.

### 3.5.2 CloudKit-Funktionen

Um die Daten, die im Laufe der Zeit durch die App generiert werden, an einem zentralen Ort speichern zu können, wird das CloudKit-Framework von Apple eingesetzt. Nachdem eine allgemeine Einführung in das Framework bereits in Kapitel 2.4 gegeben wurde, sollen nun die Funktionen vorgestellt werden, durch die mit der Datenbank kommuniziert werden kann. Wie fast jede Datenbank, so setzt auch CloudKit auf die vier Standardbefehle *create*, *read*, *update* und *delete*. Für jeden Befehl stellt das Framework eine Funktion bereit. Bevor diese allerdings verwendet werden können, muss zuerst eine Verbindung zu Apples iCloud hergestellt werden. Hierfür muss das *CloudKit* zu Beginn importiert werden. Anschließend kann mit Hilfe des Befehls *let publicDB = CKContainer.default().publicCloudDatabase* eine Verbindung zu der Cloud aufgebaut werden. Der Befehl bewirkt, dass über die Konstante „*publicDB*“ mit der Datenbank kommuniziert werden kann. Anschließend können die vier folgenden Funktionen verwendet werden, um die Befehle an die Datenbank zu schicken.

#### fetchData()

Mit der *fetchData*-Funktion können Daten aus der Datenbank ausgelesen werden. Durch sie wird also der *read*-Befehl realisiert. In Abbildung 16 wird gezeigt, wie diese Funktion aussehen kann. Zu Beginn der Funktion wird eine *CKQuery* erstellt (oberer Kasten). Diese legt fest, welche Daten aus der Datenbank ausgelesen werden sollen. In diesem Fall soll die Funktion alle Daten des Datentyps

„user“ zurückgeben. Anschließend wird der Befehl an die Datenbank geschickt und ausgeführt (unterer Kasten). Hier können die erhaltenen Daten direkt verarbeitet und eine Fehlerbehandlung implementiert werden, also was passieren soll, wenn der Befehl nicht erfolgreich ausgeführt werden konnte. Hierfür wird die vom Framework bereitgestellte Error-Variable inklusive ihrer Eigenschaften ausgenutzt. Geht bei der Übertragung der Daten zu der Datenbank etwas schief, steht in dieser Variablen ein Fehlercode, der den aufgetretenen Fehler beschreibt, ansonsten enthält sie den Wert nil.

```
func fetchData() {
    let query = CKQuery(recordType: "user", predicate: NSPredicate(value: true))

    publicDB.perform(query, inZoneWith: nil) { (records, error) in
        if error != nil {
            // Führe eine Fehlerbehandlung durch.
        } else {
            // Verarbeite die erhaltenen Daten.
        }
    }
}
```

Abbildung 16: Die fetch-Funktion aus dem CloudKit-Framework.

### saveData()

Der create-Befehl wird durch die *saveData*-Funktion umgesetzt (Abbildung 17), welche immer aufgerufen wird, wenn ein neues Objekt in der Datenbank gespeichert werden soll. Um ein Objekt speichern zu können, muss dieses zuerst erstellt werden. Hierfür muss eine einzigartige ID und der Datentyp angegeben werden (erster Kasten). Anschließend kann jedem Attribut dieses Datentyps ein Wert zugeordnet werden (zweiter Kasten), das ist jedoch nicht zwingend erforderlich. Falls für ein Attribut kein Wert angegeben wird, wird der Eintrag in der Datenbank leer sein und kann bei Bedarf nachträglich befüllt werden. Die eigentliche Durchführung des create-Befehls findet im dritten Kasten statt. Auch hier hat der Entwickler wieder Dank der Error-Variablen die Möglichkeit, eine Fehlerbehandlung durchzuführen.

```
func saveData() {
    let QuestRecordID = CKRecord.ID(recordName: "Name der Aufgabe")
    let QuestsRecord = CKRecord(recordType: "Quests", recordID: QuestRecordID)

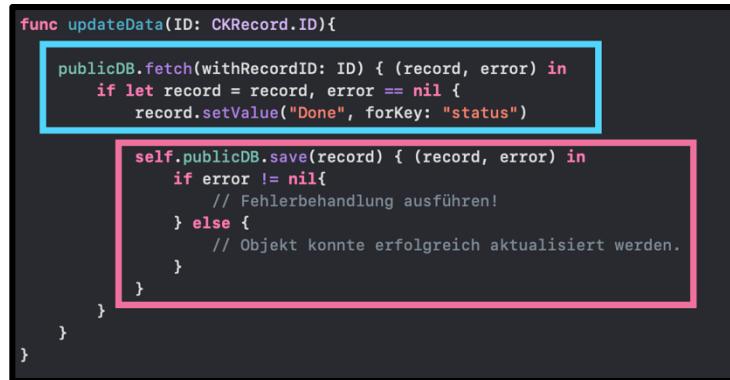
    QuestsRecord["description"] = "keine"
    QuestsRecord["name"] = "Klo putzen"
    QuestsRecord["points"] = "5"
    QuestsRecord["resident"] = "Liam"
    QuestsRecord["status"] = "erledigt"
    QuestsRecord["wg"] = "eine WG"

    publicDB.save(QuestsRecord) { (record, error) in
        if error != nil{
            // Führe eine Fehlerbehandlung durch.
        } else {
            // Objekt konnte erfolgreich angelegt werden.
        }
    }
}
```

Abbildung 17: Die save-Funktion aus dem CloudKit-Framework.

## updateData()

Um Einträge in der Datenbank aktualisieren zu können, wird die *updateData*-Funktion verwendet. Diese Funktion benötigt dafür lediglich die ID des entsprechenden Objektes. Anschließend können per *record.setValue(„neuer Wert“, forKey: „Attribut“)*-Befehl die Werte von beliebig vielen Attributen überschrieben werden. Danach muss das Objekt mit den neuen Werten gespeichert werden. Der dafür benötigte Code ist in Abbildung 18 zu sehen. Im oberen Kasten wird ein Datum aus der Datenbank geholt und eines seiner Attributwert verändert. Im unteren Kasten wird dieses anschließend an die Datenbank zurückgeschickt. Auch hier kann eine Fehlerbehandlung implementiert werden.



```
func updateData(ID: CKRecord.ID) {
    publicDB.fetch(withRecordID: ID) { (record, error) in
        if let record = record, error == nil {
            record.setValue("Done", forKey: "status")
            self.publicDB.save(record) { (record, error) in
                if error != nil {
                    // Fehlerbehandlung ausführen!
                } else {
                    // Objekt konnte erfolgreich aktualisiert werden.
                }
            }
        }
    }
}
```

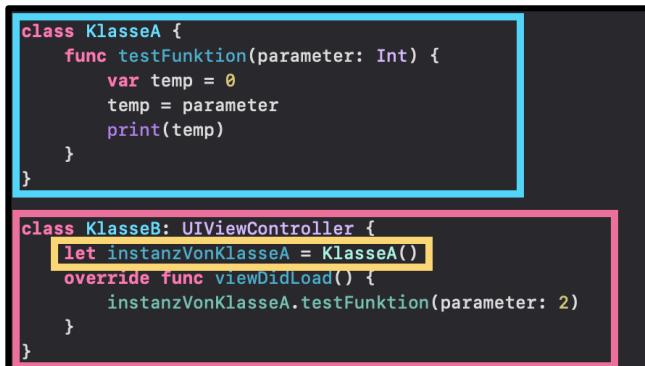
Abbildung 18: Die update-Funktion aus dem CloudKit.

## deleteData()

Das Löschen von Objekten einer Datenbank wird durch die *deleteData*-Funktion umgesetzt. Genauso wie zum Aktualisieren von Daten wird dafür die ID des Objektes benötigt, das gelöscht werden soll. Durch den Befehl *publicDB.delete(withRecordID: „ID des zu löschen Objektes“)* wird das Objekt daraufhin unwiderruflich gelöscht. Anschließend kann auch bei dieser Funktion der Programmierer festlegen, was geschehen soll, wenn ein Objekt nicht gelöscht werden konnte.

### 3.5.3 Funktionen aus anderen Klassen aufrufen

Es kann sinnvoll sein, Funktionen auszulagern, die in verschiedenen Klassen verwendet werden. Dies verhindert, dass die gleichen Methoden in mehreren Klassen erneut deklariert werden müssen, was einerseits kein guter Programmierstil ist und andererseits die Klassen nicht unnötig groß werden lässt, wodurch diese unübersichtlicher bleiben.



```
class KlasseA {
    func testFunktion(parameter: Int) {
        var temp = 0
        temp = parameter
        print(temp)
    }
}

class KlasseB: UIViewController {
    let instanzVonKlasseA = KlasseA()
    override func viewDidLoad() {
        instanzVonKlasseA.testFunktion(parameter: 2)
    }
}
```

Abbildung 19: Klasse B verwendet die Methode aus Klasse A.

Um eine Funktion auszulagern, muss diese erst einmal in einer neuen Klasse „A“ erstellt werden (Abbildung 19, oberer Kasten). Soll anschließend aus einer anderen Klasse „B“ (unterer Kasten) heraus nun die erstellte Methode verwendet werden, so muss innerhalb der Klasse B zuerst eine Instanz der Klasse A erstellt werden (mittlerer Kasten). Daraufhin kann auf die gewünschte Funktion via „*Instanzname.Funktionsname*“ zugegriffen werden.

Mit Hilfe dieses Kapitels sollte der Leser grundlegende Kenntnisse in Swift erlangt haben. Diese werden benötigt, um den anschließenden Kapiteln besser folgen zu können. Das Kapitel sollte dabei nicht als Kompendium der Sprache Swift dienen, sondern es wurden nur die Aspekte behandelt, die bei der Entwicklung der App eine Rolle spielten. Das nächste Kapitel befasst sich mit den Designentscheidungen, die während der Entwicklung der App getroffen wurden. So musste einerseits ein sinnvolles Design für die App gefunden werden, andererseits mussten Backend-Entscheidungen getroffen und der Ausprägungsgrad der Gamification bestimmt werden.

## 4. Design

Im Entwicklungsprozess der hier implementierten App mussten viele Designentscheidungen getroffen und Aspekte berücksichtigt werden. In diesem Kapitel soll diese Phase der Entwicklung, in der alle wesentlichen Designaspekte für die App diskutiert wurden, dem Leser nähergebracht werden. So werden die Gedanken und Überlegungen, die sich der Entwickler zu den einzelnen Punkten gemacht hat, im Folgendem beschrieben. Dadurch soll der Leser nachvollziehen können, weshalb die App letztendlich so programmiert wurde, wie es in Kapitel 5 beschrieben wird. Alle Entscheidungen und Überlegungen des Entwicklers lassen sich dabei in die Rubriken Frontend der App, verwendete Gamification-Elemente und eingesetzte Speicherverwaltung einteilen. Diese drei Bereiche stellen gleichzeitig die Unterkapitel dieses Abschnitts dar. Zu Beginn wird auf die Anforderungen bezüglich des optischen Designs und die Navigation durch die App eingegangen. Anschließend werden die Bedingungen beschrieben, die die Gamification-Mechaniken erfüllen sollten. Am Ende dieses Kapitels werden die Eigenschaften genannt, die die verwendete Speicherverwaltung mitbringen sollte.

### 4.1 Anforderungen an das Frontend der App

*Keep it simple, stupid!* – das sogenannte KISS-Prinzip [39] beschreibt den Umstand, dass ein System möglichst schlicht und einfach designt werden sollte, da es auf diese Weise am besten von den Benutzern bedient werden kann. Nach diesem Leitsatz soll die hier implementierte App entwickelt werden. Dadurch soll sichergestellt werden, dass sich der Benutzer nicht durch unnötig lange Dialoge klicken muss oder von zu vielen Ereignissen und Knöpfen auf einmal erschlagen und dadurch überfordert wird. Der Anspruch des Entwicklers ist es, einerseits das Erscheinungsbild der App recht schlicht und aufgeräumt zu halten, gleichzeitig sollte die App jedoch nicht langweilig wirken. Zudem sollte der Benutzer alle wichtigen Informationen kompakt einsehen können. Um diesen Spagat zwischen Verspieltheit und Schlichtheit zu bewerkstelligen, wurden eine Vielzahl an Anforderungen bezüglich des Frontends der App formuliert, auf die in diesem Abschnitt genauer eingegangen werden sollen. Viele dieser Anforderungen basieren dabei auf den Richtlinien, welche Ben Shneiderman und Catherine Plaisant in ihrem Buch „Designing the User Interface: Strategies for Effective Human-Computer Interaction“ zu dem Thema User Interface Design aufgestellt haben [35].

#### 4.1.1 Grundsätzliche Anforderungen an das User Interface

Die wichtigste Anforderung an das User Interface der hier vorgestellten App war, dass es einen strukturierten und schlichten Eindruck bei den Benutzern hinterlässt. Deshalb sollten glänzende oder gar blinkende Elemente vermieden werden, da diese die Benutzer von den wesentlichen Dingen ablenken könnten. Außerdem erscheinen die einzelnen Screens der App sonst schnell überladen und unübersichtlich. Des Weiteren sollten bunte Texte nur vereinzelt eingesetzt werden, da das User Interface sonst nach Meinung des Entwicklers dazu neigt, unaufgeräumt auszusehen. Auch auf einen auffälligen Hintergrund wird am besten verzichtet, damit die Benutzer durch diesen nicht abgelenkt werden. Gleichzeitig sollte das User Interface jedoch auch nicht langweilig wirken, also nicht ausschließlich in den Farben Schwarz und Weiß gehalten werden. Ein lieblos designtes Interface wirkt sich in der Regel negativ auf die User Experience der App aus, was zur Folge haben wird, dass die Benutzer nur ungerne mit dieser arbeiten werden. Als User Experience werden die Eindrücke und die Emotionen des Benutzers bezeichnet, die durch die Nutzung eines Systems hervorgerufen werden [17]. Je positiver diese ausfallen, desto mehr Spaß hat der Benutzer, wenn er mit dem System arbeitet. Aus diesem Grund ist ein ansprechendes User Interface auch für die hier vorgestellte App von großer Bedeutung.

Um die App also optisch ansprechender zu gestalten, wäre es von Vorteil, wenn ein paar farbige Elemente gezielt platziert werden. Auf sehr auffällige Farben, wie etwa Neonfarben, sollte dabei allerdings verzichtet werden. Stattdessen sollten die verwendeten Farben untereinander gut harmonieren, weshalb sie nicht aus gänzlich unterschiedlichen Bereichen des RGB-Farbraumes stammen sollten, da das User Interface sonst dazu neigt, chaotisch zu wirken. Zudem ist es wichtig, dass die Farben einem gemeinsamen Schema folgen. So sollte vermieden werden, dass auf einigen Screens die Texte bunt gefärbt und die Knöpfe schwarz-weiß gehalten werden und auf anderen wiederum die Texte schwarz und die Knöpfe bunt sind. Darüber hinaus erfüllen die Farben im Idealfall noch eine weitere Funktion. Beispielsweise könnte jeder Bereich der App eine eigene Farbe erhalten, um diese noch besser voneinander abzugrenzen.

Neben einem ansprechenden Erscheinungsbild des User Interfaces, spielt die Konsistenz in diesem Zusammenhang eine wichtige Rolle. Die gleiche Funktion sollte also nicht auf verschiedene Arten durchführbar sein. Auch die verwendete Terminologie sollte konsistent und für die Benutzer verständlich sein. Deshalb sollten Begriffe wie Aufgaben, Todos und Quests nicht synonym verwendet werden, da die Benutzer bei fehlender Konsistenz schnell verwirrt und frustriert werden könnten.

#### **4.1.2 Bilder und Icons**

Da die App an sich eher schlicht gehalten werden soll, sollten die verwendeten Bilder und Icons den spielerischen Charakter der App durch ihren Look unterstreichen und diese zudem optisch etwas aufpeppen. Aus diesem Grund würde es sich anbieten, dass die Bilder und Icons einen comicartigen Stil besitzen und im Gegensatz zu der restlichen App sehr bunt ausfallen. Weiterhin sollte die Auswahl an Avatar-Bildern und Todo-Icons ausreichend groß sein. Sonst besteht die Gefahr, dass das User Interface eintönig wirkt, da auf den Screens häufig die gleichen Symbole zu sehen sind. Damit die App keinen chaotischen Eindruck auf die Benutzer macht, sollte zudem darauf geachtet werden, dass die Bilder und Icons ein einheitliches Setting besitzen, also nicht beispielsweise einerseits Fantasiewesen und andererseits Dinosaurier zeigen.

#### **4.1.3 Navigieren durch die App**

Neben dem User Interface wirkt sich die Art und Weise, wie durch eine App navigiert wird, ebenfalls stark auf die User Experience aus. Benötigen die Benutzer beispielsweise viele Klicks, um an eine gewünschte Stelle innerhalb der App zu gelangen, oder wissen gar nicht, wie sie dort hinkommen können, so kann dies die Benutzer frustrieren. Deshalb sollten die Benutzer zu jedem Zeitpunkt einen groben Überblick über den kompletten Inhalt der App haben und mit möglichst wenigen Aktionen zu ihren Zielpunkten kommen. Des Weiteren sollte die Navigation übersichtlich und somit auf einem Blick einsehbar sein. Die Benutzer sollten also nicht scrollen müssen, um alle Punkte der Navigation sehen zu können, da sie sonst rasch den Überblick verlieren. Neben aussagekräftigen Bezeichnungen für die verschiedenen Bereiche, können Icons an dieser Stelle unterstützend eingesetzt werden. Jeder Punkt der Navigation sollte zudem einen strikten Kontextwechsel mit sich bringen. Ist das nicht der Fall, muss darüber nachgedacht werden, ob es nicht sinnvoller wäre, die entsprechenden Bereiche unter einem Navigationspunkt zusammenzulegen.

Die Benutzer sollten darüber hinaus keine neue Navigation erlernen müssen. Vielmehr sollte diese intuitiv oder aus anderen Apps bereits bekannt sein. Dropdown-Menüs, die in vielen Computerprogrammen zur Navigation erfolgreich eingesetzt werden, sollten allerdings vermieden werden. Einerseits, können Benutzer durch diese Navigationsweise nicht immer alle Bereiche auf einem Blick einsehen, und andererseits werden tendenziell mehr Klicks benötigt als bei anderen Navigationsmethoden. Die Dropdown-Menüs eignen sich für Computerprogramme in erster Linie so gut, da die

Bildschirme hier um ein Vielfaches größer sind und mit dem Mauszeiger präziser geklickt werden kann als mit dem Finger.

#### **4.1.4 Anlegen von Ereignissen**

Das Anlegen von Accounts, Todos und Gruppen innerhalb der App sollte so entworfen werden, dass es die Benutzer vor keine großen Herausforderungen stellt. Das setzt allerdings voraus, dass diese Vorgänge einerseits nach einem bereits bekannten Schema ablaufen und andererseits in sich konsistent sind. Accounts sollten also nicht auf eine komplett andere Weise erstellt werden als die Todos oder Gruppen. Damit sich das Anlegen der Ereignisse für die Benutzer vertraut anfühlt, sollte recherchiert werden, wie dieser Vorgang in anderen bekannten Apps umgesetzt wurde.

Des Weiteren ist es wichtig, die Benutzer vor Fehleingaben zu schützen. Müssen diese beispielsweise einige Angaben bei der Erstellung machen, sollten die Benutzer wenn möglich durch aussagekräftige Beispiele unterstützt werden. So könnte in den Eingabefeldern Beispielwerte stehen, anhand denen die Benutzer sich jeweils orientieren können. Fehler werden sich dennoch nicht komplett vermeiden lassen. Für diese Fälle ist es wichtig, den Benutzern aussagekräftige und für sie verständliche Fehlermeldungen bereitzustellen. Fehlermeldungen wie etwa „Es ist ein Fehler aufgetreten“ oder „Der Fehler 0xF3A1 ist aufgetreten“ sollten deshalb unbedingt vermieden werden. Stattdessen sollte ein Vokabular gewählt werden, welches den Benutzern geläufig ist. Weiterhin sollte ihnen erklärt werden, was sie falsch gemacht haben und wie sie ihren Fehler selbst beheben können. Um die Frustration bei den Benutzern möglichst gering zu halten, sollten die Fehlermeldungen zudem unmittelbar nachdem ein Fehler aufgetreten ist angezeigt werden.

Darüber hinaus sollten die Benutzer sich den Erstellungsvorgang nicht bewusst merken müssen. Vielmehr sollten sie durch Hinweise in Form von Bildern oder kurzen Texten daran erinnert werden, wie sie diesen Prozess erfolgreich absolvieren können. Da das Speichern der Ereignisse in die Datenbank im Hintergrund abläuft und somit nicht für die Benutzer sichtbar ist, ist es wichtig, die Benutzer über den aktuellen Stand des Systems zu informieren. Andererseits besteht die Gefahr, dass der Benutzer mit einem Ereignis interagieren möchte, obwohl dieses zu diesem Zeitpunkt noch gar nicht angelegt wurde. Diese Situation kann auftreten, da die Daten einige Augenblicke benötigen, um an die Datenbank geschickt zu werden. Dadurch könnten die Benutzer verwirrt und die App für sie unberechenbar werden. Aus diesem Grund sollten die Benutzer zumindest immer dann über den aktuellen Systemzustand informiert werden, wenn im Hintergrund für sie relevante Prozesse laufen. Hierfür könnten beispielsweise Popup-Fenster eingesetzt werden, in denen durch eine kurze Animation dargestellt wird, was das System zu diesem Moment gerade macht.

### **4.2 Gamification-Mechaniken**

Damit die App möglichst viele Benutzer motiviert, sollte darauf geachtet werden, dass alle in Kapitel 2.1.3 beschriebenen Spielertypen berücksichtigt werden. Für jede eingesetzte Gamification-Mechanik muss im Vorfeld also sorgfältig überlegt werden, welche Spielergruppe durch diese angesprochen wird. Zudem sollte vermieden werden, dass ein bestimmter Spielertyp deutlich häufiger durch Mechaniken angesprochen wird, als die übrigen. Das könnte sonst zu Frust bei den weniger beachteten Spielertypen führen.

Auch spielt die Anzahl der eingesetzten Gamification-Mechaniken innerhalb der App eine entscheidende Rolle. Sind es zu wenige, kann die Motivation der Benutzer darunter leiden, da es sich für sie nicht ausreichend stark nach einem Spiel anfühlt, sondern eher wie eine gewöhnliche Todo-Listen App. Das Verwenden von zu vielen Gamification-Mechaniken kann sich jedoch ebenfalls negativ auf das Spielerlebnis der App auswirken. Gerade Benutzer mit weniger Videospielerfahrung könnten sich in diesem Fall überfordert und abgehängt fühlen und deshalb die App meiden. Der Ent-

wickler legt allerdings großen Wert darauf, dass die App von der breiten Masse verwendet werden kann.

Neben der Anzahl sollte auch darauf geachtet werden, welche Mechaniken eingesetzt werden. Mechaniken, durch die nur ein sehr geringer Teil der Spielergruppe angesprochen wird, sollten nicht verwendet werden. So gibt es beispielsweise in vielen Videospielen einen sogenannten Hardcore-Modus. In diesem spielen die Spieler häufig die gleichen Inhalte nur unter erschwerten Bedingungen, etwa dass Gegner mehr Schaden verursachen oder verschiedene Level unter Zeitdruck absolviert werden müssen. Einigen Spielern bereitet diese Mechanik mit hoher Wahrscheinlichkeit viel Spaß, jedoch kommen die meisten Spieler in der Regel mit derartigen Modi nie in Berührung. Aus diesem Grund sollten eher Mechaniken verwendet werden, durch die sich der Großteil der Spieler motivieren lässt, wie Ranglisten oder Punktesysteme.

Darüber hinaus sollten die benutzten Gamification-Mechaniken aufeinander aufbauen oder Synergien bilden, damit das Spielerlebnis für die Benutzer runder wirkt. Beispielsweise ist es sinnvoll, ein Punktesystem in Kombination mit einer Rangliste einzusetzen. So wissen die Benutzer direkt, weshalb es sich lohnt, Punkte zu erarbeiten.

### **4.3 Speicherverwaltung**

Im Laufe der Zeit werden durch die App viele Daten generiert, die verwaltet und gespeichert werden müssen. Da innerhalb der App mehrere Benutzer miteinander interagieren werden und deshalb jeder Nutzer nicht nur auf seine eigenen Daten zugreifen muss, wird eine zentrale Datenbank benötigt. Auch wenn die gewählte Speicherverwaltung im Idealfall sehr kostengünstig oder gar kostenlos sein sollte, ist es wichtig, dass die Daten dennoch in akzeptabler Geschwindigkeit übertragen werden. Die Benutzer sollten also nicht minutenlang auf angefragte Daten warten müssen, da sich dies negativ auf den Workflow auswirken würde.

In der heutigen Zeit ist ein verantwortungsvoller Umgang mit den Daten im Netz wichtiger denn je. Das bedeutet in erster Linie, dass diese sicher verwahrt werden müssen, um so vor Hacker-Angriffen geschützt zu werden. Da auch in der hier vorgestellten App zum Teil sensible Daten generiert werden, spielt der Schutz dieser Daten eine wichtige Rolle. So sollte zumindest eine Hashfunktion auf die Passwörter angewendet werden, bevor sie auf der Datenbank abgelegt werden. Durch eine Hashfunktion werden aus normalen Texten zufällige Zeichenfolgen, die keine Schlüsse auf den ursprünglichen Text zulassen. Dadurch wird es Hackern zusätzlich erschwert, an sicherheitskritische Daten zu gelangen. Des Weiteren wäre es von Vorteil, wenn der komplette Sicherheitsaspekt von dem Datenbankanbieter übernommen wird. Eine eigene Sicherheitsstruktur zu implementieren, würde den zeitlichen Rahmen dieser Masterthesis sprengen.

Da zu dem Zeitpunkt der Entwicklung nicht absehbar ist, wie viele Benutzer die App einmal haben wird, sollte die gewählte Speicherverwaltung gut skalierbar sein. Engpässe des Datentransfers oder der Datenbank, aufgrund höherer Benutzerzahlen, müssen unbedingt vermieden werden. Im schlimmsten Fall müsste die eingesetzte Speicherverwaltung durch eine andere ersetzt werden. Das wiederum hätte zur Folge, dass vermutlich große Teile des Codes nicht mehr verwendbar wären und die komplette Architektur der App ersetzt werden müsste. Um dieses Szenario auszuschließen, sollte also von Anfang an auf eine gut skalierbare Lösung gesetzt werden.

## 5. Implementierung

In diesem Kapitel wird die Implementierungsphase der App-Entwicklung vorgestellt. So wird zu Beginn beschrieben, wie die geforderten Punkte aus dem vorangegangenen Kapitel umgesetzt wurden. Anschließend wird auf die Herausforderungen und deren Lösungen eingegangen, die beim Programmieren aufgetreten sind. Am Ende des Kapitels erhält der Leser anhand einiger aussagekräftiger Beispiele ein Einblick in die iOS App-Entwicklung. Durch diesen Einblick bekommt dieser hoffentlich ein besseres Verständnis von der App-Entwicklung mit Hilfe der Programmiersprache Swift und der Entwicklungsumgebung Xcode.

### 5.1 Implementierung der App unter Berücksichtigung der Anforderungen aus Kapitel 4

Nachdem in Kapitel 4 die Anforderungen an das User Interface, die Gamification-Mechaniken, die Speicherverwaltung und das generelle Arbeiten mit der App an sich formuliert wurden, befasst sich dieser Abschnitt damit, wie diese Punkte letztendlich implementiert wurden. Da es sich bei dem Programmieren von Software immer um einen iterativen Prozess handelt, ganz gleich wie viel Planungsaufwand im Vorhinein betrieben wird, wird in diesem Abschnitt nicht nur auf die finale Umsetzung eingegangen. Stattdessen werden auch die vorangegangenen Iterationen beschrieben, die zwangsläufig Einfluss auf die finale Version der App genommen haben.

#### 5.1.1 Implementierung des User Interfaces

Schon relativ früh in der Entwicklung stand fest, dass die App ein eher schlichtes Design erhalten soll. Deshalb sollte die App entweder ein recht helles Design mit dunkler Schrift erhalten oder ein dunkles Design mit heller Schrift. Auch stand gleich zu Beginn fest, dass der Hintergrund nicht eingefärbt wird, um nicht von den darauf liegenden Inhalten abzulenken. Stattdessen wurden nur vereinzelt Farben eingesetzt, damit die App nicht zu trostlos wirkt. Nachdem sowohl ein heller als auch ein dunkler Prototyp der App erstellt wurde, legte sich der Entwickler auf das helle Design fest, da dieses in seinen Augen besser mit den eingesetzten Farben harmonierte.

Bei einer recht frühen Version der App wurden deshalb nur die verwendeten Bilder und Icons eingefärbt und die restlichen Inhalte komplett schwarz beziehungsweise weiß gehalten. In Abbildung 20 im linken Kasten wird beispielsweise ein Screen aus dieser Version gezeigt. Diese Farbzusammensetzung war jedoch noch nicht zufriedenstellend, da die App zu diesem Zeitpunkt noch sehr lieblos wirkte. Aus diesem Grund wurden in die nächste Version zusätzliche Farbtupfer eingebaut, indem einige Buttons eingefärbt wurden, wie anhand der Szenen in Abbildung 20 im rechten Kasten zu sehen ist. Diese Variante brachte allerdings auch Probleme mit sich, da keine Farbe gefunden wurde, die sich für dieses Design anbot. Entweder waren die Farben zu hell und wurden deshalb auf dem weißen Hintergrund kaum wahrgenommen oder der Benutzer könnte mit den Farben bestimmte Dinge assoziieren, beispielsweise mit Rot (schlecht, böse, dringend) oder mit Grün (gut, einfach).

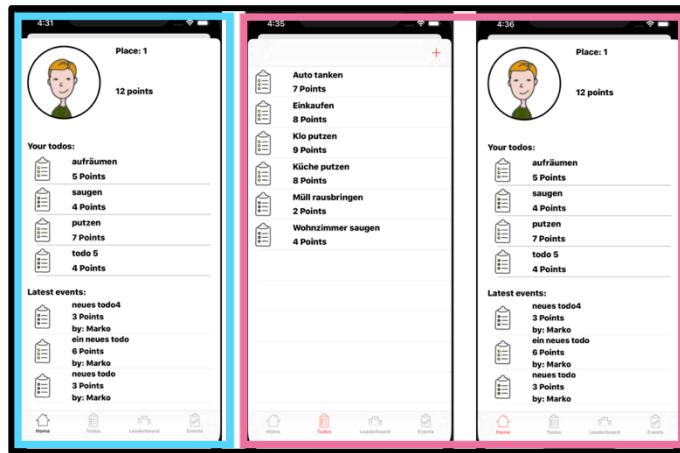


Abbildung 20: Links: Ein erster Entwurf des Home-Screens der App. Rechts: Weitere Entwürfe der App mit mehr farbigen Elementen (Home-Screen und Todo-Screen).

Deshalb wurde auch diese Variante verworfen und so kam es letztendlich nach einigen weiteren Versuchen zu dem Design, welches in der fertigen App verwendet wird. Wie in Abbildung 21 ange deutet wird, befindet sich im oberen Bereich jedes App-Screens ein individuell eingefärbtes Banner, in dem der Name des aktuellen Screens in weißer Schrift steht. Durch diese Designlösung weiß der Benutzer einerseits immer direkt, in welchem Bereich der App er sich befindet und andererseits erhalten die Screens ein zusätzliches farbiges Element, wodurch sie weniger langweilig und eintönig erscheinen. Darüber hinaus werden die unterschiedlichen Bereiche auf diese Weise auch farblich voneinander getrennt, wodurch dem Benutzer verdeutlicht werden soll, dass diese sich mit verschiedenen Themen befassen. Da die Banner jedoch sehr prominent auf den Screens zu sehen sind, musste darauf geachtet werden, dass die eingesetzten Farben gut untereinander harmonieren. Hierfür wurde mit Hilfe von Farbpaletten eine Kombination ausgearbeitet, die der Entwickler als sehr ansprechend empfindet. Um die Konsistenz auch in diesem Zusammenhang zu wahren, wurden die Farben in der kompletten App nach dem gleichen Schema eingesetzt. Auf jedem Screen ist jeweils nur das Banner und die Tab Bar in der selben Farbe eingefärbt. Alle anderen Elemente, abgesehen von den Bildern und Icons, wurden in den Farben Schwarz und Weiß gehalten. Durch diese Maßnahmen soll der Vorgabe eines schlichten aber dennoch nicht langweiligen Designs Rechnung getragen werden.

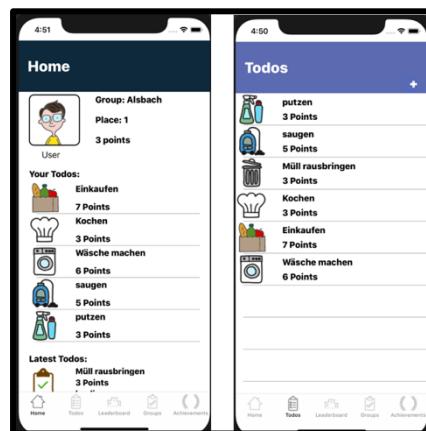


Abbildung 21: Zwei Szenen der finalen Version der App.

### 5.1.2 Verwendete Bilder und Icons

Alle in der App verwendeten Bildern und Icons wurden von der Schwester des Entwicklers selbstständig entworfen. Sie hat lediglich die Vorgabe erhalten, die Bilder und Icons in einem comicartigen Stil und sehr farbenfroh zu gestalten. Darüber hinaus sollten alle Motive einem Setting entstammen, um die App nicht absurd wirken zu lassen. Da die App ansonsten eher schlicht gehalten ist, soll durch die Bilder der spielerische Aspekt der App hervorgehoben und der Look insgesamt interessanter werden. Weiterhin mussten in diesem Zusammenhang zwei grundsätzliche Entscheidungen bezüglich der eingesetzten Bilder und Icons getroffen werden: zum einen, welche Metrik die Icons für die Todos verfolgen sollen, und zum anderen, ob die Benutzer ihre Profilbilder frei wählen dürfen oder sie aus einer Auswahl von Bildern eines auswählen müssen.

In einer früheren Version der App sollten durch die für die Todos verwendeten Icons angedeutet werden, wie anspruchsvoll das jeweilige Todo ist. Dafür wurde ein Todo-Icon in den Farben Grün, Gelb und Rot erstellt, wobei das grüne Icon bei weniger anspruchsvollen und das rote Icon bei anspruchsvollen Aufgaben verwendet werden sollte. Das Problem bei dieser Variante war jedoch, dass die Farben missinterpretiert werden könnten. So könnten die Benutzer bei einem roten Todo davon ausgehen, dass es demnächst verfällt und deshalb dringend bearbeitet werden muss. Des Weiteren wirkten die App-Screens durch diese Variante langweilig, da überall nur das gleiche Icon zu sehen war, wie es zum Beispiel in Abbildung 22 der Fall ist.



Abbildung 22: Die Todo-Szene aus einer recht frühen Iteration der App.

Aus diesen Gründen wurde diese Variante verworfen und dem Benutzer stattdessen eine Auswahl an Icons zur Verfügung gestellt, aus denen er eines für sein Todo frei wählen kann. Durch die Icons ist es ihm einerseits möglich, die Tätigkeiten seiner erstellten Todos visuell hervorzuheben und andererseits wirken die Screens durch die verschiedenen Icons lebendiger und abwechslungsreicher (siehe Abbildung 21). Durch die Punktzahl, die neben jedem Todo steht, können die Benutzer trotzdem den Schwierigkeitsgrad des jeweiligen Todos ablesen.

Was die Profilbilder angeht, war bereits früh im Entwicklungsprozess klar, dass die Benutzer diese nicht komplett frei wählen dürfen, da sonst die Gefahr besteht, dass der comicartige Look der App gestört werden könnte, beispielsweise wenn ein Bild von einem Auto als Profilbild verwendet werden würde. Stattdessen werden den Benutzern einige Avatare bereitgestellt, aus denen sie sich einen für ihr Profilbild auswählen können.

Durch regelmäßiges Einbinden neuer Bilder soll zudem sichergestellt werden, dass den Benutzern eine ausreichend große Auswahl zur Verfügung steht. Dadurch sollen sich ständig wiederholende Bilder auf den Screens vermieden werden.

### 5.1.3 Eingesetzte Navigation durch die App

Um durch die App navigieren zu können, wurde sich für eine sogenannte Tab Bar [4] entschieden. Bei einer Tab Bar handelt es sich um eine Leiste am oberen oder unteren Rand des Displays, auf der Buttons fixiert sind. Mit Hilfe dieser Buttons kann der Benutzer durch die einzelnen Tabs der App navigieren. Die Instagram-App und Facebook-App sind zwei sehr prominente Beispiele von vielen Apps, die auf eine Tab Bar zur Navigation setzen. Auch wenn Tab Bars ein sehr beliebtes Mittel sind, um durch Apps zu navigieren, sollten dennoch einige Punkte bei ihrem Einsatz berücksichtigt werden. Zu diesem Thema hat Apple selbst deshalb einige Guidelines aufgestellt [4], die im Folgenden sinngemäß wiedergegeben werden.

**Verwende Tab Bars ausschließlich zur Navigation.** Sollen durch die Buttons auf der Tab Bar abgesehen von einem Tab-Wechsel eine andere Aktion ausgeführt werden, dann eignet sich die Tab Bar für diese Situation nicht.

**Vermeide zu viele Tabs.** Es wird empfohlen, zwischen drei und fünf Tabs zu verwenden. Befinden sich zu wenige Tabs auf der Tab Bar, wirkt das Interface schnell leer und der Benutzer könnte das Gefühl bekommen, dass der Umfang der App sehr gering ist. Werden hingegen zu viele Tabs auf der Tab Bar platziert, wird die App zu komplex und der Benutzer weiß nicht genau, an welcher Stelle er die gewünschten Informationen erhält. Zudem werden die Buttons auf der Tab Bar bei zu vielen Tabs immer kleiner, wodurch sich der Benutzer häufiger verklicken wird. Das kann zur Folge haben, dass dieser mehr und mehr frustriert wird und deshalb nur noch ungerne mit der App arbeitet.

**Verstecke die Tab Bar zu keinem Zeitpunkt.** Die Tab Bar sollte für den Benutzer immer sichtbar sein, ganz egal in welchem Bereich der App dieser sich befindet. Ist das nicht der Fall, wird die App für den Benutzer unvorhersehbar und kann ihn verwirren.

**Jeder Tab sollte einen eigenen Kontext behandeln.** Jedes Mal, wenn der Tab gewechselt wird, sollte ein strikter Kontextwechsel erfolgen. Zwei Tabs sollten also nicht an unterschiedliche Stellen des selben Screens navigieren. Außerdem sollten sich zwei Tabs inhaltlich nicht mit dem gleichen Thema befassen, zum Beispiel, dass im Tab „A“ eine Account-Erstellung beginnt und diese im Tab „B“ fortgesetzt wird.

Da das Konzept der hier entwickelten App gegen keine der gerade genannten Richtlinien verstößt, wurde sich für die Tab Bar als Navigationstool entschieden, wie beispielsweise in Abbildung 21 zu sehen ist. Darüber hinaus stellt diese eine übersichtliche und einfach zu verwendende Navigationslösung dar, die für die Benutzer zu jedem Zeitpunkt sichtbar ist. Mit diesen Eigenschaften erfüllt die Tab Bar alle in Kapitel 4.1.3 formulierten Anforderungen an die Navigation und eignet sich somit hervorragend für die App.

Alternativ hätte die Navigation entweder mittels Dropdown-Menüs erfolgen können oder durch vor- und zurück-Buttons in jedem App-Screen. Bei beiden Varianten müsste der Benutzer jedoch deutlich mehr klicken, um an die gewünschte Stelle innerhalb der App zu kommen. Außerdem könnte der Benutzer durch diese Lösungen schnell den Überblick über die App verlieren, da diese für ihn nicht permanent sichtbar sind. Diese Lösungen bringen also nicht die geforderten Eigenschaften mit sich und stellen durch ihr umständliches Verhalten zudem einen Widerspruch zu dem in Kapitel 4.1 erwähnten KISS-Prinzip dar. Aus diesem Grund wurden diese Varianten nicht in Betracht gezogen.

### 5.1.4 Anlegen von Ereignissen

Innerhalb der App können neben neuen Accounts auch Todos und Gruppen angelegt werden. Alle Ereignisse werden dabei nach dem gleichen Schema erstellt. Dabei wurde darauf geachtet, dass Icons und kurze Texte unterstützend eingesetzt werden, damit der Benutzer sich den Erstellungs vorgang nicht bewusst merken muss. Durch ein Plus-Symbol in der oberen rechten Ecke des entsprechenden Screens symbolisiert, gelangt der Benutzer auf das Erstellungsformular für das jeweilige Ereignis (siehe Abbildung 23). Dort angekommen, muss dieser lediglich noch ein paar Angaben zu dem Ereignis machen, zum Beispiel welchen Namen dieses erhalten soll. Auch an dieser Stelle erhält der Benutzer Unterstützung, in Form von Beispieleingaben in den Textfeldern. Durch diese Eingaben soll grob angedeutet werden, was in das jeweilige Textfeld eingetragen werden soll. Mit einem weiteren Klick auf den create-Button wird das Ereignis anschließend erstellt. Da sich Fehler dennoch nicht gänzlich vermeiden lassen, erhält der Benutzer an dieser Stelle eine Fehlermeldung, falls bei dem Erstellen etwas schiefgelaufen ist. In dieser wird ihm genau beschrieben, weshalb ein Fehler aufgetreten ist und wie dieser behoben werden kann. Im Idealfall benötigt der Benutzer insgesamt also nur zwei Klicks, um ein neues Ereignis anzulegen. Auch dieses Vorgehen entspricht der KISS-Philosophie und wurde deshalb ganz bewusst so implementiert. Zudem ist dieser Vorgang den meisten iOS-Benutzern bereits vertraut. In vielen Apps, die auf den iOS-Geräten bereits vorinstalliert sind, werden nach genau diesem Schema neue Ereignisse erstellt. So setzen beispielsweise die Apple-Apps Kontakte, Uhren und Kalender auf dieses Vorgehen.

Da in diesem Zusammenhang im Hintergrund Daten zwischen der App und der Datenbank gesendet werden und das einige Augenblicke in Anspruch nimmt, wird der Benutzer an dieser Stelle über den aktuellen Systemstatus informiert. Wie auf der rechten Seite in Abbildung 23 zu sehen ist, wird mit Hilfe eines Popup-Fensters dargestellt, woran das System in diesem Moment arbeitet. Durch die gerade genannten Punkte soll sich das Anlegen von neuen Ereignissen für den Benutzer intuitiv und reibungslos anfühlen. Dadurch soll dieser weniger verwirrt und überfordert werden und sich das Arbeiten mit der App angenehmer anfühlen.

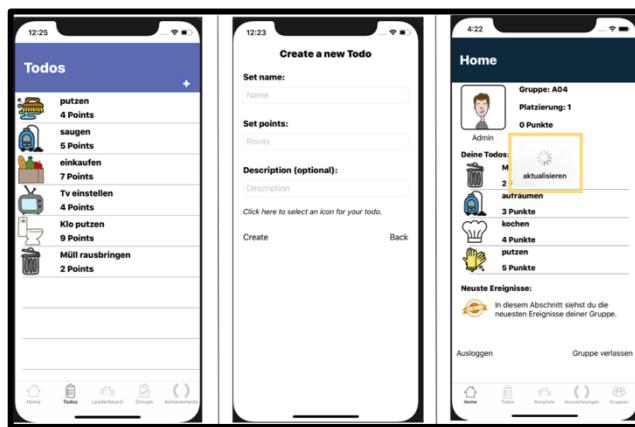


Abbildung 23: Links: Der Todo-Screen mit dem Plus-Symbol, das den Benutzer zum Erstellungsformular weiterleitet.

Mitte: Das Erstellungsformular, um ein Todo zu erstellen. Rechts: Der Home-Screen mit einer Ladeanimation.

### 5.1.5 Präsentation der Auszeichnungen

In diesem Abschnitt soll es noch nicht darum gehen, wie Auszeichnungen, die auch als Achievements bezeichnet werden, in dem Zusammenhang mit Gamification eingesetzt werden können. Auf dieses Thema wird in Kapitel 5.1.6 eingegangen. Vielmehr wird in diesem Abschnitt beschrieben, wie die Auszeichnungen dem Benutzer am sinnvollsten präsentiert werden.

Zu Beginn stand die Frage im Raum, ob alle in der App erreichbaren Auszeichnungen von Anfang an für den Benutzer sichtbar sein sollen oder nur die, die dieser bereits erreicht hat. Nach der Auffassung des Entwicklers ist es wichtig, dass der Benutzer direkt am Anfang einsehen kann, welche Herausforderungen auf ihn warten, da er sonst nur schwer nachvollziehen kann, weshalb er eine Auszeichnung erhalten hat. Außerdem kann er nicht wissen, ob Auszeichnungen Folgeauszeichnung haben, zum Beispiel folgt auf die Auszeichnung „Schließe 1 Todo ab“ die Auszeichnung „Schließe 5 Todos ab“. Der Benutzer könnte in diesem Fall von den Auszeichnungen eher verwirrt und frustriert werden, da er möglicherweise auf eine Auszeichnung hinarbeitet, die gar nicht existiert. Aus diesem Grund wurde die Variante umgesetzt, bei der alle Auszeichnungen gleich zu Beginn sichtbar sind. Durch diese Variante ergibt sich allerdings ein neuer Aspekt, der berücksichtigt werden muss: wie soll für den Benutzer kenntlich gemacht werden, welche Auszeichnungen er bereits erreicht hat und welche noch ausstehen?

Als Lösung für dieses Problem wurden die Auszeichnungen, die noch nicht erreicht wurden, in einem hellen Grau dargestellt, wohingegen Auszeichnungen, die bereits erreicht wurden, farbig angezeigt werden (siehe Abbildung 24). Durch diese Lösung soll der Benutzer direkt sehen können, wie weit sein Fortschritt bezüglich der Auszeichnungen ist. Zudem sieht er durch diese Lösung direkt, was für die einzelnen Auszeichnungen erwartet wird und muss sich nicht umständlich durch Dialoge klicken, um an die gewünschten Informationen zu kommen. Auch an dieser Stelle wurde versucht, das Arbeiten mit der App für den Benutzer so unkompliziert zu halten wie möglich.

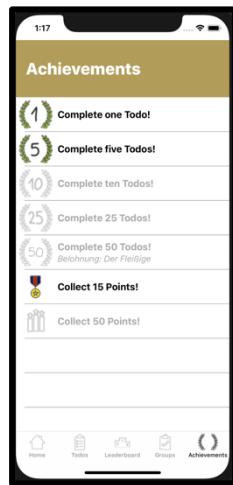


Abbildung 24: Die Ansicht des Auszeichnung-Tabs der App.

### 5.1.6 Benutzte Gamification-Mechaniken

Nachdem in den letzten Abschnitten primär der Entwicklungsprozess des Frontends beschrieben wurde, sollen in diesem Abschnitt die verwendeten Gamification-Mechaniken vorgestellt werden, durch die die Benutzer für das Bearbeiten der Todos motiviert werden sollen. Hierfür wird auf die einzelnen Mechaniken selbst und deren Implementierung eingegangen, die innerhalb der App eingesetzt werden. Zudem wird erläutert, welche Spielertypen von welcher Spielmechanik angesprochen werden und gegebenenfalls wie eine Mechanik alternativ hätte umgesetzt werden können. Zur Erinnerung: bei den Spielertypen aus Kapitel 2.1.3 handelt es sich um vier grundsätzliche Eigenschaften, die einem Spieler zugeschrieben werden können. Der Competitor steht für den Konkurrenzkampf, der Achiever für das Sammeln von Auszeichnungen, der Socialiser für das Zusammen-

arbeiten mit anderen Spielern und der Explorer für das Erkunden seiner Umwelt. Jedem Spieler können dabei gleich mehrere oder gar alle Spielertypen zugeordnet werden.

### Punktesystem

In fast jeder Gamification-Umsetzung ist der Einsatz eines Punktesystems Pflicht. Zu dieser Ansicht kommt unter anderem die in Kapitel 2.1.4 vorgestellte Studie [23]. Der Grund dafür liegt wohl in der Tatsache, dass sich der Großteil der Spieler sehr gut durch Punktesysteme motivieren lassen. Zudem bauen viele weitere Gamification-Mechaniken auf einem Punktesystem auf, beispielsweise Ranglisten oder Auszeichnungen. Auch im Fall der hier vorgestellten App bietet sich der Einsatz eines Punktesystems an. So kann sich der Benutzer Punkte verdienen, indem er Todos bearbeitet. Diese Punkte werden anschließend seinem Punktekonto gutgeschrieben. In Kombination mit anderen Spielmechaniken lassen sich auf diese Weise gleich mehrere Spielertypen ansprechen. Um welche Spielertypen es sich dabei im Einzelnen handelt, wird in den Abschnitten Rangliste und Auszeichnungen beschrieben.

Häufig wird das Punktesystem noch um ein Stufensystem erweitert. Die Spieler steigen dann nach dem Erreichen von festgelegten Punktegrenzen jeweils eine Stufe auf. Ein solches System bringt jedoch einige Seiteneffekte mit sich. Erreicht der Spieler eine neue Stufe, erwartet er eine Belohnung, sonst wird ein solcher Stufenaufstieg für ihn wenig sinnvoll erscheinen. Die Anwendung sollte den Spieler folglich entweder intrinsisch und/oder extrinsisch belohnen, also entweder den Avatar des Spielers verstärken oder diesem eine neue Ausrüstung schenken. Beide Belohnungsarten machen allerdings nur Sinn, wenn der Spieler diese Dinge benötigt, um im Spiel voranzukommen. Das setzt wiederum voraus, dass der Spieler gegen andere Spieler oder irgendeine Art von Monstern kämpfen muss, sonst hat der Spieler nichts davon, dass sein Avatar stärker wird. Folglich müssten neben dem Stufensystem mindestens noch ein Kampfsystem, gegnerische Einheiten und ein Beutesystem implementiert werden.

All diese Spielmechaniken können für bestimmte Zielgruppen sehr motivierend sein, jedoch werden sich viele Spieler, die mit dem Rollenspiel-Genre nicht vertraut sind, überfordert fühlen. Da die hier entwickelte App allerdings die breite Masse ansprechen soll, wurde das Punktesystem absichtlich simpel gehalten und bewusst auf ein Stufensystem verzichtet.

### Rangliste

Bei Ranglisten handelt es sich um eine weitere Spielmechanik, die häufig bei Gamification-Anwendungen verwendet wird [23]. Um Ranglisten sinnvoll einsetzen zu können, wird der Einsatz eines Punktesystems vorausgesetzt, da ein Kriterium benötigt wird, anhand dessen die Spieler innerhalb der Rangliste sortiert werden. Durch Ranglisten werden aus den Spielern also Konkurrenten, da jeder Spieler möglichst viele Punkte erarbeiten möchte, um sich somit in den vorderen Plätzen der Ranglisten wiederzufinden. Aus diesem Grund werden durch Ranglisten also in erster Linie die Competitor angesprochen. Im Fall der hier vorgestellten App wirkt sich der Einsatz einer Rangliste zudem positiv auf die gesamte Gruppe aus. Wenn nämlich jeder Spieler motiviert ist, möglichst viele Punkte zu erreichen, werden folglich viele Todos von der Gruppe bearbeitet. Von diesem Umstand wiederum profitiert die gesamte Gruppe.

Der Grad zwischen Motivation und Belastung ist bei Ranglisten allerdings so schmal, wie bei kaum einer anderen Spielmechanik. Befindet sich ein Spieler beispielsweise uneinholbar weit vorne oder ist abgeschlagen auf dem letzten Platz, wird die Motivation bei den betroffenen Spielern stark sinken, frei nach dem Motto: „Wieso soll ich mich überhaupt noch anstrengen, ich kann die anderen sowieso nicht mehr einholen.“. Diese Situation kann zum Beispiel eintreten, wenn durch eine

Gruppe eine WG organisiert wird. zieht in die WG ein neuer Bewohner ein und schließt sich deshalb der WG-Gruppe an, so wird er mit hoher Wahrscheinlichkeit einen großen Punkterückstand auf die anderen Bewohner haben. Für einen solchen Fall ist es wichtig, die App um sogenannte Catch-Up-Mechaniken zu erweitern. Bei Catch Up-Mechaniken handelt es sich um Systeme, durch die Spieler, die zu einem späteren Zeitpunkt einem Spiel beitreten, den dadurch entstandenen Rückstand in relativ kurzer Zeit wieder aufholen können. In diesem Zusammenhang wurden zwei mögliche Ansätze betrachtet.

Entweder erhalten Spieler, die einen deutlichen Punkterückstand auf andere Spieler haben, mehr Punkte für das Bearbeiten von Todos und können dadurch auf die übrigen Spieler aufschließen, oder es wird ein Saisonsystem eingeführt. Bei diesem System werden die Punkte aller Spieler in einem festgelegten Intervall zurückgesetzt und jeder Spieler startet mit den gleichen Chancen in eine neue Saison, ähnlich wie es auch im Fußball üblich ist. Da es sich bei dem Saisonsystem nach Meinung des Entwicklers um das fairere System handelt, wurde dieses letztendlich implementiert. So können die Mitglieder einer Gruppe bestimmen, wann eine neue Saison anfangen und wie lange diese dauern soll. Ein großer Kritikpunkt des ersten Ansatzes ist, dass es sich bei diesem für die Spieler nicht lohnt, möglichst viele Punkte zu verdienen, da sie sich ohnehin keinen Vorsprung gegenüber den anderen Spielern erarbeiten können.

Auch bei der Implementierung der Rangliste wurden also einige Punkte berücksichtigt. Werden Ranglisten nämlich mit Bedacht eingesetzt, so sind sie ein mächtiges Werkzeug im Hinblick auf die Motivationssteigerung, andernfalls kann dieser Effekt schnell verpuffen oder sich sogar negativ auf die Spieler auswirken.

## Todos

Nachdem in den letzten Abschnitten sowohl das Punktesystem als auch die Ranglisten vorgestellten wurden, wird nun noch eine weitere Mechanik benötigt, durch die die angesprochenen Punkte verteilt werden. Dies erfolgt in Spielen typischerweise immer nach dem gleichen Schema. Der Spieler bekommt eine Aufgabe zugewiesen, bearbeitet diese und erhält anschließend unter anderem eine festgelegte Anzahl an Punkten als Belohnung. In der Regel bringen anspruchsvollere Aufgaben dem Spieler auch bessere Belohnungen ein. Diese Aufgaben werden in der Videospiel-Szene auch als Quests oder Todos bezeichnet. In der hier vorgestellten App wurde die Bezeichnung Todo gewählt, da sie für die meisten Benutzer wohl geläufiger ist.

Je nachdem in welchem Zusammenhang die Todos in einer Anwendung auftauchen, können diese gleich mehrere Spielertypen ansprechen. Müssen Aufgaben beispielsweise von mehreren Spielern gemeinsam bearbeitet werden, so kommen vor allem die Socialiser auf ihre Kosten. Geht es in einer Aufgabe hingegen darum, die Mitspieler zu übertrumpfen, fühlen sich in erster Linie die Competitor-Spieler angesprochen. Darüber hinaus lässt sich der Spielertyp Achiever für das Bearbeiten von Aufgaben begeistern, indem dieser durch den Abschluss von Aufgaben Auszeichnungen verdienen kann. Aus diesen Gründen gehören Todos in vielen Gamification-Umsetzungen deshalb zu dem Grundgerüst. Allerdings mussten auch an dieser Stelle wieder einige Entscheidungen bezüglich der Umsetzung getroffen werden.

So stand zu Beginn die Frage im Raum, ob die Benutzer die Anzahl der Belohnungspunkte frei wählen können oder aus einer vordefinierten Auswahl an Punktewerten auswählen müssen. Da der Entwickler nicht abschätzen kann, wie komplex die Aufgaben der Benutzer später einmal werden und diese nicht durch vorgegebene Punkte einschränken möchte, wurde letztendlich die erste Variante umgesetzt. Im Erstellungsformular für neue Aufgaben gibt es deshalb ein extra Textfeld, in das die gewünschte Punktzahl eingetragen werden kann. Als nächstes musste festgelegt werden, ob die Benutzer selbst bestimmen können, wann sie eine Aufgabe als erledigt markieren oder ob die übri-

gen Gruppenmitglieder diesen Part übernehmen sollen. Das hätte den Vorteil, dass niemand fälschlicherweise eine Aufgabe für abgeschlossen erklären kann und sich somit zu Unrecht die Punkte dafür gutschreiben lässt. Allerdings ist dieses Vorgehen gerade in großen Gruppen, in denen viele Aufgaben verteilt und bearbeitet werden, nur schwer umsetzbar, da die Gruppenmitglieder in diesem Fall wohl nur noch am Absegnen von Aufgaben wären und das eigentliche Spielgeschehen in den Hintergrund rücken würde. Deshalb besitzt der Benutzer in der fertigen App die Möglichkeit, selbst anzugeben, wenn er eine Aufgabe erledigt hat. Daraus ergibt sich jedoch ein neuer Aspekt, der berücksichtigt werden muss: wie soll mit Benutzern umgegangen werden, die Aufgaben fälschlicherweise für bearbeitet erklären?

Bemerken die anderen Gruppenmitglieder, dass ein Benutzer schummelt, so kann dieser bestraft werden, indem ihm einige Punkte abgezogen werden. Die Höhe der abgezogenen Punkte hängt dabei von der Punktzahl ab, die dem Spieler zu Unrecht zugeschrieben wurden. Damit die Gruppenmitglieder sich nicht wahllos gegenseitig Punkte abziehen können, wurde diese Funktion so implementiert, dass nur die Administratoren der Gruppen den anderen Benutzern Punkte abziehen können.

Zuletzt galt es, sich bezüglich dieser Spielmechanik zu überlegen, wie vermieden werden kann, dass sich ein Benutzer alle Aufgaben reserviert und die anderen somit keine Chance haben, ebenfalls Punkte zu erarbeiten. Hierfür wurde eine Begrenzung für die Anzahl der Aufgaben implementiert, die ein Benutzer gleichzeitig bearbeiten kann. Befinden sich bereits ausreichend viele Aufgaben in der Aufgabenliste des Benutzers, wird er durch ein Popup-Fenster darauf hingewiesen, dass er aktuell keine weiteren Aufgaben annehmen kann.

## Auszeichnungen

Bei der letzten in diesem Kapitel vorgestellten Spielmechanik handelt es sich um Auszeichnungen, die auch als Achievements bekannt sind. Diese können von den Spielern verdient werden, indem sie bestimmte vom Spiel vorgegebene Kriterien erfüllen. Häufig müssen die Spieler in diesen Fällen normale Aufgaben unter erschwerten Bedingungen meistern. So können die Spieler aufgefordert werden, Aufgaben unter Zeitdruck oder mit eingeschränkten Fähigkeiten abzuschließen. Oftmals müssen sich die Spieler in diesem Fällen deshalb intensiver mit dem Spiel auseinandersetzen. In der Regel erhalten sie jedoch keine Belohnungen in Form von Punkten oder Ausrüstungsteilen. Vielmehr ist das dadurch gesteigerte Ansehen bei den anderen Spielern als Belohnung zu betrachten. Vereinzelt erhalten die Spieler darüber hinaus Titel, mit denen sie ihren Namen für alle Spieler sichtbar schmücken können. So wurde in der hier vorgestellten App unter anderem eine Auszeichnung implementiert, durch die die Spieler den Titel „der Fleißige“ erhalten, nachdem sie sich diese Auszeichnung verdient haben.

Wie der Name dieser Spielmechanik bereits vermuten lässt, werden durch sie in erster Linie die Achiever angesprochen, da diese danach streben, möglichst viele Auszeichnungen zu erhalten. Allerdings können auch die Competitor für diese Mechanik motiviert werden, indem ihnen die Möglichkeit gegeben wird, ihre erreichten Auszeichnungen mit den übrigen Spielern zu vergleichen. Hierfür wurde die Rangliste der App um ein Feature erweitert. Jeder Spieler kann anhand dieser nun direkt sehen, wie viele Auszeichnungen die anderen Spieler bereits erreicht haben.

## Fazit

In der im Rahmen dieser Masterthesis entwickelten App wurden also insgesamt vier Spielmechaniken eingesetzt (ein Punktesystem, eine Rangliste, Todos und Auszeichnungen). Natürlich existieren darüber hinaus noch weitere Spielmechaniken, die alle ihre Vor- und Nachteile haben. In diesem Fall wurde jedoch bewusst darauf verzichtet, die App mit zu vielen Spielementen zu befüllen, da

einige Benutzer sich dadurch überfordert fühlen könnten. Aus diesem Grund wurde sich auf ein paar wenige Spielmechaniken beschränkt, die sich nach Meinung des Entwicklers am besten für diesen Kontext eignen.

### 5.1.7 Eingesetzte Speicherverwaltung

Heutzutage werden durch fast jede Anwendung Unmengen an Daten produziert. Der Einsatz einer effizienten Speicherverwaltung ist für einen reibungslosen Ablauf dieser Anwendungen deshalb unabdingbar. Nachdem in diesem Kapitel bereits ein Einblick in den Entwicklungsprozess bezüglich des Frontends und der eingesetzten Gamification-Mechaniken gegeben wurde, soll es nun um die verwendete Speicherverwaltung gehen. Auch an dieser Stelle mussten einige Iterationen durchlaufen werden, bis eine sinnvolle Lösung gefunden wurde.

Dieser Prozess lässt sich dabei grob in drei Phasen unterteilen, in deren Verlauf der Code der App zum Teil erheblich überarbeitet werden musste auf Grund der geänderten Speicherarchitektur. So wurden die Daten in einer ersten Version noch lokal auf dem Smartphone jedes Benutzers gespeichert. Erst in einer späteren Iteration wurden die Daten mit Hilfe von PHP-Skripten in einer zentralen Datenbank gespeichert. Da diese Lösung jedoch auch nicht ideal war, wurde letztendlich mit Apples CloudKit [3] die perfekte Lösung für die hier vorgestellte App gefunden.

Dieser Prozess soll im Folgenden genauer beschrieben werden. Des Weiteren wird am Ende dieses Abschnitts eine weit verbreitete und sehr beliebte Alternative kurz vorgestellt und es wird begründet, weshalb CloudKit in diesem Fall dennoch die bessere Lösung ist.

#### Erster Ansatz: Core Data

In der ersten Version der App wurden die generierten Daten lokal auf den Geräten der Benutzer gespeichert. Hierfür wurde das von Apple bereitgestellte Framework Core Data verwendet. Bei Core Data handelt es sich im Prinzip um eine lokale Datenbank, durch die die Daten auch über das Beenden einer App hinaus gespeichert werden können. Wird die App anschließend erneut gestartet, ist es weiterhin möglich, auf die gespeicherten Daten zuzugreifen. Core Data eignet sich also vor allem für Apps, die einen lokalen und persistenten Speicher benötigen, wie Todo-Listen Apps oder Spiele ohne Online-Funktionalitäten. Mit den Befehlen create, read, update und delete stellt das Framework den Entwicklern zudem die vier Standardbefehle zum Managen von Datenbanken zur Verfügung. Um Core Data verwenden zu können, muss bereits bei der Erstellung eines Projektes angegeben werden, dass Xcode dem Entwickler das Framework bereitstellen soll. Dadurch wird der Package Explorer um eine Datei erweitert. Hinter dieser Datei verbirgt sich das Core Data UI, in dem der Entwickler neue Entitäten anlegen kann. Jeder Entität können beliebig viele Attribute zugeordnet und der Datentyp für diese festgelegt werden. Die Entitäten entsprechen somit also den Tabellen von gewöhnlichen Datenbanken.

Zu Beginn des Entwicklungsprozesses war die Idee, zuerst das Grundgerüst der App zu implementieren und anschließend eine geeignete Speicherverwaltung zu finden. Da jedoch recht früh in der Entwicklung eine Möglichkeit benötigt wurde, um Daten zu speichern, wurde einfachheitshalber zunächst Core Data verwendet. Als die App sowohl inhaltlich als auch visuell in einem fortgeschrittenen Stadium war, wurde begonnen, sich hinsichtlich der Speicherverwaltung Gedanken zu machen. Die App war zu diesem Zeitpunkt bereits lauffähig und bot schon einen Großteil der finalen Funktionalitäten, allerdings wurden die Daten noch lokal und nicht wie gewünscht an einem zentralen Ort gespeichert. Als deshalb nach einer Lösung recherchiert wurde, die Daten zentral zu speichern, wurde schnell klar, dass viele Teile des bereits geschriebenen Codes nicht mehr verwendet werden konnten. Nichtsdestotrotz wurde von diesem Zeitpunkt ab an einer Version gearbeitet, bei der die Daten in einer zentralen Datenbank gespeichert werden.

## Zweiter Ansatz: MySQL-Datenbank und PHP-Skripte

Nachdem in der ersten Version der App die Daten lokal gespeichert wurden, sollte dieser Umstand in der nächsten Version korrigiert und die Daten zentral in einer Datenbank gespeichert werden. Leider hatte diese Neustrukturierung zur Folge, dass der bereits geschriebene Code im Grunde genommen komplett umgeschrieben werden musste. Jedoch war dies ein notwendiger Schritt, da die Benutzer sonst nicht miteinander interagieren können, was allerdings eines der Hauptfeatures der App darstellt. Nach einiger Recherche fiel die Wahl auf MySQL-Datenbanken und PHP-Skripte. Wie diese Umsetzung konkret aussah, wird im Folgenden beschrieben.

Bei MySQL [31] handelt es sich um ein weltweit sehr verbreitetes Open-Source-Datenbankverwaltungssystem. Es wurde im Jahr 1994 von dem schwedischen Software-Unternehmen MySQL AB entwickelt und 2010 von dem US-amerikanischen Unternehmen Oracle Corporation übernommen [11, 27]. PHP [48] (kurz für *Personal Home Page Tools*) ist eine Skriptsprache, die häufig verwendet wird, wenn Webanwendungen erstellt werden sollen. Beeinflusst wurde diese Sprache unter anderem von C++, Java und Perl. Sie wurde 1995 ebenfalls als freie Software veröffentlicht [19, 12]. Abbildung 25 kann entnommen werden, wie diese zwei Tools eingesetzt wurden, um die von der App generierten Daten zentral zu verwalten.

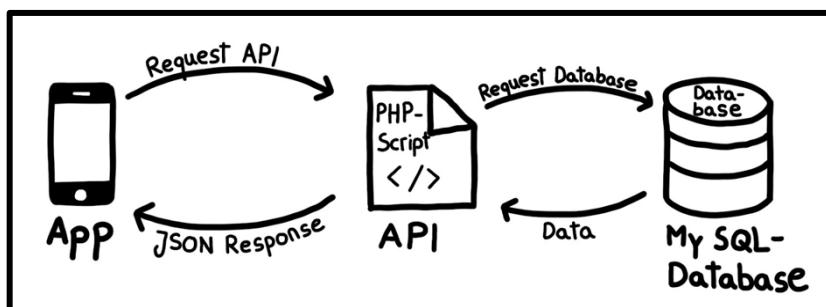


Abbildung 25: Schematischer Aufbau einer Speicherverwaltung mittels einer MySQL-Datenbank und PHP-Skripten.

Quelle: erstellt von Vanessa Schmall.

Wie anhand dieser Abbildung deutlich wird, sind Anwendungen selbst nicht in der Lage, direkt mit Datenbanken zu kommunizieren. Stattdessen wird eine sogenannte API benötigt. API steht für Application Programming Interface und beschreibt eine Schnittstelle zwischen zwei Systemen [13]. In der Regel stellen APIs Frameworks zur Verfügung, mit denen Daten aus einem System in ein anderes überführt werden können. Um also Daten auf einer Online-Datenbank speichern zu können, werden insgesamt drei Komponenten gebraucht: die Anwendung selbst, eine Schnittstelle und eine Datenbank. In diesem Fall kommuniziert die App mit Hilfe von PHP-Skripten mit einer MySQL-Datenbank. Hierfür werden im Swift-Code PHP-Skripte aufgerufen, die wiederum eine Anfrage mittels SQL-Query an die MySQL-Datenbank schicken. Die Datenbank sendet daraufhin die angefragten Daten im JSON-Format an die App zurück. In der App können die erhaltenen Daten anschließend weiterverarbeitet werden. Sollen einem Benutzer beispielsweise seine Todos in einer Tabelle angezeigt werden, wird im Code ein Skript aufgerufen, das die Datenbank um genau diese Informationen anfragt. Daraufhin schickt die Datenbank die gewünschten Daten als JSON verpackt an die App zurück. Dort wird das JSON-Dokument mit Hilfe einer Dekodier-Funktion in einen Swift-Datentyp umgewandelt und abschließend in einem Array gespeichert. In diesem Array stehen nun die Todos des Benutzers und können diesem angezeigt werden.

Im Grunde genommen wurde durch diese Architektur das angestrebte Verhalten umgesetzt; die Daten werden auf einer zentralen Datenbank gespeichert. Allerdings ergaben sich durch diesen Ansatz auch einige Probleme. Da die verwendeten Komponenten von dem Entwickler selbst zusammengestellt wurden, wäre dieser somit auch für die Sicherheit der Daten verantwortlich gewesen. Vor allem die eingesetzten SQL-Queries sind anfällig für Code-Injektion-Angriffe. Bei diesen Angriffen schleusen Angreifer ausführbaren Code in den Quellcode einer Anwendung, indem sie die SQL-Queries geschickt manipulieren. Auf diesem Weg gelangen die Angreifer beispielsweise an vertrauliche Daten aus den Datenbanken oder können diese vollständig löschen [9]. Da eine adäquate Sicherung der App den Rahmen dieser Masterthesis allerdings sprengen würde, kann die Sicherheit durch den hier gewählten Ansatz nicht gewährleistet werden.

Ein weiteres Problem war, dass für jeden Query-Typen, der an die Datenbank geschickt werden würde, ein extra PHP-Skript erstellt werden müsste. Das wären unter Umständen eine Menge geworden, wodurch das Projekt sehr unübersichtlich geworden wäre. Zudem erfordert der hier vorgestellte Ansatz einen Server, auf dem die Datenbank und die Skripte gelagert werden können. Dieser hätte also extra angemietet werden müssen, wodurch Kosten verursacht worden wären. Aus diesen Gründen wurde nach einer weiteren Alternative gesucht, die sowohl die Sicherheit für den Datenverkehr gewährleistet als auch einen Server anbietet, auf dem die Daten kostenlos gespeichert werden können.

### Dritter Ansatz: CloudKit

Durch die Einschränkung, dass das gesuchte Speicherverwaltungssystem sowohl den Sicherheitsaspekt übernehmen als auch den Server stellen soll, hat sich die Auswahl an möglichen Lösungsansätzen stark reduziert. So fiel die Entscheidung letztendlich auf das in Kapitel 2.4 bereits vorgestellte Framework CloudKit. Wie dort beschrieben wurde, ist die Verwendung dieses Frameworks weitgehend kostenlos. Da die Daten zudem in der iCloud gespeichert werden, übernimmt Apple selbst die Sicherung dieser Daten. Durch den Einsatz von CloudKit wurden also die größten Kritikpunkte des letzten Ansatzes behoben.

Sowohl die Entwicklungsumgebung Xcode als auch das CloudKit werden von Apple bereitgestellt, deshalb gestaltet sich die Integration von CloudKit in die verwendete Entwicklungsumgebung vergleichsweise unkompliziert. In den Einstellungen des Xcode-Projekts muss lediglich angegeben werden, dass CloudKit aktiviert werden soll. Anschließend kann durch eine einzige Zeile Code, wie im Abschnitt 3.5.2 beschrieben, eine Verbindung zu der Cloud und damit zu der Datenbank der entsprechenden App aufgebaut werden. Fortan ist es möglich, mit der iCloud mittels der vier Standarddatenbankbefehlen zu kommunizieren, welche ebenfalls in Abschnitt 3.5.2 beschrieben wurden. Damit jedoch überhaupt Daten zwischen der App und der Cloud ausgetauscht werden können, müssen in dem iCloud-Webinterface die in Kapitel 2.4 bereits angesprochenen Datentypen angelegt werden. In Abbildung 26 ist das Webinterface zu sehen. Im linken Kasten sind alle Datentypen aufgelistet, die innerhalb der App benötigt werden. Im rechten Kasten hingegen sind die Attribute aufgelistet, die die Datentypen beschreiben; in diesem Fall handelt es sich um die Attribute, die dem Datentyp Benutzer zugeordnet werden. Durch den Add Field- beziehungsweise New Type-Button lassen sich so beliebig viele Datentypen, inklusive den dazugehörigen Attributen erstellen. Durch das CloudKit können also sehr einfach neue Daten erstellt und zwischen der iCloud und der App ausgetauscht werden. Somit stellt es die optimale Lösung für die Speicherverwaltung dieser App dar.

The screenshot shows the 'Record Types' section of the iCloud developer portal. On the left, a sidebar lists 'System Types' (Users) and 'Custom Types' (Achievements, Events, Points, Quests, Residents). A blue box highlights the 'Residents' entry. Below the sidebar are buttons for '+ New Type' and 'Delete Type'. The main area is divided into two sections: 'System Fields' and 'Custom Fields'. The 'System Fields' section lists standard fields like 'recordName', 'createdBy', 'createdAt', etc., with their field type (e.g., Reference, Date/Time, String) and indexability (e.g., Queryable, None). The 'Custom Fields' section lists fields specific to the 'Residents' type: 'image', 'name', 'password', 'points', 'rank', 'todoCounter', and 'wg'. The 'points', 'rank', and 'todoCounter' fields are marked as 'Queryable, Searchable, Sortable'. At the bottom right of the 'Custom Fields' section is a link 'L Edit Indexes'.

Abbildung 26: Das iCloud-Webinterface, in dem neue Datentypen und Attribute erstellt werden können.

Wie anhand der letzten Abschnitte aufgezeigt wurde, existiert zum Thema Speicherverwaltung keine Lösung, die alle anderen dominiert. Vielmehr kursieren im Netz unzählige Ansätze, um Daten in Datenbanken zu speichern. Welche sich davon am besten eignen, hängt immer von dem spezifischen Anwendungsfall ab. So gibt es neben Apples CloudKit unter anderem noch eine weitere Lösung, die einen ähnlichen Funktionsumfang bietet und somit auch als Lösung für die hier implementierte App in Frage gekommen wäre.

Bei dem US-amerikanischen Cloud-Computing-Anbieter Amazon Web Services (kurz AWS) [2] handelt es sich um ein Tochterunternehmen von Amazon.com Inc., welches den Benutzern eine große Bandbreite an Diensten rund um das Thema Cloud-Computing zur Verfügung stellt. Unter anderem stellt AWS den Benutzern verschiedene Lösungen bereit, um Daten in Datenbanken zu speichern. Ähnlich wie bei Apples CloudKit wird der hierfür benötigte Server gestellt und der sichere Austausch der Daten von Amazon selbst gewährleistet. Allerdings gestaltet sich das Aufsetzen der Datenbank hier deutlich komplizierter als es bei CloudKit der Fall ist. Zudem sind die AWS-Dienste nicht komplett kostenlos. So entstanden bereits nach einigen Stunden des Testens die ersten Kosten durch die benutzten Dienste. Diese Kosten hätten sich im Laufe der Zeit immer weiter aufsummiert, da sich AWS alle Dienste pro Zeiteinheit bezahlen lässt. Aus diesen Gründen wurde sich letztendlich gegen die Lösung von AWS entschieden. Dies soll jedoch nicht bedeuten, dass die Dienste von AWS für diesen Anwendungsrahmen im Allgemeinen ungeeignet sind. Würde die App beispielsweise nicht nur für iOS-Geräte entwickelt werden, sondern auch für das Android-Betriebssystem, müsste die Situation neu bewertet werden und womöglich würden sich die AWS-Dienste in diesem Fall besser eignen. Dieses Beispiel verdeutlicht einmal mehr die Tatsache, dass es nicht „die eine Lösung“ gibt, die sich für jeden Anwendungsfall ideal ist.

### 5.1.8 Resümee des Entwicklungsprozesses

Das Ziel dieses Abschnitts war es, dem Leser den Entwicklungsprozess der App näherzubringen. So wurde beschrieben, wie die einzelnen Iterationen ausgesehen haben, die letztendlich zu dem finalen Design des User Interfaces geführt haben. Anschließend wurden die Gamification-Mechaniken vorgestellt, die innerhalb der App eingesetzt werden. In diesem Zusammenhang wurde auch begründet, weshalb die Mechaniken so umgesetzt wurden, wie es bei der fertigen App der Fall ist und weshalb neben den vier verwendeten Mechaniken auf weitere verzichtet wurden. Der letzte Teil dieses Abschnitts befasste sich mit der eingesetzten Speicherverwaltung. Auch an dieser Stelle

wurde der dafür durchlaufene Prozess genauer beschrieben. Neben den bereits geschilderten Schwierigkeiten während der Implementierung traten noch weitere zum Teil unerwartete Herausforderungen auf, für die Lösungen gefunden werden mussten. Diese Herausforderungen und deren Lösungen werden im nächsten Abschnitt vorgestellt.

## 5.2 Herausforderungen

Wie in vermutlich jedem größeren Projekt sind auch bei der Entwicklung der im Rahmen dieser Masterthesis erstellten App einige Herausforderungen aufgetreten, die nicht immer vorhersehbar waren und für die Lösungen gefunden werden mussten. In den folgenden Abschnitten wird auf einige dieser Herausforderungen eingegangen und es wird beschrieben, wie mit ihnen umgegangen wurde.

### 5.2.1 Direktes Anzeigen der angelegten Ereignisse

Wie bereits zuvor beschrieben wurde, können innerhalb der App Ereignisse angelegt werden, mit denen die Benutzer interagieren können. Das Anlegen dieser Ereignisse benötigt jedoch einige Augenblicke, da die erstellten Daten an die Datenbank geschickt werden müssen, bevor sie weiterverarbeitet werden können. Daraus ergibt sich allerdings ein grundsätzliches Problem, das anhand des folgenden Beispiels verdeutlicht werden soll. Es wird davon ausgegangen, dass ein Benutzer ein neues Todo anlegt, welches dieser daraufhin direkt annehmen möchte. Hierfür verwendet er das Erstellungsformular für Todos und gibt alle geforderten Informationen an. Anschließend erstellt er das Todo mit Hilfe des Erstellungsbutton und wird automatisch wieder auf das Todo-Tab geleitet. An dieser Stelle erwartet der Benutzer nun, dass ihm sein Todo direkt angezeigt wird, um dieses annehmen zu können. Aufgrund der bereits genannten Gründe ist ein solches Verhalten der App jedoch nicht möglich, da die Daten für den Transfer einige Zeit benötigen. Aus diesem Grund wurde nach einer Lösung gesucht, wie die erstellten Daten direkt für die Benutzer sichtbar und benutzbar sind. Um die Lösung für dieses Problem leichter nachvollziehen zu können, wird im Folgenden beschrieben, wie die Daten in diesem Fall gespeichert und angezeigt werden würden, wenn diese ausschließlich in der iCloud gespeichert werden.

Nachdem der Benutzer auf den Erstellungsbutton gedrückt hat, werden die Daten für dieses Todo an die iCloud geschickt, um dort einen neuen Eintrag in der entsprechenden Datentabelle anzulegen (siehe Abbildung 27, Schritt 1a). Im selben Augenblick wird der Benutzer auf das Todo-Tab zurückgeleitet, woraufhin die App eine Anfrage an die Cloud sendet und nach allen für den Benutzer relevanten Todos fragt (Schritt 2). Eine solche Anfrage wird jedes Mal durchgeführt, wenn auf das Todo-Tab gewechselt wird. Die angefragten Todos werden daraufhin in einem Array gespeichert und die Inhalte des Arrays dem Benutzer mit Hilfe einer Tabelle präsentiert (Schritt 3). Allerdings wird bei den angezeigten Todos nicht das von dem Benutzer gerade erstellte dabei sein, da dieses zu dem Zeitpunkt der Anfrage an die Datenbank noch gar nicht existiert hat. Das Problem besteht also darin, dass der Erstellungsbefehl für das Todo und der Anfragebefehl der App fast gleichzeitig an die Cloud geschickt werden. Um also das neue Todo sehen zu können, müsste der Benutzer nach der Erstellung des Todos auf ein anderes Tab wechseln, dort einige Augenblicke warten und anschließend auf das Todo-Tab zurückkehren. Die App würde dann erneut eine Anfrage für die entsprechenden Todos an die Datenbank schicken und der Benutzer letztendlich sein erstelltes Todo sehen. Zwar würde dieser Vorgang nur einige Sekunde benötigen, dennoch ist ein solches Verhalten der App für die Benutzer nicht hinnehmbar, da sie dadurch stark verwirrt werden würden.

Aus diesem Grund wurde sich dazu entschieden, die erstellten Todos neben der Cloud auch direkt in dem angesprochenen Array zu speichern, um sie so für die Benutzer direkt sichtbar zu machen. Konkret sieht die Lösung vor, dass an der Stelle, an der das Todo an die Cloud geschickt wird, ebenfalls eine Kopie dieses Todos an das Array gehängt wird (Schritt 1b), welches wie bereits erwähnt alle Todos enthält, die dem Benutzer angezeigt werden sollen. Das Anhängen eines Wertes an ein Array benötigt nämlich nur einen Bruchteil der Zeit, wie der Transfer dieser Daten an die Cloud. Auf diese Weise ist es möglich, dem Benutzer sein angelegtes Todo direkt anzuzeigen. Alle anderen Benutzer sehen dieses Todo, wenn sie in ihren Accounts einige Sekunden später auf das Todo-Tab wechseln.

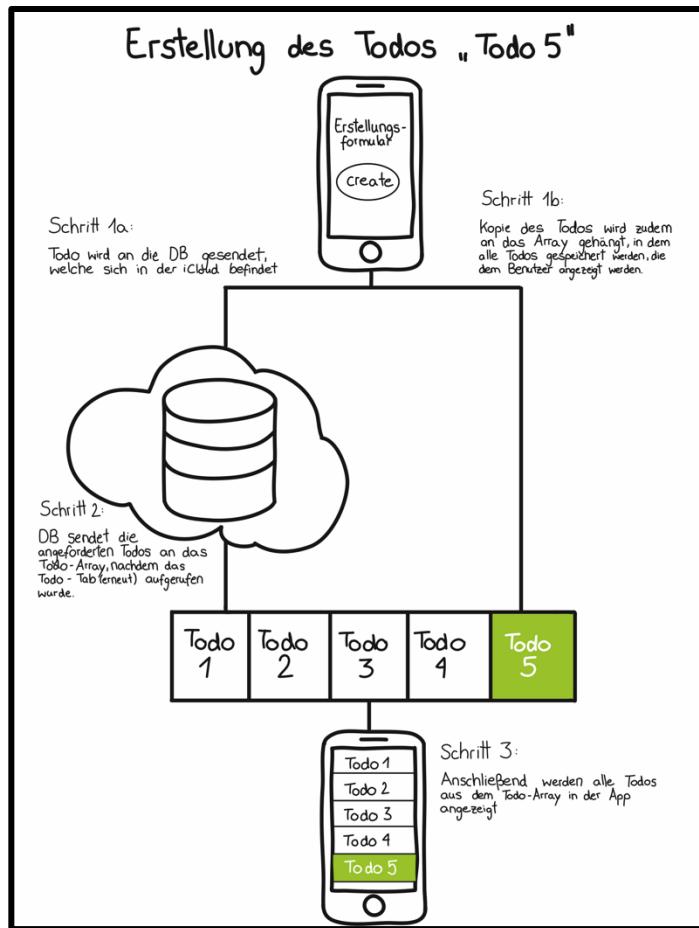


Abbildung 27: Dieses Schaubild skizziert den Ablauf der Todos-Erstellung. Quelle: erstellt von Vanessa Schmall.

Bei dem Erstellen neuer Accounts existiert prinzipiell das gleiche Problem. Allerdings ist der gerade genannte Kniff in diesem Fall nicht möglich und in den meisten Fällen wohl auch nicht notwendig. Würden die Zugangsdaten wie bei den Todos für kurze Zeit lokal gespeichert werden, bestünde die Gefahr, dass sich zwei oder mehr Benutzer zeitgleich einen Account mit dem selben Benutzernamen anlegen. Dies ist im Normalfall nicht möglich, da bei der Account-Erstellung geprüft wird, ob der angegebene Benutzername bereits verwendet wird. Letztendlich würde jedoch nur ein Account mit diesem Benutzernamen in der Datenbank angelegt werden. Die übrigen Benutzer könnten sich anschließend nicht mit ihren angegebenen Zugangsdaten einloggen, weshalb dieser Lösungsansatz keine Option darstellt. Darüber hinaus wird eine alternative Lösung wie bereits erwähnt im Grunde genommen auch gar nicht gebraucht. Nachdem der Benutzer seinen Account erstellt hat, benötigt

der Transfer dieser Daten an die Datenbank maximal zwei Sekunden. Anders als bei den Todos kann die Zeit in diesem Fall jedoch überbrückt werden, indem der Benutzer seine Zugangsdaten eingeben muss.

Auch bei dem Anlegen von neuen Gruppen wurde auf den Kniff verzichtet, der bei den Todos eingesetzt wurde. Zwar besteht bei den Gruppen das gleiche Problem, jedoch ist eine kurze Wartezeit für die Benutzer in diesem Fall eher hinnehmbar, da Gruppen deutlich seltener erstellt werden als Todos. Aus diesem Grund wird der Benutzer, nachdem er eine neue Gruppe angelegt hat, durch eine Lade-Animation aufgefordert zu warten, bis die Erstellung der Gruppe abgeschlossen ist.

### **5.2.2 Verwaltung der Gruppenprofile eines Benutzers**

Ursprünglich war geplant, die Anzahl der verschiedenen Datentypen, die innerhalb der iCloud verwendet werden, möglichst gering zu halten, um die Speicherverwaltung nicht unnötig komplex werden zu lassen. Aus diesem Grund wurde ein Datentyp angelegt, der sämtliche Informationen zu den Benutzern beinhalten sollte. So umfasste dieser Datentyp die Attribute Name, Passwort, Punktzahl, Profilbild, Rang und gesammelte Auszeichnungen der einzelnen Benutzer. Die Informationen zu den Todos und Gruppen wurden mit Hilfe von zwei weiteren Datentypen verwaltet. Alle durch die App generierten Daten wurden also durch die drei Datentypen Benutzer, Todos und Gruppen realisiert. Die Aufgliederung der Daten in diese drei Datentypen erschien auf den ersten Blick sinnvoll. Nachdem jedoch die erste Version implementiert und getestet wurde, in der die Daten mit Hilfe dieser Datentypen verwaltet wurden, zeigte sich schnell, dass sich dieser Lösungsansatz nicht für die hier vorgestellte App eignet.

Das Problem bei dieser Variante besteht nämlich darin, dass völlig außer Acht gelassen wurde, dass die Benutzer in mehreren Gruppen gleichzeitig sein können. Schließt sich ein Benutzer einer neuen Gruppe an, so werden alle Angaben (Punktzahl, Rang und gesammelte Auszeichnungen) zur alten Gruppe überschrieben und somit unwiderruflich gelöscht. Wenn ein Benutzer beispielsweise Mitglied von zwei Gruppen ist, so würde sein gesamter Fortschritt jedes Mal gelöscht werden, sobald dieser in die jeweils andere Gruppe wechselt. Da der Entwickler den Benutzern jedoch nicht vorgeben möchte, in wie viele Gruppen diese gleichzeitig aktiv sein können, ist die oben genannte Aufteilung der Datentypen für diese Situation nicht sinnvoll. Stattdessen wurde ein weiterer Datentyp namens GruppenProfil hinzugefügt, durch den die relevanten Informationen der Benutzer zu den einzelnen Gruppen gespeichert werden (Name des Benutzers, Punktzahl, Rang, gesammelte Auszeichnungen und der Name der Gruppe). Der Umfang des Datentyps Benutzer wurde deshalb stark reduziert, sodass dieser nur noch verwendet wird, um den Namen, das Passwort und das Profilbild eines Benutzers zu speichern. Wie es möglich ist, mit diesem Ansatz die Daten eines Benutzers zu mehreren Gruppen gleichzeitig zu speichern, wird im Folgenden beschrieben.

Nachdem der Benutzer sich einen Account erstellt hat, wird ein neuer Eintrag des Datentyps Benutzer angelegt, durch den die Zugangsdaten des Benutzers gespeichert werden. Tritt dieser anschließend einer Gruppe bei, wird zudem ein neuer Eintrag des Datentyps GruppenProfil erstellt, in dem wie bereits erwähnt unter anderem der Name des Benutzers und der Gruppe gespeichert werden. Die Zuordnung des Benutzers zu den einzelnen Gruppenprofilen erfolgt demnach über den Benutzernamen, da dieser bei beiden Datentypen gespeichert wird. Auf diese Weise können die Benutzer beliebig vielen Gruppen beitreten, ohne Gefahr laufen zu müssen, dass ihre Daten verloren gehen, sobald sie die Gruppe wechseln. Da im Datentyp GruppenProfil sowohl der Name des Benutzers als auch der der Gruppe gespeichert werden, ist es relativ leicht, die gewünschten Informationen zu den Benutzern korrekt aus der iCloud auszulesen und in der App anzeigen zu lassen.

### **5.2.3 Dark Mode durch das iOS 13er-Update**

Als Apple am 19. September 2019 die Version (13.0) seines iOS-Betriebssystems veröffentlichte, freuten sich viele Benutzer über die neueingeführten Features. Für App-Entwickler bedeutete dieses Update allerdings, dass jede App aktualisiert werden musste. Verantwortlich dafür war die Einführung des sogenannte Dark Modes. Der Dark Mode stellt eine Alternative zu dem sonst eher hell gestalteten User Interface von Apple dar. In diesem Modus werden alle hellen Flächen durch dunkle ersetzt, wodurch das User Interface, wie der Name bereits vermuten lässt, ein deutlich dunkleres Erscheinungsbild erhält. Auch die Popup-Fenster und der Statusbalken am oberen Rand des Displays, in dem unter anderem die Uhrzeit und der Batteriestand angezeigt werden, werden von diesem Modus beeinflusst. Durch die Aktivierung des Dark Modes kann es also dazu kommen, dass das ursprüngliche von den Entwicklern erarbeitete Design nicht mehr stimmig aussieht, aufgrund der gerade erwähnten Veränderungen.

Unglücklicherweise veröffentlichte Apple dieses Update erst, nachdem das Design für die hier erstellte App bereits final umgesetzt worden war, weshalb das Design erneut überarbeitet werden musste. Für diesen Fall existieren zwei mögliche Lösungsansätze. Entweder es wird neben dem Design für den hellen Modus noch ein weiteres erstellt, das an den Dark Mode besser angepasst ist und ausgewählt wird, sobald der Dark Mode aktiviert wird oder der Dark Mode wird in der App unterdrückt. Da die verwendeten Bilder, Icons und Farben wie in Kapitel 5.1.1 bereits erwähnt nicht sonderlich gut mit einem dunklen Design harmonieren, entschied sich der Entwickler für die zweite Variante und dafür den Dark Mode zu unterdrücken. Hierfür muss lediglich die von Apple bereitgestellte Abfrage `if available(iOS 13.0, *) { overrideUserInterfaceStyle = .light }` in die viewDidLoad-Methode jedes App-Screens eingefügt werden. Durch diese Zeile Code wird abgefragt, ob die Softwareversion des verwendeten Gerätes größer oder gleich 13 ist. Wenn ja, wird der Light Mode innerhalb der App aktiviert. Durch diesen Trick kann das Problem, das der Dark Mode für App-Betreiber mit sich bringt, relativ leicht gelöst werden.

Durch das Kapitel 5.2 sollte aufgezeigt werden, dass unabhängig von dem betriebenen Planungsaufwand eines Softwareprojekts jederzeit Probleme auftreten können, für die Lösungen gefunden werden müssen. Einige von ihnen können nicht vorhergesehen werden, da sie sich erst durch veränderte Umwelteinflüsse während der Entwicklung ergeben (Dark Mode), andere wiederum werden erst während der Entwicklung bemerkt, weil nicht alle möglichen Szenarien im Vorhinein abgedeckt wurden. Gerade bei Entwicklerteams, die, wie in diesem Fall, nur aus einer Person bestehen, ist es fast unmöglich, alle relevanten Szenarien im Vorfeld zu beachten.

Auch wenn jedes auftretende Problem letztendlich einzigartig ist und deshalb individuell gelöst werden muss, repräsentieren die drei vorgestellten Probleme den Großteil der Herausforderungen, die es im Laufe dieses Projekts zu lösen galt.

## **5.3 Implementierung einer iOS-Beispiel-App**

Im letzten Abschnitt dieses Kapitels soll der Leser anhand einer kleinen Beispiel-App einen besseren Einblick in die App-Entwicklung mit Swift und Xcode erhalten. Hierfür wird im Folgenden gezeigt, wie die grundlegenden Komponenten der hier vorgestellten App implementiert wurden. So wird unter anderem gezeigt, wie in Xcode ein neuer Screen erzeugt und dieser mit Elementen gefüllt wird. Um den Rahmen dieser Masterthesis nicht zu sprengen und Redundanzen zu vorherigen Kapiteln zu vermeiden, soll es in diesem Abschnitt ausschließlich um die Grundlagen gehen.

### 5.3.1 Umfang der Beispiel-App

Mit der App, die im Verlauf dieses Abschnittes entwickelt werden soll, soll es möglich sein, die Flächeninhalte von Rechtecken und Kreisen zu bestimmen. Hierfür kann der Benutzer wahlweise die Kantenlänge oder den Radius angeben und erhält anschließend den Flächeninhalt des entsprechenden Objekts. Die berechneten Flächeninhalte werden ihm anschließend in einer Tabelle angezeigt. Die App sollte also über insgesamt zwei Tabs verfügen: in einem werden die Flächeninhalte berechnet und in dem anderen tabellarisch aufgelistet. Mit Buttons, Textfeldern, Tabellen und mehreren Tabs umfasst die Beispiel-App einen Großteil der Komponenten, die in der im Rahmen dieser Masterthesis programmierten App ebenfalls verwendet wurden. Diese Beispiel-App eignet sich deshalb gut, um dem Leser einen Einblick in die Entwicklung der eigentlichen App zu geben.

### 5.3.2 Anlegen eines neuen Projektes

Bereits bei der Projekterstellung muss sich der Entwickler für ein Template entscheiden. Abhängig von der Funktionalität der App eignen sich einige Templates besser als andere. Sie unterscheiden sich in der Anzahl und Vernetzung der voreingestellten Screens. Da die Beispiel-App mehrere Tabs umfassen soll, wird das Tabbed App-Template ausgewählt (siehe Abbildung 28).

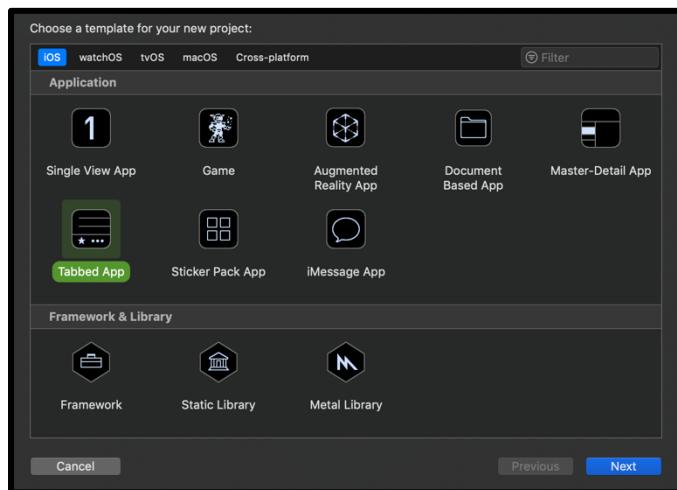


Abbildung 28: Erstellungsdialog eines neuen Projektes in Xcode.

Anschließend müssen noch weitere Informationen bezüglich des Projektes angegeben werden, wie beispielsweise der Name oder der Speicherort. Nachdem das Projekt erstellt wurde, erhält der Entwickler innerhalb der Entwicklungsumgebung einen Überblick über das gesamte Projekt. Wie in Abbildung 29 zu sehen ist, werden im Package Explorer alle Dateien aufgelistet, die das Projekt zu diesem Zeitpunkt umfasst. Neben dem *Main.storyboard* sind vor allem die zwei Dateien *FirstViewController.swift* und *SecondViewController.swift* für den Entwickler von Interesse. Xcode hat die zwei Swift-Klassen aufgrund des ausgewählten Templates bereits eigenständig mit den dazugehörigen Screens verbunden (siehe Pfeile in Abbildung 29). Im Hauptfenster von Xcode befindet sich das gerade erwähnte Main Storyboard der App. In diesem werden alle Screens und deren Verbindungen dargestellt. Da das Tabbed App-Template ausgewählt wurde, wurden von Xcode bereits einige Screens automatisch generiert. Bei dem linken Screen in Abbildung 29 handelt es sich um den Initialisierungsscreen, welcher direkt nach dem Starten der App angezeigt wird. Die beiden rechten Screens stellen wiederum alle Szenen dar, aus denen die App zu diesem Zeitpunkt besteht. Bei Bedarf können dem Storyboard beliebig viele Screens hinzugefügt werden und so der Umfang der App erweitert werden. Die Verbindungen, die zwischen den Screens zu sehen sind, weisen darauf hin, dass ein

Übergang zwischen diesen besteht. Es ist bei diesem Beispiel also möglich, von dem Initialisierungsscreen aus zu den beiden anderen zu navigieren. Xcode stellt den Entwicklern dabei verschiedene Arten von Übergängen zur Verfügung. In diesem Fall werden die Übergänge, wie der Name des Templates bereits vermuten lässt, durch das Klicken auf die Tab Bar im unteren Bereich der einzelnen Screens ausgelöst. Da für den Umfang der Beispiel-App zwei Screens ausreichen, muss an dieser Stelle kein weiterer hinzugefügt werden. Stattdessen kann direkt mit der Erstellung der App begonnen werden.

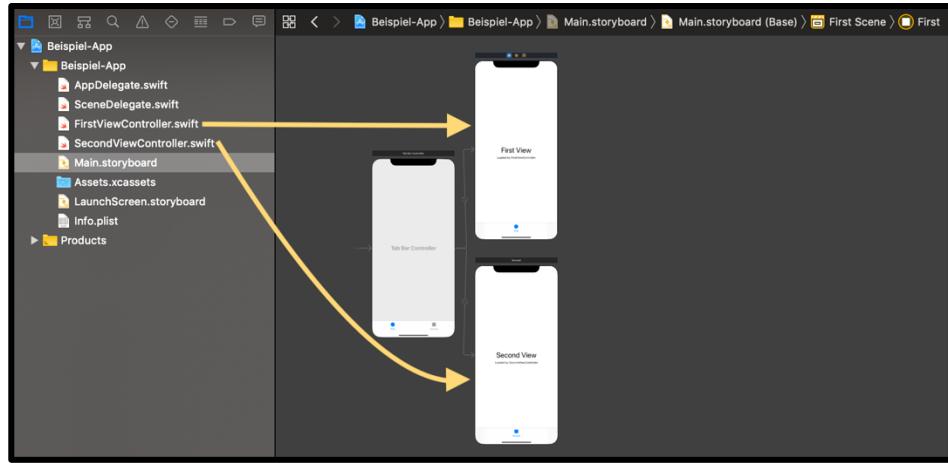


Abbildung 29: Das Storyboard der Beispiel-App unmittelbar nach der Projekterstellung.

### 5.3.3 Befüllen der Screens mit Inhalten

Die App-Entwicklung mit Xcode läuft normalerweise immer nach dem gleichen Prinzip ab: zuerst wird das Aussehen der App erstellt, sprich alle Buttons, Bilder, Tabellen, etc. werden via Drag and Drop an den vorgesehenen Positionen platziert. Im nächsten Schritt erhalten die positionierten Elemente ihre Funktionalität, indem diese in den dazugehörigen Swift-Klassen programmiert wird. In der Regel verläuft die Implementierung einer App selten linear ab. Vielmehr werden auch hier mehrere Iterationen benötigt, in denen die zwei Phasen immer wieder durchlaufen werden. Der Umfang der Beispiel-App ist jedoch so gering, dass in diesem Fall eine Iteration ausreichend ist. Um einem Screen Elemente hinzuzufügen, wird der Library-Button benötigt (oben rechts in Abbildung 30). Klickt der Entwickler auf diesen, werden ihm alle Elemente angezeigt, die auf dem Screen platziert werden können. Per Drag and Drop werden die Elemente anschließend an die gewünschte Stelle gezogen. Soll die App auf mehreren Geräten mit unterschiedlich großen Displays laufen, ist diese Art der Positionierung allerdings nicht ratsam. In diesem Fall sollten alle Elemente relativ zueinander platziert werden, da das Layout der App auf einigen Geräten sonst nicht wie gewünscht aussehen wird. Für die Beispiel-App wird jedoch davon ausgegangen, dass diese nur auf einem Gerät laufen wird, weshalb die Elemente einfach an die vorgesehenen Stellen gezogen wurden. In Abbildung 30 ist das Ergebnis der ersten Phase zu sehen. Alle benötigten Elemente wurden so angeordnet, wie sie in der fertigen App zusehen sein sollen. Jedes Element kann darüber hinaus individuell angepasst werden mit Hilfe der Einstellmöglichkeiten, die sich am rechten Rand befinden. Das Table View, also die Tabelle, in der die berechneten Flächeninhalte angezeigt werden sollen, wird sein finales Aussehen hingegen erst dann erhalten, wenn die App in Betrieb genommen wird. Das liegt daran, dass davor noch keine Werte existieren, mit denen die Tabelle gefüllt werden kann. Bis dahin wird lediglich ein Platzhalter für die Tabelle angezeigt.

Die erste Phase ist mit dem Platzieren aller Elemente abgeschlossen und es kann begonnen werden die Logik für die App zu programmieren.

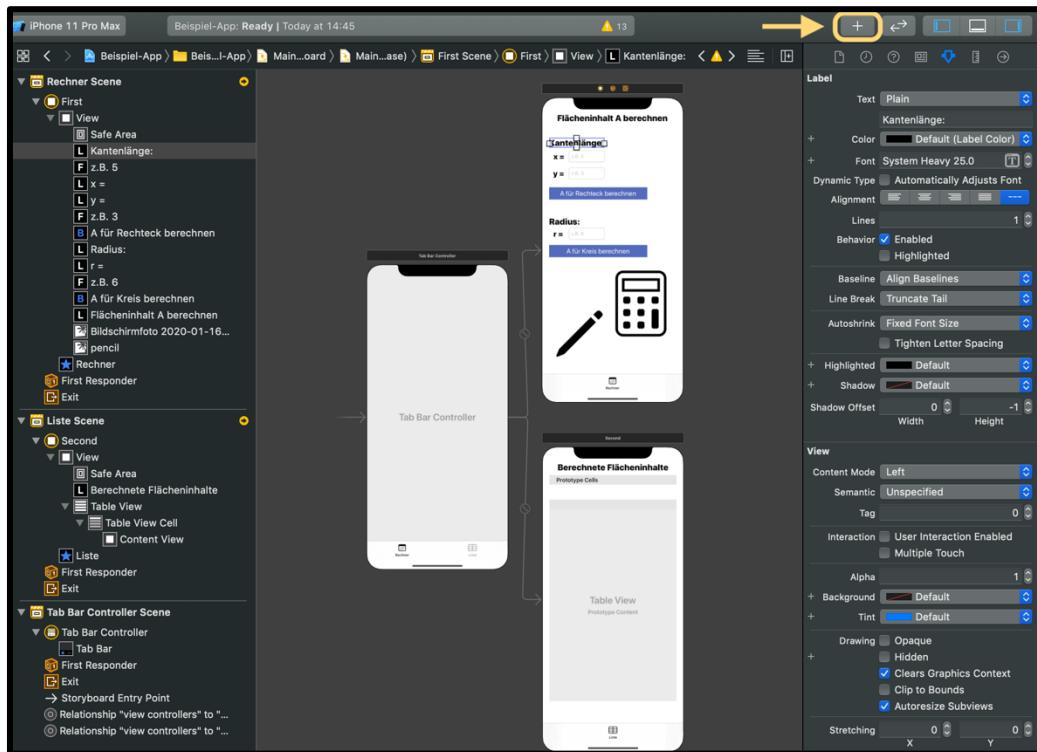


Abbildung 30: Das Storyboard der Beispiel-App, nachdem die Screens mit Elementen gefüllt wurden.

### 5.3.4 Implementierung der Logik der App

Nachdem im letzten Abschnitt beschrieben wurde, wie das Aussehen der App mit Xcode erstellt wurde, ist es nun an der Zeit, das Backend zu programmieren, um der App ihre Funktionalität zu verleihen. Hierfür müssen alle gerade platzierten Elemente mit den dazugehörigen Swift-Klassen verbunden werden. Wie Abbildung 31 entnommen werden kann, werden die Elemente nacheinander in den Programmcode gezogen. Im Anschluss öffnet sich jeweils ein Dialogfenster, in dem einerseits der Name und andererseits der Typ (siehe Kapitel 3.1.3: IBOutlet oder IBAction) der zu erstellenden Referenz angegeben werden muss. Durch einen Klick auf den Connect-Button (Kasten in Abbildung 31) wird die Referenz erstellt. Durch diese Referenz ist es möglich, das Element anzusprechen und sein Verhalten zu programmieren.

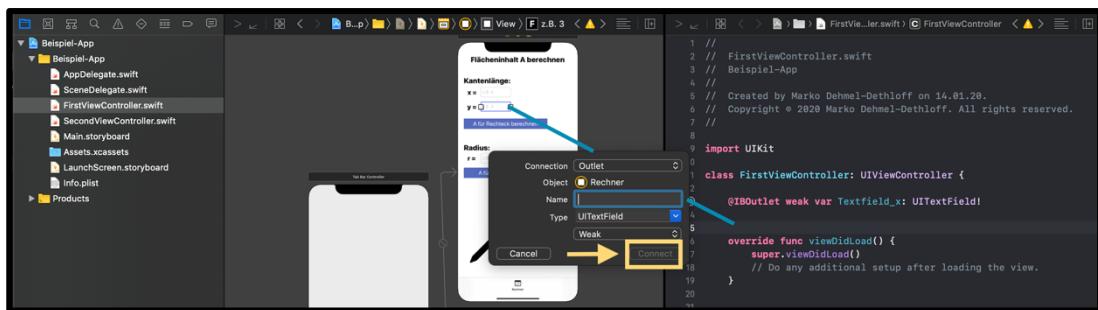


Abbildung 31: Verbinden eines Elementes mit der dazugehörigen Swift-Klasse.

Nachdem alle Elemente auf diese Weise mit ihren Swift-Klassen verbunden wurden, kann mit der eigentlichen Programmierung begonnen werden. Da es sich bei der Beispiel-App wie bereits erwähnt um eine eher einfache App handelt, fallen die zwei implementierten Klassen kurz aus, sodass sie im Ganzen gezeigt werden können. Im Folgenden werden beide Klassen präsentiert und im Anschluss jeweils erläutert. Auf eine detaillierte Beschreibung der Klassen wird an dieser Stelle allerdings verzichtet, um Redundanzen zu vorherigen Kapiteln zu vermeiden. Vielmehr soll grob erklärt werden, wie die jeweilige Klasse aufgebaut ist und welchen Zweck sie hat. Begonnen wird mit der Swift-Klasse, die dem ersten Tab zugeordnet wurde, also dem, in dem die Flächeninhalte berechnet werden sollen.

```
1 class FirstViewController: UIViewController {
2
3     @IBOutlet weak var Textfield_width: UITextField!
4     @IBOutlet weak var Textfield_height: UITextField!
5     @IBOutlet weak var Textfield_r: UITextField!
6     static var Array: [Float] = []
7
8     override func viewDidLoad() {
9         super.viewDidLoad()
10    }
11    // Diese Funktion wird immer dann ausgeführt, wenn der „A für Rechteck berechnen“-Button gedrückt wird
12    @IBAction func FlaecheninhaltFuerRechteck(_ sender: UIButton) {
13        if(Textfield_width.text!.isEmpty || Textfield_height.text!.isEmpty) {
14            print("Mindestens ein Wert fehlt!")
15        } else {
16            let X: Int = Int(Textfield_width!.text!)!
17            let Y: Int = Int(Textfield_height!.text!)!
18            let temp: Float = Float(X * Y) *-1
19            FirstViewController.Array.append(temp)
20        }
21    }
22    // Diese Funktion wird immer dann ausgeführt, wenn der „A für Kreis berechnen“-Button gedrückt wird
23    @IBAction func FlaecheninhaltFuerKreis(_ sender: UIButton) {
24        if(Textfield_r.text!.isEmpty) {
25            print("Mindestens ein Wert fehlt!")
26        } else {
27            let R: Int = Int(Textfield_r!.text!)!
28            let temp: Float = Float(Float.pi * Float(R * R))
29            FirstViewController.Array.append(temp)
30        }
31    }
32 }
```

Programmcode 1: Programmcode der Klasse FirstViewController.

Zu Beginn der Klasse FirstViewController sind die Outlet-Referenzen für die Textfelder des Be-rechnungsscreens zu sehen (Zeilen 3-5). Die Textfelder werden benötigt, damit der Benutzer die Kantenlängen eines Rechtecks (*Textfield\_x* und *Textfield\_y*) beziehungsweise den Radius eines Krei-

ses (*Textfield\_r*) in die App eingeben kann. Diese Informationen werden später gebraucht, um die Flächeninhalte zu berechnen. In Zeile 6 wird zudem ein statisches Array des Typs Float angelegt. In diesem Array sollen alle berechneten Flächeninhalte gespeichert werden und bei Bedarf in einer Tabelle angezeigt werden. Da der exakte Flächeninhalt von Kreisen in der Regel nicht durch die natürlichen Zahlen ausgedrückt werden kann, sondern durch die rationalen Zahlen, wurde sich gegen den üblichen Typ Int und für den Typ Float entschieden. Nach der Initialisierungsfunktion der App (Zeilen 8-10) folgt in den Zeilen 12-21 die Actionfunktion für den ersten Button. Wird dieser von dem Benutzer geklickt, soll der Flächeninhalt eines Rechtecks mit den angegebenen Kantenlängen berechnet und anschließend an das erstellte Array gehängt werden. Hierfür wird in Zeile 13 abgefragt, ob die benötigten Textfelder ausgefüllt wurden. Ist das nicht der Fall, wird auf der Konsole eine Fehlermeldung angezeigt. Ansonsten werden die Werte aus den vorgesehenen Textfeldern ausgelesen und in Konstanten abgespeichert (Zeilen 16-17). In Zeile 18 findet dann die eigentliche Berechnung für den Flächeninhalt statt und das Ergebnis wird zu einem Float gecastet. Weshalb das Ergebnis im Anschluss mit „-1“ multipliziert wird, wird klar, wenn der Code der nächsten Klasse vorgestellt wird. Mit dem Befehl aus der letzten Zeile dieser Funktion (Zeile 19) wird der eben berechnete Wert an das Array gehängt.

Die Actionfunktion für den zweiten Button umfasst die Zeilen 23-31. Diese ist im Grunde genommen genauso aufgebaut wie die Funktion für den ersten Button, nur wird in diesem Fall der Flächeninhalt für Kreise berechnet.

```

1  class SecondViewController: UIViewController, UITableViewDataSource, UITableViewDelegate {
2
3      @IBOutlet weak var TabelleFuerFlaecheninhalte: UITableView!
4      var index: Int = 0
5
6      override func viewDidLoad() {
7          super.viewDidLoad()
8          TabelleFuerFlaecheninhalte.delegate = self
9          TabelleFuerFlaecheninhalte.dataSource = self
10     }
11
12     override func viewDidAppear(_ animated: Bool) {
13         index = 0
14         self.TabelleFuerFlaecheninhalte.reloadData()
15     }
16     // Diese Funktion ermittelt die Anzahl der Einträge für die Tabelle.
17     func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
18         return FirstViewController.Array.count
19     }
20     // Diese Funktion bestimmt den Inhalt der einzelnen Einträge.
21     func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
22         let cell = TabelleFuerFlaecheninhalte.dequeueReusableCell(withIdentifier: "cell", for: indexPath)
23
24         if(FirstViewController.Array[index] < 0) {
25             cell.textLabel?.text = "Flächeninhalt für Rechteck: \(- (FirstViewController.Array[index]))"
26         } else {
27             cell.textLabel?.text = "Flächeninhalt für Kreis: \(FirstViewController.Array[index])"
28         }
29     }
30 }
```

```

28         }
29         index += 1
30         return cell
31     }
32 }

```

Programmcode 2: Programmcode der Klasse SecondViewController.

In der Klasse SecondViewController wird die Logik für das zweite Tab programmiert. In diesem sollen die berechneten Flächeninhalte mittels einer Tabelle aufgelistet werden. Zudem soll bei jedem Eintrag stehen, ob es sich um ein Rechteck oder ein Kreis handelt. Aus diesem Grund erbt die Klasse von den Protokollen UITableViewDataSource und UITableViewDelegate. Diese werden benötigt, um die Tabelle nach den eigenen Wünschen modifizieren zu können. Bei den Zeilen 3-4 handelt es sich um die Referenz der Tabelle und eine Variable, die innerhalb der Klasse benötigt wird. Anschließend folgen die Initialisierungs- und Updatefunktionen der App (Zeilen 5-15). Durch die zwei übrigen Funktionen werden das Aussehen und der Inhalt der Tabelle implementiert, wobei die erste Funktion (Zeilen 17-19) die Anzahl der Einträge und die zweite (Zeilen 21-31) den Inhalt der einzelnen Einträge festlegt. Da die Anzahl der Tabelleneinträge gleich der Größe des Arrays sein sollte, das in der Klasse FirstViewController angelegt wurde, gibt die erste Funktion die Länge dieses Arrays zurück (Zeile 18). Zur Erinnerung: dieser Wert bestimmt, wie häufig die zweite Funktion aufgerufen wird.

Durch die zweite Funktion werden anschließend die einzelnen Einträge und deren Inhalte erstellt. In Zeile 22 wird deshalb zuerst einmal ein neuer Eintrag angelegt. Die Inhalte der Tabelleneinträge werden wiederum in den Zeilen 24-28 bestimmt. Um unterscheiden zu können, ob es sich bei dem aktuellen Arrayeintrag um den Flächeninhalt eines Rechtecks oder Kreises handelt, muss lediglich das Vorzeichen des jeweiligen Eintrages überprüft werden (Zeile 24). Ist der in diesem Durchlauf betrachtete Arrayeintrag kleiner als Null, gehört der Flächeninhalt zu einem Rechteck, ansonsten zu einem Kreis. Diese Unterscheidung ist möglich, da die Flächeninhalte für Rechtecke bei deren Berechnung alle negiert wurden. Der konkrete Inhalt für die Tabelleneinträge wird letztendlich wahlweise in Zeile 25 oder Zeile 27 erstellt. In den Zeilen 29-30 wird abschließend die Variable *index* um eins erhöht und der gerade erstellte Eintrag zurückgegeben und somit der Tabelle hinzugefügt. Die *index* Variable wird benötigt, um über das Array iterieren zu können.

Mit der fertigen Implementierung der zwei gezeigten Klassen ist die Erstellung der Beispiel-App abgeschlossen. Das Ergebnis ist in Abbildung 32 zu sehen.

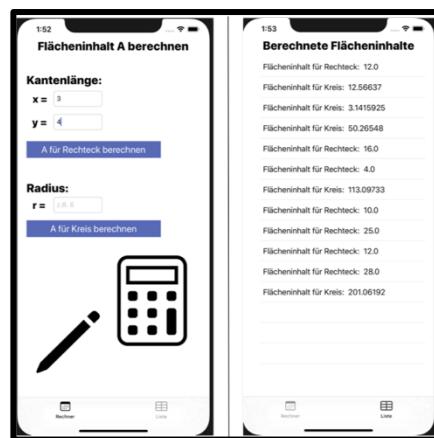


Abbildung 32: Die beiden Screens der fertig implementierten Beispiel-App.

---

Die App könnte nun theoretisch in Apples App-Store hochgeladen und von jedermann heruntergeladen werden. Natürlich gibt es noch viele Verbesserungen, die vorgenommen werden könnten. Für dieses Beispiel wurde jedoch ganz bewusst darauf verzichtet, damit der Umfang nicht den Rahmen dieser Masterthesis sprengt. Ziel dieses Abschnitts war es nämlich nicht, dem Leser die App-Erstellung mit Xcode im Detail zu erklären, vielmehr sollte diesem das allgemeine Vorgehen nähergebracht werden.

In diesem Kapitel wurde dem Leser die Entwicklung der App nähergebracht. So wurde zu Beginn beschrieben, weshalb die in Kapitel 4 aufgestellten Anforderungen so umgesetzt wurden, wie es in der finalen Version der App letztendlich der Fall ist. Des Weiteren wurden die Herausforderungen, die während der Entwicklung aufgetreten sind, vorgestellt und es wurde erläutert, wie mit diesen umgegangen wurden. Das Kapitel endete mit einem Beispiel, anhand dessen der grobe Verlauf einer App-Entwicklung mit Xcode gezeigt wurde.

## **6. Aufbau und Ergebnisse der Benutzerstudie**

Ein sehr wichtiger Aspekt der App-Entwicklung ist das Evaluieren der Software, die in diesem Zusammenhang programmiert wurde. Zu diesem Zweck wurde eine Benutzerstudie durchgeführt und die Ergebnisse im Anschluss ausgewertet und interpretiert. Für die Benutzerstudie wurde eigens eine App erstellt, die die Benutzer in das Thema einführen soll und anschließend bezüglich der im Rahmen dieser Masterthesis entwickelten App befragt. Durch die so gewonnenen Daten soll die eingangs gestellte Frage beantwortet werden, ob sich die App positiv auf das Verhalten der Benutzer auswirkt und wenn ja, wie stark. Dieses Kapitel beschäftigt sich also mit dem Aufbau, der Durchführung und den Teilnehmern der Benutzerstudie. Darüber hinaus werden die gewonnenen Daten präsentiert und am Ende dieses Kapitels die Auswirkungen der App auf die Benutzer evaluiert.

### **6.1 Aufbau, Durchführung und Teilnehmer der Benutzerstudie**

Für die Durchführung der Benutzerstudie wurde neben der eigentlichen App noch eine zweite App erstellt, die die Probanden durch die Studie leiten soll. Für den Entwickler gab es in erster Linie zwei Faktoren, die gegen herkömmliche Fragebögen sprachen und für die Implementierung einer eigenen Benutzerstudie-App: wenn der Fragebogen nicht gerade selbstständig mit Word oder einem ähnlichen Programm erstellt wird, werden für diesen in der Regel weitere Kosten anfallen. Spätestens jedoch, wenn die durch den Fragebogen generierten Daten außerdem online gespeichert werden sollen, handelt es sich dabei fast immer um einen kostenpflichtigen Service. Des Weiteren war dem Entwickler wichtig, den Probanden ein Tool zur Verfügung zu stellen, das diese durch die komplette Benutzerstudie führt.

#### **6.1.1 Aufbau der App für die Benutzerstudie**

Wie gerade erwähnt, soll die App so gestaltet werden, dass sie die Probanden durch die Benutzerstudie leitet und somit die Teilnehmer die Studie weitestgehend selbstständig durchführen können. Zu diesem Zweck besteht die App aus insgesamt drei Abschnitten. Zu Beginn erhalten die Probanden eine kurze Einleitung in die Benutzerstudie und in das Thema, mit dem sich die vorliegende Arbeit befasst. So wird an dieser Stelle unter anderem erwähnt, dass im Rahmen dieser Masterthesis eine App entwickelt wurde, deren Auswirkungen auf die Benutzer evaluiert werden sollen. In dem zweiten Abschnitt werden den Probanden anschließend einige Aufgaben gestellt, die sie mit Hilfe der eigentlichen App bearbeiten sollen. Nachdem die Probanden alle Aufgaben erfolgreich abgeschlossen haben, gelangen sie zu dem Fragebogen der Benutzerstudie. Sind sie mit diesem ebenfalls fertig, erscheint der End-Screen der App, in dem sich der Entwickler für die Teilnahme bedankt. Zudem werden die ausgefüllten Fragebögen an dieser Stelle automatisch an die iCloud des Entwicklers geschickt. Um das Arbeiten mit der Benutzerstudie-App für die Probanden so angenehm wie möglich zu gestalten, wurde sich auch in diesem Fall Gedanken bezüglich des Designs gemacht. Auf diese soll im Folgenden kurz eingegangen werden.

Damit die Ergebnisse der Benutzerstudie möglichst zuverlässig und wahrheitsgetreu sind, sollten die Probanden konzentriert und motiviert bei der Durchführung sein. Häufige bei Studien gemachte Fehler, sind einerseits jedoch, dass die Dauer eines Durchlaufes zu hoch ist, oder dass die Probanden im Unklaren gelassen werden, wie weit sie bereits fortgeschritten sind und wie viel sie dementsprechend noch vor sich haben [7]. Infolgedessen neigen Teilnehmer dazu, ungeduldig zu werden und die Studie möglichst schnell abzuschließen, wodurch die Qualität der Ergebnisse abnimmt. Aus diesem Grund wurde die App so konstruiert, dass für einen Durchlauf maximal zehn Minuten benötigt werden. Zudem wird den Probanden ihr aktueller Fortschritt visualisiert. So befindet sich am oberen Rand des Displays durchgängig ein Banner, in dem der Name des aktuellen Abschnitts

und der Fortschritt innerhalb diesem angezeigt wird (siehe Abbildung 33). Zur besseren Unterscheidung erhalten die Banner der einzelnen Abschnitte verschiedene Farben. Dadurch wissen die Teilnehmer immer direkt, wann sie einen neuen Abschnitt erreicht haben und können einordnen, an welchem Punkt der Benutzerstudie sie sich gerade befinden. Darüber hinaus wurde darauf geachtet, dass sich der Umgang mit der Benutzerstudie-App intuitiv anfühlt. Die Probanden sollen unter keinen Umständen durch die App verwirrt oder von der eigentlichen Studie abgelenkt werden. So erklärt die App den Probanden immer genau, was als nächstes zu tun ist. Die Knöpfe, um sich innerhalb der Studie vor- oder rückwärts zu bewegen, befinden sich links beziehungsweise rechts unten in den Ecken. Diese Art sich durch einen Dialog zu bewegen sollte dem Großteil der Teilnehmer bereits vertraut sein. Durch die gerade genannten Maßnahmen sollte die App in der Lage sein, die Probanden während der Studie sinnvoll zu unterstützen.

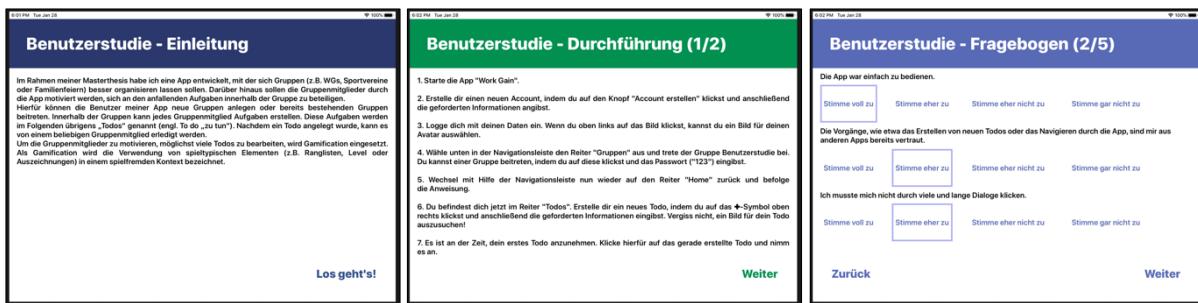


Abbildung 33: Links: die Einleitung in die Studie (erster Abschnitt der Studie). Mitte: ein Screen des zweiten Abschnitts. Rechts: ein Teil des Fragebogens (letzter Abschnitt).

### 6.1.2 Aufbau und Ablauf der Benutzerstudie

Für die Durchführung der Studie benötigten die Probanden zwei iOS-betriebene Geräte. Diese wurden von dem Entwickler bereitgestellt (ein iPad und ein iPhone). Auf dem iPad lief die App, die die Teilnehmer durch die Benutzerstudie führt und auf dem iPhone die im Rahmen dieser Masterthesis entwickelte App. Über diese Ausgangssituation klärte der Entwickler die Probanden in einer kurzen Einführung auf. Des Weiteren wurde ihnen gesagt, dass die Studie aus den drei Abschnitten Einleitung, Durchführung und Fragebogen besteht. So wissen die Teilnehmer gleich zu Beginn, was auf sie zu kommt, wodurch die Situation für sie vorhersehbarer wird und sie somit gelassener an die Studie gehen können [7]. Nach diesen Ankündigungen platzierte der Entwickler sich so im Raum, dass er weder sehen konnte, was die Teilnehmer auf den Geräten machen, noch die Teilnehmer selbst. Blickkontakt zu ihnen baute er nur dann auf, wenn die Probanden Fragen hatten. Ansonsten bearbeiteten die Teilnehmer ganz auf sich alleine gestellt die Benutzerstudie. Nach Abschluss der eigentlichen Benutzerstudie entstanden häufig Gespräche, in denen die Teilnehmer informell über ihre Erfahrungen mit dem Thema Gamification berichteten und Ideen äußerten, welche Inhalte sie sich für die App noch wünschen. Auch wenn diese Gespräche im Vorhinein nie geplant waren, entpuppten sie sich als sehr informativ für den Entwickler, sodass dieser viele neue Features sammeln konnte, die früher oder später umgesetzt werden sollen.

### 6.1.3 Anonymisierung der Ergebnisse

Ein kritischer Aspekt bei jeder Benutzerbefragung ist, wie ehrlich die Teilnehmer auf die gestellten Fragen antworten. Das hängt häufig von gleich mehreren Faktoren ab, etwa ob die Teilnehmer beobachtet werden, sie die Fragen überhaupt verstehen oder bestimmte Antwortmöglichkeiten gesellschaftlich geächtet werden [7]. Um von den Probanden möglichst ehrliches und somit brauchbares Feedback zu erhalten, wurden deshalb bezüglich der Durchführung einige Punkte berücksichtigt.

Wie bereits beschrieben wurde, war dem Entwickler wichtig, dass die Teilnehmer die Benutzerstudie weitgehend eigenständig durchführen können. Dadurch ist dieser nicht gezwungen, während der Studie direkt neben den Probanden zu sitzen und kann diese folglich nicht überwachen. Darüber hinaus wurde den Teilnehmern gleich zu Beginn mitgeteilt, dass die im Laufe der Benutzerstudie generierten Fragebögen anonymisiert an den Entwickler geschickt werden. Jeder Fragebogen muss zwar eine einzigartige ID besitzen, um an die iCloud geschickt zu werden, jedoch wurde diese zufällig erzeugt und anschließend gehasht. Dadurch, dass die Fragebögen alle digital ausgefüllt wurden, können die Probanden zudem nicht durch ihre Handschrift identifiziert werden. Der Entwickler hat also keine Möglichkeit zu ermitteln, welcher Fragebogen von welchem Teilnehmer stammt. Durch diese Maßnahmen verspricht sich der Entwickler ehrlicheres Feedback.

#### **6.1.4 Teilnehmer der Benutzerstudie**

Da die App, die im Rahmen dieser Masterthesis erstellt wurde, keine spezifische Bevölkerungsgruppe ansprechen soll, sondern vielmehr die breite Masse, wurde bei der Auswahl der Probanden darauf geachtet, dass jede Altersgruppe und jedes Geschlecht vertreten ist. So gaben von den insgesamt 24 Teilnehmern zehn „männlich“, 12 „weiblich“, einer „divers“ und ein weiterer „keine Angabe“ als Geschlecht an. Auch bei den Altersgruppen (11-20, 21-30, 31-40, 40+) war jede vertreten, wobei die Altersgruppe der 21- bis 30-jährigen mit 14 Personen am stärksten repräsentiert war. Mit 13 zu 11 war das Verhältnis von Videospielern zu nicht-Videospielern sehr ausgeglichen.

Bei dem Großteil der Probanden handelte es sich um Verwandte und Bekannte des Entwicklers. Jedoch konnten auch einige dem Entwickler völlig fremde Personen für die Benutzerstudie gewonnen werden. Diese wurden auf dem Gelände der Technischen Universität Darmstadt angesprochen und erklärten sich daraufhin bereit, an der Studie teilzunehmen. Die vergleichsweise geringe Anzahl an Teilnehmern wird damit begründet, dass aufgrund des Studienaufbaus der Entwickler bei jedem Durchlauf mit seinen iOS-Geräten vor Ort sein musste. Es war also nicht möglich, die Benutzerstudie an eine größere Masse zu schicken, da die für die Studie benötigte Software zu diesem Zeitpunkt noch in der Entwicklung war und deshalb nur auf ausgewählten Geräten lief. Zudem verfügen die meisten Teilnehmer ohnehin nicht über ein iOS-betriebenes Gerät, weshalb diese nicht an der Studie hätten teilnehmen können.

#### **6.1.5 Aufbau des Fragebogens**

Der Fragebogen, den die Teilnehmer am Ende der Benutzerstudie ausfüllen sollten, umfasste insgesamt 13 Fragen. Bei den Fragen handelte es sich um 12 geschlossene und eine offene Frage. In der Literatur werden Fragen als geschlossen bezeichnet, wenn der Proband aus vorgegebenen Antwortmöglichkeiten wählen kann [38]. Bei offenen Fragen werden hingegen keine Antworten vorgegeben. Stattdessen soll der Proband bei diesen Fragen seine Antwort frei formulieren. Durch offene Fragen können den Probanden somit zwar Informationen entlockt werden, die allein durch geschlossene Fragen nicht erreichbar sind, allerdings ist das Beantworten von offenen Fragen auch deutlich zeitintensiver für die Teilnehmer [30]. Zudem schrecken viele Probanden vor diesem Fragetypen zurück, da es ihnen schwer fällt, ihre Gedanken präzise niederzuschreiben [7]. Offene Fragen sollten deshalb nur vereinzelt, möglichst nicht zu Beginn und optional gestellt werden. Werden die Probanden gezwungen, eine freie Antwort geben zu müssen, können sie schnell frustriert werden, wodurch die Aussagekraft des gesamten Fragebogens negativ beeinflusst werden kann. Aus diesem Grund wurde in diesem Fall entschieden, nur eine offene Frage am Ende des Fragebogens optional zu stellen. Durch sie gibt der Entwickler den Probanden die Möglichkeit, ihre Meinung zur App in ihren eigenen Worten aufzuschreiben, ohne Gefahr zu laufen, dass er sie mit dieser Frage frustriert.

Die Reihenfolge der übrigen 12 Fragen wurden nach dem Trichterprinzip festgelegt. Dieses besagt, dass ein Fragebogen mit den einfachen Fragen beginnt und diese im Verlauf immer anspruchsvoller werden [7]. Dieses Prinzip sollte allerdings nur bei eher kurzen Fragebögen verwendet werden, da die Konzentration der Teilnehmer bei langen Fragebögen zum Ende hin zwangsläufig nachlässt. Des Weiteren wurden die Fragen thematisch geordnet. So werden zu Beginn allgemeine Fragen gestellt, um die Probanden charakterisieren zu können. Anschließend folgen Fragen zur Bedienbarkeit der App, gefolgt von Fragen, die sich mit den eingesetzten Gamification-Mechaniken befassen. Abschließend geht es um die Auswirkungen, die die App auf die Teilnehmer hat, bevor der Fragebogen mit der bereits erwähnten offenen Frage endet.

Bei jedem Fragebogen stellt sich erneut die Frage, wie viele Antwortmöglichkeiten den Probanden zur Verfügung gestellt werden sollen. Gerade bei Fragen, bei denen die Teilnehmer einer Aussage anhand einer Skala zustimmen oder widersprechen sollen, macht es einen großen Unterschied, ob eine gerade oder ungerade Anzahl von Antworten angeboten werden. Skalen mit ungerade vielen Antwortmöglichkeiten besitzen ein mittleres / neutrales Element. Probanden, die eine Tendenz zur Unentschiedenheit besitzen werden, vermutlich häufig auf dieses Element ausweichen, wodurch der Informationsgehalt sinkt. Bei geraden Skalen gibt es ein solches Ausweich-Element nicht. Hier sind die Teilnehmer gezwungen, sich für eine Tendenz zu entscheiden, auch wenn sie sich eigentlich der Mitte zuordnen würden [18]. Beide Ansätze haben also ihre Vor- und Nachteile, weshalb nicht pauschalisiert werden kann, welcher sich besser für Fragebögen eignet. Vielmehr sollte für jede Frage innerhalb eines Fragebogens individuell entschieden werden, welcher Skalentyp in diesem Fall der sinnvollere ist. Bei Fragen, bei denen es zumutbar war, dass sich die Probanden für eine eindeutige Tendenz festlegen, wurde eine gerade Skala gewählt, wohingegen Fragen, bei denen ein mittleres Element als Antwort durchaus legitim war, eine ungerade Skala erhalten haben. So wurden etwa bei der Frage „Die eingesetzten Gamification-Mechaniken wurden sinnvoll gewählt?“ eine gerade Anzahl von Antworten angeboten („Stimme voll zu“, „Stimme eher zu“, „Stimme eher nicht zu“, „Stimme gar nicht zu“). Die Aussage „Die Anzahl der eingesetzten Gamification-Mechaniken war...“ wiederum besitzt eine ungerade Skala („zu hoch“, „genau richtig“, „zu gering“). Durch dieses Vorgehen sollen durch den generierten Fragebogen einerseits möglichst viele Informationen gesammelt werden, ohne die Probanden jedoch zu ungewollten Aussagen zu drängen.

Des Weiteren wurde darauf geachtet, die Fragen eindeutig und für die Teilnehmer verständlich zu stellen. Doppelfragen, wie „Hat dir die App gefallen und wie findest du den Home-Screen?“ wurden deshalb vermieden. Auch wurde nur Vokabular verwendet, das den Probanden bekannt ist, weshalb die beiden benötigten Fremdwörter „Gamification“ und „Todo“ zu Beginn der Benutzerstudie eingeführt wurden. Damit der Entwickler besser abschätzen kann, ob die Teilnehmer den Begriff Gamification richtig verstanden haben, wurde eine Kontrollfrage in den Fragebogen integriert. Bei dieser sollten die Probanden alle Gamification-Mechaniken auswählen, die innerhalb der App eingesetzt wurden. Dadurch kann der Entwickler schnell die unbrauchbaren Fragebögen aussortieren. Der Informationsgehalt eines ausgefüllten Fragebogens zu einem Thema, das der Teilnehmer nicht verstanden hat, ist schließlich sehr gering.

Durch die im Vorfeld der Benutzerstudie getätigten Überlegungen und der daraus resultierten Umsetzung, erhoffte sich der Entwickler die Durchführung für die Teilnehmer so angenehm wie möglich zu gestalten, um von diesen somit unverfälschtes und konstruktives Feedback zu erhalten. Die Ergebnisse der Studie werden in dem nächsten Abschnitt präsentiert.

## 6.2 Ergebnisse der Benutzerstudie

Der Fragebogen lässt sich grob in die vier Bereiche „demographische Eigenschaften der Teilnehmer“, „Bedienbarkeit der App“, „verwendete Gamification-Mechaniken“ und „Auswirkungen der

App auf die Benutzer“ unterteilen. In den folgenden vier Abschnitten werden die Ergebnisse jedes Bereiches präsentiert. Diese Zahlen dienen als Grundlage für das Kapitel 6.3, in dem die Ergebnisse anschließend interpretiert werden.

### 6.2.1 Demographische Eigenschaften der Teilnehmer

Von den 24 Teilnehmern haben 13 (54%) angegeben, gelegentlich Videospiele oder Spiele-Apps zu spielen. 11 (46%) Probanden spielen nach eigenen Angaben hingegen gar keine digitalen Spiele. Von den 13 Spielern, haben wiederum fünf (38%) angegeben maximal einmal pro Woche zu spielen. Weitere fünf (38%) Teilnehmer ordneten sich der Gruppe zwei- bis viermal zu. Insgesamt drei (24%) Probanden gaben an, sogar häufiger als viermal pro Woche zu spielen.

Mit 12 (50%) weiblichen Teilnehmern stellte dieses Geschlecht zudem die größte Gruppe dar, gefolgt von zehn (42%) männlichen. Die Antwortmöglichkeiten „divers“ und „keine Angabe“ wurden jeweils einmal (4%) gewählt. Auch bei der Altersangabe der Beteiligten wurde jede Kategorie mindestens einmal ausgewählt. So gibt es eine (4%) Person, die in die Gruppe der 11- bis 20-jährigen fällt. 14 (58%) Probanden und somit der Großteil der Befragten ordneten sich der Gruppe „21 bis 30 Jahre“ zu. Zwischen 31 und 40 Jahren alt sind hingegen nur zwei (8%) Teilnehmer. In die letzte Gruppe „älter als 40 Jahre“ fallen noch einmal sieben (30%) Probanden.

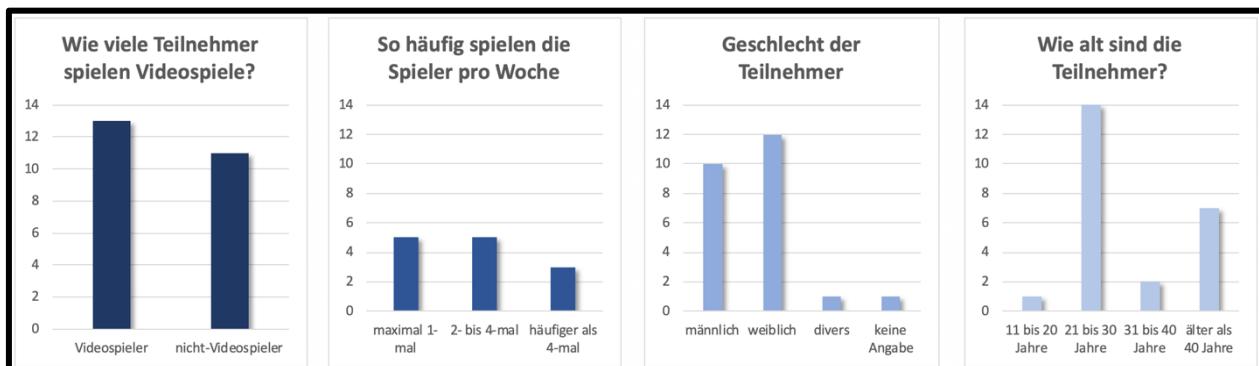


Abbildung 34: Die Ergebnisse des ersten Abschnitts, dargestellt in Diagrammen.

### 6.2.2 Bedienbarkeit der App

Durch den zweiten Abschnitt des Fragebogens sollte ermittelt werden, wie leicht den Probanden das Arbeiten mit der App fiel. Hierfür wurden ihnen drei Aussagen präsentiert, denen sie anhand einer Vierer-Skala zustimmen oder widersprechen sollten. So stimmten der Aussage „Die App war einfach zu bedienen.“ 12 (50%) der befragten Personen voll zu. Weitere 12 (50%) entschieden sich für die Antwortmöglichkeit „Stimme eher zu“. Kein Teilnehmer wählte hingegen die Antworten „Stimme eher nicht zu“ oder „Stimme gar nicht zu“. Die zweite Aussage „Die Vorgänge, wie etwa das Erstellen von neuen Todos oder das Navigieren durch die App, sind mir aus anderen Apps bereits vertraut.“ befürworteten 15 (63%) Probanden mit der Antwort „Stimme voll zu“. Sechs (25%) Teilnehmer legten sich fest, der Aussage eher zuzustimmen. Für die Antwort „Stimme eher nicht zu“ entschieden sich in diesem Fall drei (12%). Keiner der Teilnehmer wählte auch bei dieser Aussage die Antwortmöglichkeit „Stimme gar nicht zu“. Auf die letzte Aussage „Ich musste mich nicht durch viele und lange Dialoge klicken.“ entgegneten 12 (50%) Probanden „Stimme voll zu“. Mit zehn (42%) Teilnehmer entschieden sich fast genauso viele für die Antwort „Stimme eher zu“. Zwei (8%) stimmten dieser Aussage eher nicht zu. Gar nicht zugestimmt hat auch dieser Aussage kein Proband.

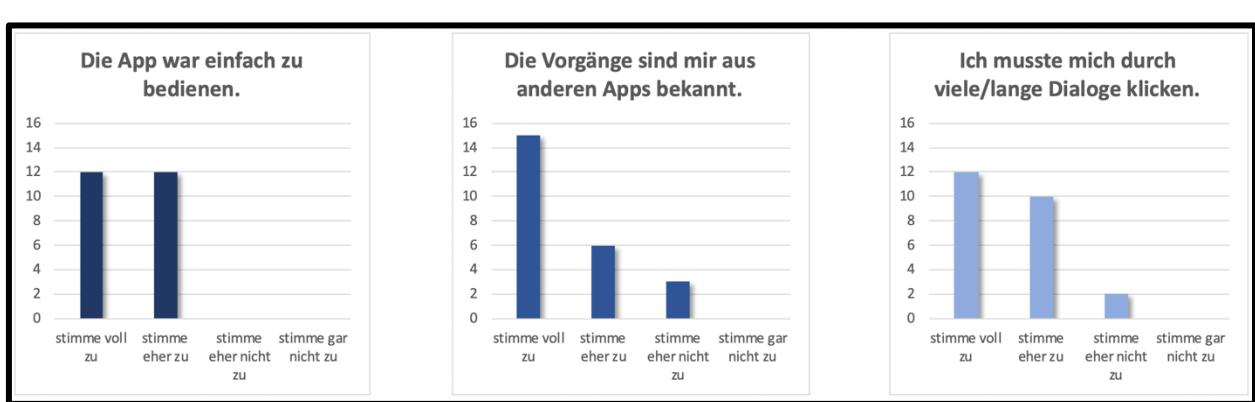


Abbildung 35: Die Ergebnisse des zweiten Abschnitts des Fragebogens visualisiert durch Diagramme.

### 6.2.3 Verwendete Gamification-Mechaniken

Im dritten Abschnitt wurden die Probanden bezüglich der eingesetzten Gamification-Mechaniken befragt. Da es an dieser Stelle interessant sein kann, wie sich die Aussagen zwischen den Videospielern und den nicht-Videospielern unterscheiden, werden diese im Folgenden separat betrachtet. Die Prozentzahlen in den Klammern beziehen sich deshalb auch immer auf den Anteil der gerade betrachteten Gruppe.

Bei der ersten Frage in diesem Abschnitt handelte es sich um die bereits erwähnte Kontrollfrage „Welche Gamification-Mechaniken werden in der App eingesetzt?“. Von den Videospielern entschieden sich fast alle für die zwei richtigen Antworten „Rangliste“ (12, 92%) und „Auszeichnungen“ (13, 100%). Die übrigen zwei Antworten „Zeitdruck“ und „Kampfsystem“ wurden korrekteweise gar nicht gewählt. Ein sehr ähnliches Bild zeichnet sich bei den nicht-Videospielern ab. Hier haben sich mit 11 (100%) Probanden für „Rangliste“ und zehn (91%) für „Auszeichnungen“ ebenfalls nahezu alle für die zwei richtigen Antworten entschieden. Auch hier wählte kein Teilnehmer eine der falschen Antworten.

Als nächstes sollten die Probanden der Aussage „Die Anzahl der eingesetzten Gamification-Mechaniken war...“ zustimmen oder widersprechen. Unter den 13 Videospielern empfanden 11 (85%) Teilnehmer die Anzahl für „genau richtig“. Keiner war der Meinung, dass zu viele Gamification-Mechaniken verwendet wurden, jedoch hätten sich zwei (15%) Probanden mehr Mechaniken gewünscht. Von den 11 nicht-Videospielern entschieden sich insgesamt sieben (64%) für die Antwort „genau richtig“. Die Antworten „zu hoch“ wurden innerhalb dieser Gruppe dreimal (27%) gewählt. Als zu niedrig empfand es ein (9%) Teilnehmer.

Bei der letzten Frage wollte der Entwickler wissen, ob die eingesetzten Gamification-Mechaniken sinnvoll gewählt wurden. Sechs (46%) Videospieler stimmten dieser Aussage voll zu. Die übrigen sieben (54%) entschieden sich für die Antwort „Stimme eher zu“. Die Antwortmöglichkeiten „Stimme eher nicht zu“ und „Stimme gar nicht zu“ wurden hingegen gar nicht gewählt. Von den nicht-Videospielern stimmten in diesem Fall vier (36%) voll zu. Für die Antwort „stimme eher zu“ entschieden sich bei dieser Gruppe ebenfalls sieben (64%) Teilnehmer. Auch in diesem Fall werden die durch den Fragebogen gewonnenen Ergebnisse mit Hilfe von Diagrammen visualisiert. Die blaugefärbten Diagramme stellen dabei immer die Werte der Videospieler dar. Bei den grüngefärbten Diagrammen handelt es sich hingegen um die Zahlen der nicht-Videospieler. Begonnen wird mit den Diagrammen zur ersten Frage dieses Abschnittes.

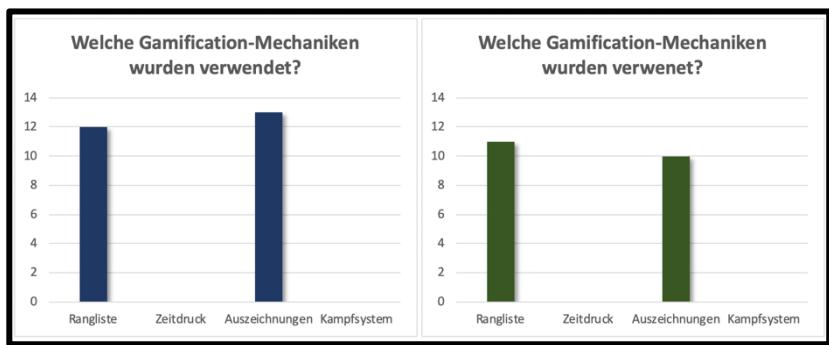


Abbildung 36: Die Ergebnisse der ersten Frage des dritten Abschnitts dargestellt in Diagrammen.

Im Folgenden werden die Ergebnisse der zweiten und dritten Frage mit Hilfe von Diagrammen visualisiert.

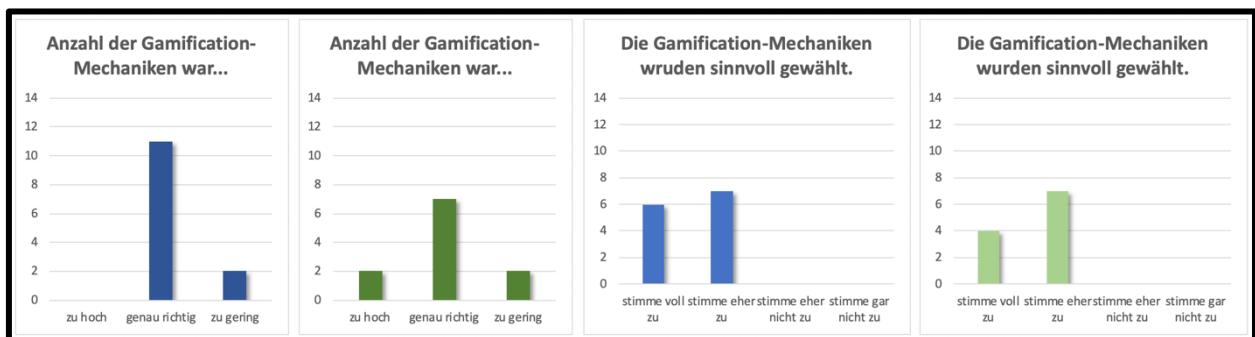


Abbildung 37: Die Ergebnisse der zweiten und dritten Fragen dargestellt in Diagrammen.

#### 6.2.4 Auswirkungen der App auf die Benutzer

Im letzten Abschnitt des Fragebogens sollten die Probanden die Auswirkungen der App auf die eigenen Gewohnheiten einschätzen. Auch hier werden die Videospieler und die nicht-Videospieler separat betrachtet.

Der ersten Aussage dieses Abschnitts „Ich kann mir vorstellen, dass ich durch die App motivierter bin, alltägliche Aufgaben zu erledigen.“ stimmten fünf (38%) der Videospieler voll zu. Sieben (54%) wählten die Antwort „Stimme eher zu“. Für die Antwort „Stimme eher nicht zu“ entschied sich ein (8%) Teilnehmer. Für die Option „Stimme gar nicht zu“ stimmte aus dieser Gruppe kein Proband. Bei den nicht-Videospielern wurde die Antwort „Stimme voll zu“ nur einmal (9%) gewählt. Genauso wie es bei den Videospielern der Fall war, stimmten auch aus dieser Gruppe insgesamt sieben (64%) Personen für „Stimme eher zu“. Für die Antwort „Stimme eher nicht zu“ entschieden sich drei (27%) Teilnehmer. Gar nicht zugestimmt hat auch hier kein Proband.

Bei der letzten im Fragebogen gestellten Frage, sollten die Befragten die Aussage „Ich kann mir vorstellen, dass sich die App positiv auf die Organisation von kleineren Gruppen auswirken kann.“ bewerten. Unter den Videospielern waren acht (62%) Teilnehmer dabei, die dieser Aussage voll zustimmen konnten. Weitere fünf (38%) wählten „Stimme eher zu“. Von den Antworten „Stimme eher nicht zu“ und „Stimme gar nicht zu“ haben die Videospieler kein Gebrauch gemacht. Bei den nicht-Videospielern entschieden sich vier (36%) für „Stimme voll zu“. Die Mehrheit (sechs, 55%) wählten in diesem Fall die Antwort „Stimme eher zu“. Ein (9%) Teilnehmer konnte der Aussage hingegen eher nicht zustimmen. Für die letzte Antwortmöglichkeit entschied sich auch aus dieser Gruppe niemand. Es folgen erneut die Diagramme, in denen die Ergebnisse für diesen Abschnitt

dargestellt werden. In den blauen Diagrammen werden wieder die Zahlen der Videospieler abgebildet und in den grünen die der nicht-Videospieler.

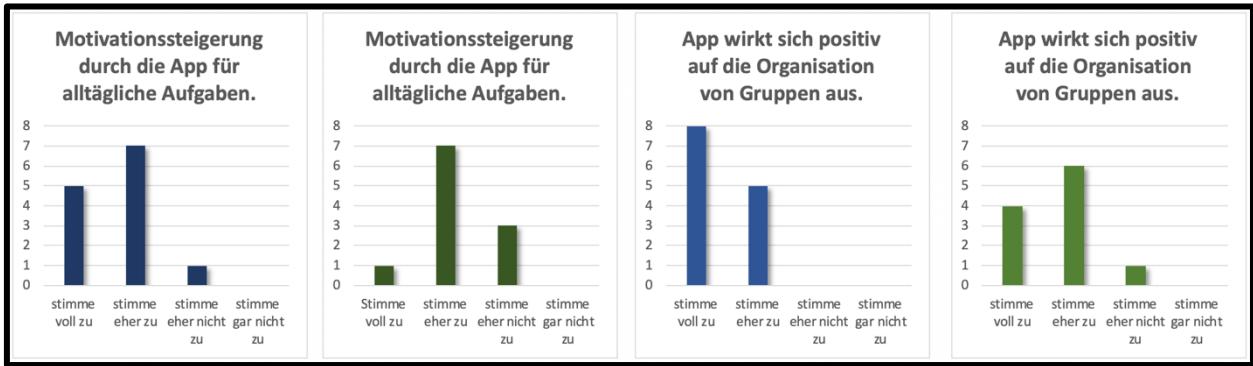


Abbildung 38: Die Ergebnisse des dritten Abschnitts visualisiert durch Diagramme.

### 6.3 Interpretation der Ergebnisse der Benutzerstudie

Nachdem im letzten Unterkapitel die Ergebnisse der Benutzerstudie präsentiert wurden, sollen diese in diesem Abschnitt nun interpretiert werden. Mit 24 Teilnehmern ist die Teilnehmeranzahl zwar eher gering, dennoch weist die Gruppe einige parallele demographischen Eigenschaften zu der Bevölkerung in ganz Deutschland auf und besitzt somit durchaus eine gewisse Aussagekraft. So waren im Jahr 2018 52% der Einwohner in Deutschland weiblich und 48% männlich [47]. Bei den Probanden ist das Verhältnis mit 12 zu zehn (55% zu 45%) fast identisch. Laut dem Verband der deutschen Games-Branche spielten im selben Jahr 42% der Deutschen Videospiele [22]. Bei der Benutzerstudie waren es mit 54% zwar ein wenig mehr, allerdings wurden bei dem Wert der deutschen Games-Branche im Gegensatz zu der Studie Spiele-Apps nicht berücksichtigt. Wäre dies der Fall, so würde der Prozentsatz deutschlandweit vermutlich höher ausfallen und sich dem Wert der Studie weiter annähern. Im Folgenden werden die durch den Fragebogen gewonnenen Daten aus den drei Bereichen Bedienbarkeit der App, Verwendete Gamification-Mechaniken und Auswirkungen der App auf die Benutzer, interpretiert.

Wie die Zahlen aus dem Abschnitt 6.2.2 (Bedienbarkeit der App) gezeigt haben, hatte der Großteil der Probanden keine Schwierigkeiten, die App zu bedienen. Das spricht dafür, dass die Intention des Entwicklers, die App für die Benutzer möglichst intuitiv und leicht bedienbar designen zu wollen, erfolgreich umgesetzt wurde. Ein wichtiger Aspekt hierfür war, bereits bekannte Abläufe aus anderen Apps zu übernehmen. Diese Tatsache bemerkten mit 85% ebenfalls fast alle Teilnehmer. Lediglich zwei Teilnehmer gaben an, sich innerhalb der App eher weniger gut zurechtgefunden zu haben. Das könnte daran liegen, dass diese Personen generell wenig Erfahrung mit dem Umgang von Smartphones und dem Bedienen von Apps haben. Unter den Probanden waren nämlich ein paar wenige dabei, die selbst gar kein Smartphone besitzen.

Durch die Kontrollfrage aus dem Abschnitt, der sich mit den verwendeten Gamification-Mechaniken innerhalb der App beschäftigt, wurde erfreulicherweise deutlich, dass im Prinzip alle Teilnehmer den Grundgedanken hinter dem Gamification-Ansatz verstanden haben, weshalb kein Fragebogen aussortiert werden musste. Die Videospieler waren zudem im Großen und Ganzen mit der Anzahl der eingesetzten Gamification-Mechaniken zufrieden. Nur zwei Teilnehmer dieser Gruppe hätten sich einen stärkeren Einsatz von Gamification gewünscht, was angesichts der bewusst gering gehaltenen Anzahl an Mechaniken schon fast überraschend ist. Der Entwickler hatte an dieser Stelle eigentlich mit mehr Videospielern gerechnet, die sich einen stärkeren Einfluss von

Gamification in der App gewünscht hätten, da diese Teilnehmer schließlich Spaß am Spielen von Videospielen haben. Allerdings empfanden sie es augenscheinlich angenehmer, in diesem Kontext nicht mit Spielmechaniken überschüttet zu werden. Bei den nicht-Videospielern ist die Meinung zu diesem Thema etwas differenzierter. Zwar ist auch bei dieser Gruppe die Mehrheit der Meinung, dass die Anzahl der eingesetzten Gamification-Mechaniken genau richtig ist, jedoch empfinden mit insgesamt 36% mehr als ein Drittel den Einfluss von Gamification in diesem Fall als zu hoch oder zu gering. Da sich diese Personen in ihrer Freizeit nicht mit Videospielen beschäftigen, ist es nicht verwunderlich, dass sie den Gamification-Mechaniken anscheinend nicht viel abgewinnen können und deshalb die wenigen eingesetzten Mechaniken dennoch als störend empfinden. Zusammenfassend lässt sich jedoch festhalten, dass der in Kapitel 4.2 formulierte Ansatz, die App mit Gamification-Inhalten zu versehen, aber nicht mit diesen zu überfluten, bei den meisten Teilnehmern auf Zustimmung stößt. Die Tatsache, dass alle Teilnehmer die verwendeten Gamification-Mechaniken als sinnvoll gewählt empfanden, bestätigt zudem die im Vorfeld gemachten Gedanken bezüglich der eingesetzten Mechaniken (auch Kapitel 4.2).

Wie in dem vorherigen Unterkapitel bereits beschrieben, beschäftigte sich der letzte Abschnitt des Fragebogens mit den Auswirkungen der App auf die Probanden. Auch hier unterscheiden sich die Aussagen zwischen den Videospielern und den nicht-Videospielern ein wenig. Während sich die Videospieler fast alle vorstellen konnten, dass die App sowohl auf ihre eigenen Gewohnheiten als auch auf die Organisation von Gruppen positive Auswirkungen hat, waren sich bei den nicht-Videospielern fast ein Drittel unsicher, ob sie durch die App motivierter sind, alltägliche Aufgaben schneller und besser zu erledigen. Anhand dieser Ergebnisse ist zu sehen, dass gerade Personen, die zumindest gelegentlich Videospiele spielen, leicht für Gamification zu begeistern sind und sich durch diese folglich leichter beeinflussen lassen. Auch der Großteil der Menschen, die sonst gar keine Videospiele spielen, sind offen für den Gamification-Ansatz. Jedoch ist bei ihnen der Anteil derer, die sich für Gamification nicht begeistern können, deutlich höher als bei den Videospielern.

Zu der einzigen offenen Frage in dem Fragebogen wurde 13-mal etwas geschrieben. Neben einem Lob für den Entwickler wurden hier vor allem Anregungen für weitere Inhalte gegeben, von denen einige in Zukunft umgesetzt werden sollen. Auf diese Anregungen wird in Kapitel 7 genauer eingegangen, wenn es darum geht, wie es mit der App in Zukunft weitergehen wird. Alle Antworten auf diese Frage werden zudem in dem Anhang aufgelistet.

Insgesamt fällt das durch die Benutzerstudie erhaltene Feedback fast durchweg positiv aus. Der überwiegende Teil der Befragten fand sich einerseits schnell innerhalb der App zurecht und empfand andererseits die ausgewählten Gamification-Mechaniken als sinnvoll und motivierend eingesetzt. Aufgrund dieser zwei Aspekte können sich fast alle Teilnehmer vorstellen, dass sich die App positiv auf ihr Verhalten auswirken könnte.

Durch dieses Kapitel sollte die durchgeführte Evaluierung für die hier erstellte App dem Leser präsentiert werden. So wurde am Anfang des Kapitels der Aufbau und die Durchführung der Benutzerstudie beschrieben. In diesem Abschnitt wurde auch auf die für die Studie benötigte App eingegangen. Anschließend stellte der Entwickler die durch die Studie gewonnenen Ergebnisse vor und interpretierte sie im letzten Abschnitt dieses Kapitels.

## **7. Zusammenfassung und Ausblick**

Vielen Menschen fällt es schwer, sich für alltägliche Aufgaben zu motivieren, wie etwa das Bad zu putzen oder die Küche aufzuräumen. Bei Gruppen, in denen beispielsweise eine Wohngemeinschaft oder ein Verein organisiert werden muss, kommt erschwerend hinzu, dass die anfallenden Aufgaben häufig unstrukturiert bearbeitet werden. Das hat zur Folge, dass oft nur die beliebten Aufgaben erledigt werden und die unbeliebten, aber dennoch notwendigen unaufgetastet bleiben, frei nach dem Motto: „Wieso soll ich mich um die unbeliebten Aufgaben kümmern? Es gibt doch genug andere Menschen in der Gruppe, die das machen können.“

Die vorliegende Masterthesis befasste sich mit genau dieser Problemstellung und lieferte eine mögliche Lösung in Form einer App. Durch die im Rahmen dieser Arbeit entwickelten App, sollen Gruppen einerseits besser organisiert werden können und andererseits Anreize für die Mitglieder geschaffen werden, unbeliebte oder monotone Aufgaben zu bearbeiten. Um diesem Anspruch gerecht zu werden, wurde sich der motivationssteigernden Methode Gamification bedient. Durch sie sollen die Gruppenmitglieder motiviert werden, sich freiwillig für Aufgaben einzutragen und diese im Anschluss zu erledigen. Die dafür erstellte App sieht vor, dass die Benutzer neue Gruppen anlegen und ihre Freunde und Bekannte in diese einladen können. Anschließend kann jedes Gruppenmitglied Aufgaben, welche in diesem Fall als „Todos“ bezeichnet werden, erstellen. Diese Todos können daraufhin von einem beliebigen Mitglied bearbeitet werden. Damit die Benutzer motiviert sind, Todos anzunehmen, wurden in die App einige Gamification-Mechaniken integriert. Neben den gerade erwähnten Todos werden noch ein Punktesystem, eine Rangliste und Auszeichnungen eingesetzt. Durch diese Mechaniken können sich die Benutzer Punkte verdienen, indem sie Todos abschließen und erhalten daraufhin eine vorherfestgelegte Anzahl an Punkten. Des Weiteren werden alle Gruppenmitglieder anhand ihrer bereits verdienten Punkte in einer Rangliste aufgelistet. So kann jeder Benutzer innerhalb der Gruppe direkt sehen, wer sich wie viel eingebracht hat.

Die Implementierung der App fand mittels der Entwicklungsumgebung Xcode und der Programmiersprache Swift statt. Beide Tools werden von Apple selbst bereitgestellt, weshalb die App nur auf Geräten läuft, die das Betriebssystem iOS verwenden. Durch die erstellte App werden viele Daten generiert, die an einem zentralen Ort verwaltet werden müssen. Die zentrale Datenverwaltung ist in diesem Fall notwendig, da die Benutzer untereinander interagieren sollen und es deshalb nicht ausreicht, wenn jeder Benutzer nur auf seine eigenen Daten zugreifen kann. Nachdem verschiedene online-Datenbanksysteme betrachtet wurden, wurde sich für das Framework CloudKit entschieden, das ebenfalls von Apple bereitgestellt wird. Aus diesem Grund war die Integration von CloudKit in Xcode und Swift relativ unkompliziert. Aber natürlich lief nicht der gesamte Entwicklungsprozess so reibungslos ab. So wurde der Entwickler immer wieder mit zum Teil unerwarteten Herausforderungen konfrontiert. Gerade die Speicherverwaltung und die damit einhergehenden Transferzeiten bereiteten Probleme, für die Lösungen gefunden werden mussten. Darüber hinaus veröffentlichte Apple während der Entwicklung das Update auf iOS 13, das einige Neuerungen lieferte, auf die der Entwickler ebenfalls reagieren musste, wie etwa den Dark Mode.

Nachdem eine erste Version der App zum Testen bereitstand, führte der Entwickler eine Evaluation mit einigen Teilnehmern durch, um Feedback für seine App zu erhalten. Die Evaluation in einem größeren Rahmen durchzuführen, war leider nicht möglich, da die App zu diesem Zeitpunkt nur auf ein paar ausgewählten Geräten lief und der Entwickler deshalb bei jeder Befragung selbst vor Ort sein musste. Für die Benutzerstudie wurde eigens eine weitere App programmiert, die die Probanden durch die gesamte Studie führen sollte und diese abschließend bezüglich der eigentlichen App befragte. Die Studie ergab zum einen, dass der Großteil der Befragten sich gut innerhalb der App zuretfand und zum anderen, dass die eingesetzten Gamification-Mechaniken als motivationssteigernd bewertet wurden, weshalb die in dieser Masterthesis eingangs gestellte Frage, ob die

erstellte App positive Auswirkungen auf die Benutzer hat, überwiegend mit „ja“ beantwortet werden.

Auch wenn der gesamte Entwicklungsprozess der App im Wesentlichen ohne größere Probleme verlief, gab es dennoch neben den bereits genannten Herausforderungen, die während der Implementierung auftraten, einige Punkte, die kritisch zu betrachten sind. Viele Tools, die für die Erstellung einer App mit zentraler Datenverwaltung benötigt werden, sind mit zum Teil nicht unerheblichen Kosten verbunden. So wird beispielsweise für das Hochladen einer App in Apples App-Store ein Developer-Account vorausgesetzt, welcher 100 Euro im Jahr kostet. Ebenso sind viele cloud-basierte Lösungen zum Verwalten von Datenbanken nur auf dem ersten Blick kostenlos. Häufig ist es für den Benutzer nicht genau ersichtlich, wann welche Kosten anfallen, sodass dieser unerwartet zur Kasse gebeten wird. Das kann schnell sehr teuer werden! Apple zum Beispiel verlangt 100 Euro pro zehn weitere Anfragen pro Sekunde, sobald ein bestimmter Schwellwert überschritten wird. Des Weiteren sollte nicht unterschätzt werden, was es heißt, eine App komplett alleine zu programmieren. Neben der Kernkompetenz des Entwicklers, dem Schreiben von Code, muss sich dieser auch um alle anderen anfallenden Aufgaben kümmern, etwa dem Designen des Interfaces, dem Erstellen von Grafiken oder dem Anordnen von Abläufen innerhalb der App. Auch bei der Fehler-suche kann es sehr zum Nachteil sein, wenn es darum geht, seine eigenen Fehler zu finden. Häufig fallen anderen Personen die vorhandenen Fehler schneller auf, wodurch viel Zeit gespart werden kann.

Bekanntlich ist die Entwicklung von Software nie wirklich abgeschlossen. Es gibt immer Punkte, die noch hinzugefügt oder verbessert werden können, wie auch in diesem Fall. Durch die Benutzerstudie hat der Entwickler viele neue Ideen erhalten, von denen einige in Zukunft umgesetzt werden sollen. So haben sich beispielsweise viele Teilnehmer gewünscht, dass die Punktevergabe für Todos überarbeitet werden soll, da sie die aktuelle Lösung als unfair empfinden. Zudem soll die App noch um neue Features erweitert werden, die einerseits das Zusammenspielen fördern, etwa durch Gruppenherausforderungen. Andererseits sollen jedoch auch Mechaniken implementiert werden, die den Wettkampf unter den Benutzern weiter verschärft. Durch diese Erweiterungen soll das Spielerlebnis für die Benutzer immer weiter verbessert werden.

Vor drei Jahren schrieb der Autor dieser Arbeit in seiner Bachelorthesis, dass sich Gamification noch in den Kinderschuhen befände. Heute, im Jahr 2020, unterstreichen immer mehr Zahlen und Studien, dass die Gamification ihren Kinderschuhen entwachsen ist. Zwar ist ihre Entwicklung wohl noch lange nicht abgeschlossen, dennoch sind Videospiele und somit auch die Gamification schon heute in der Mitte unserer Gesellschaft angekommen, weshalb die positiven Auswirkungen von Gamification in Zukunft vermutlich in immer mehr Bereichen ausgenutzt werden.

## Abbildungsverzeichnis

Abbildung 1: Apples Entwicklungsumgebung Xcode ist unterteilt in drei Bereiche: Links: Der Package Explorer. Mitte: Das Main Storyboard. Rechts: Der Quellcode. ....	7
Abbildung 2: Zeigt eine Rangliste in der App Runtastic.....	10
Abbildung 3: Links: Pokémon-Eier, die ausgebrütet werden. Mitte: ein Kampf gegen ein Pokémon. Rechts: Informationen zu einem ausgewählten Pokémon. ....	11
Abbildung 4: Charakteransicht eines Avatars in der App Habitica.....	12
Abbildung 5: Anlegen von einer Konstanten und zwei Variablen in Swift. ....	14
Abbildung 6: Links: Ein Label wurde als IBOutlet angelegt und ein String wurde diesem zugeordnet. Rechts: Das Label zu sehen in der App.....	15
Abbildung 7:Ein Button mit der dazugehörigen Methode. ....	15
Abbildung 8: Deklarieren eines Arrays und Initialisieren einer Booleschen-Variablen.....	16
Abbildung 9: Anlegen und Befüllen eines Any-Array in Swift.....	17
Abbildung 10: Anlegen von optional und implicitly unwrapped optional Variablen in Swift. ....	18
Abbildung 11: Eine If-Else Anweisung in Swift. Übrigens: Im Gegensatz zu anderen Sprachen ist es in Swift möglich, zwei Strings mit Hilfe des ==-Operators zu vergleichen.....	18
Abbildung 12: Signatur der Klasse HomeViewController, welche von einigen Protokollen erbt.....	19
Abbildung 13: Links: Die einzelnen Elemente einer Szene hierarchisch dargestellt. Rechts: Eine fertige Szene.....	20
Abbildung 14: Links: Der Quellcode um die Tabelle zu formatieren. Rechts: Die fertige Tabelle....	21
Abbildung 15: Eine Funktion mit zwei Parametern und einem Rückgabewert in Swift. ....	22
Abbildung 16: Die fetch-Funktion aus dem CloudKit-Framework. ....	23
Abbildung 17: Die save-Funktion aus dem CloudKit-Framework. ....	23
Abbildung 18: Die update-Funktion aus dem CloudKit.....	24
Abbildung 19: Klasse B verwendet die Methode aus Klasse A. ....	24
Abbildung 20: Links: Ein erster Entwurf des Home-Screens der App. Rechts: Weitere Entwürfe der App mit mehr farbigen Elementen (Home-Screen und Todo-Screen). ....	31
Abbildung 21: Zwei Szenen der finalen Version der App. ....	31
Abbildung 22: Die Todo-Szene aus einer recht frühen Iteration der App.....	32
Abbildung 23: Links: Der Todo-Screen mit dem Plus-Symbol, das den Benutzer zum Erstellungsformular weiterleitet. Mitte: Das Erstellungsformular, um ein Todo zu erstellen. Rechts: Der Home-Screen mit einer Ladeanimation.....	34
Abbildung 24: Die Ansicht des Auszeichnung-Tabs der App. ....	35
Abbildung 25: Schematischer Aufbau einer Speicherverwaltung mittels einer MySQL-Datenbank und PHP-Skripten. Quelle: erstellt von Vanessa Schmall.....	40
Abbildung 26: Das iCloud-Webinterface, in dem neue Datentypen und Attribute erstellt werden können. ....	42
Abbildung 27: Dieses Schaubild skizziert den Ablauf der Todos-Erstellung. Quelle: erstellt von Vanessa Schmall. ....	44
Abbildung 28: Erstellungsdialog eines neuen Projektes in Xcode.....	47
Abbildung 29: Das Storyboard der Beispiel-App unmittelbar nach der Projekterstellung.....	48
Abbildung 30: Das Storyboard der Beispiel-App, nachdem die Screens mit Elementen befüllt wurden. ....	49
Abbildung 31: Verbinden eines Elementes mit der dazugehörigen Swift-Klasse. ....	49
Abbildung 32: Die beiden Screens der fertig implementierten Beispiel-App. ....	52
Abbildung 33: Links: die Einleitung in die Studie (erster Abschnitt der Studie). Mitte: ein Screen des zweiten Abschnitts. Rechts: ein Teil des Fragebogens (letzter Abschnitt). ....	55

Abbildung 34: Die Ergebnisse des ersten Abschnitts, dargestellt in Diagrammen.....	58
Abbildung 35: Die Ergebnisse des zweiten Abschnitts des Fragebogens visualisiert durch Diagramme.....	59
Abbildung 36: Die Ergebnisse der ersten Frage des dritten Abschnitts dargestellt in Diagrammen.	60
Abbildung 37: Die Ergebnisse der zweiten und dritten Fragen dargestellt in Diagrammen.....	60
Abbildung 38: Die Ergebnisse des dritten Abschnitts visualisiert durch Diagramme. ....	61

## Literatur

- [1] Sraddhanjali Acharya, Isaac Griswold-Steiner , Richard Matovu, Abdul Serwadda: Gamification of Wearable Data Collection: A Tool for both Friend and Foe. ICCDA '19: Proceedings of the 2019 3<sup>rd</sup> International Conference on Compute and Data Analysis; März 2019; S. 68-77.
- [2] Amazon Web Services, Inc.: AWS. [https://aws.amazon.com/de/?nc2=h\\_lg](https://aws.amazon.com/de/?nc2=h_lg), letzter Zugriff am 12.02.2020.
- [3] Apple Inc.: CloudKit. <https://developer.apple.com/icloud/cloudkit/>, letzter Zugriff am 12.02.2020.
- [4] Apple Inc.: Human Interface Guidelines – Tab Bars. <https://developer.apple.com/design/human-interface-guidelines/ios/bars/tab-bars/>, letzter Zugriff am 12.02.2020.
- [5] Apple Inc.: Programming with Objective-C. <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html>, letzter Zugriff am 12.02.2020.
- [6] Apple Inc.: Xcode 11. <https://developer.apple.com/xcode/>, letzter Zugriff am 12.02.2020.
- [7] Mag. Dr. Birgit Aschenmann-Pilshofer: Wie erstelle ich einen Fragebogen? – Ein Leitfaden für die Praxis. Wissenschaftsladen Graz Institut für Wissens- und Forschungsvermittlung; 2001.
- [8] Katerina Avramides, Sara De Freitas, Kam Memarzia, Genaro Rebolledo-Mendez: Societal impact of a Serious Game on raising public awareness: the case of FloodSim. Sandbox '09: Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games; August 2009; S. 15-22.
- [9] Christian Beyerlein, Martin Johns: SMask: Preventing Injection Attacks in Web Applications by Approximating Automatic Data/Code Separation. SAC '07: Proceedings of the 2007 ACM symposium on Applied computing; März 2007; S. 284-291.
- [10] Christian Bleske: iOS-Apps programmieren mit Swift – Der leichte Einstieg in die Entwicklung für iPhone, iPad und Co. – inkl. Apple Watch. dpunkt.verlag; Heidelberg; 2016. ISBN 978-3-86490-263-5.
- [11] Yixin Chen, Chad Vicknair, Dawn Wilkins: MySQL and the Trouble with temporal Data. ACM-SE '12: Proceedings of the 50<sup>th</sup> Annual Southeast Regional Conference; März 2012; S. 176-181.
- [12] Nikolaj Cholakov: On some drawbacks of the PHP platform. ComSysTech '08: Proceedings of the 9<sup>th</sup> International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing; Juni 2008.
- [13] Hung Dang Phan, Trong Duc Nguyen, Anh Tuan Nguyen, Tien N. Nguyen: Exploring API Embedding for API Usages and Applications. ICSE '17: Proceedings of the 39<sup>th</sup> International Conference on Software Engineering; Mai 2017; S. 438-449.
- [14] Natalie Denmeade: Gamification with Moodle – Use game elements in Moodle courses to build learner resilience and motivation. Packet Publishing, Birmingham; Oktober 2015. ISBN 978-1-78217-307-6.
- [15] Sebastian Deterding, Dan Dixon, Rilla Khaled, Lennart Nacke: From Game Design Elements to Gamefulness: Defining “Gamification”. MindTrek '11: Proceedings of the 15<sup>th</sup> International Academic MindTrek Conference: Envisioning Future Media Environments; September 2011; S. 9-15.
- [16] DFL Deutsche Fußball Liga GmbH: DFL Report 2018: Profifußball erstmals mit mehr als vier Milliarden Euro Umsatz – 14 Bundesliga-Clubs über 100-Millionen-Euro-Grenze. Frankfurt am Main, 2018.
- [17] Anke Dittmar, Eija Kaasinen, Marja Liinasuo: A Study on Qualities of Service Experience and Implications for User Experience Research. ECCE '16 Proceedings of the European Conference on Cognitive Ergonomics; September 2016; Artikel 6.

- [18] Christine Duller: Einführung in die Statistik mit EXCEL und SPSS – Ein anwendungsorientiertes Lehr- und Arbeitsbuch. Springer Gabler; Linz, 2019. ISBN 978-3-662-59409-4.
- [19] Joao Duraes, Pedro Neves, Nuno Paiva: A Comparison between JAVA and PHP. C3S2E '13: Proceedings of the International C\* Conference on Computer Science and Software Engineering; Juli 2013; S. 130-131.
- [20] eSports Earnings: Larges overall prize pools in eSport. <https://www.esportsearnings.com/tournaments>, letzter Zugriff am 12.09.2019.
- [21] Flatastic: flatastic – Und der Haushalt? Läuft. <https://flatastic-app.com>, letzter Zugriff am 12.02.2020.
- [22] game – Verband der deutschen Games-Branche e.V.: Deutscher Games-Markt 2018. <https://www.game.de/marktdaten/deutscher-games-markt-2018/>, letzter Zugriff am 12.02.2020.
- [23] Annelie Götze: Die Wirkung von Gamification auf Motivation und Leistung. Hochschule für angewandte Wissenschaften Augsburg. [http://www.hs-augsburg.de/~john/Currents/Material/2016/Book\\_Gamification/Gamification\\_Kapitel3\\_AnnelieGoetze.pdf](http://www.hs-augsburg.de/~john/Currents/Material/2016/Book_Gamification/Gamification_Kapitel3_AnnelieGoetze.pdf), letzter Zugriff am 12.02.2020.
- [24] HabitRPG, Inc.: Habitica – Steige ein Level auf wo immer Du bist. <https://habitica.com/static/home>, letzter Zugriff am 12.02.2020.
- [25] Pedro M. Latorre Andrés, Jorge López Moreno, Francisco J. Serón Arbeloa, Carlos Vaz de Carvalho: TimeMesh: Producing and Evaluating a Serious Game. Interacción '14: Proceedings of the XV International Conference on Human Computer Interaction Article No. 100; September 2010.
- [26] Chris Lattner: Chris Lattner's Homepage: Swift. <http://nondot.org/sabre/>, letzter Zugriff am 12.02.2020.
- [27] Hu Liang, Duo Shuwang: Design and Realization of Material Chemical Laboratory Information Management System. BDET '18: Proceedings of the 2018 International Conference on Big Data Engineering and Technology; August 2018; S. 92-95.
- [28] André Linken: Pokémon Go: Mobile-Spiel knackt eine Milliarde Downloads. <https://www.pcgames.de/Pokemon-GO-Spiel-56108/News/mobile-spiel-knackt-eine-milliarde-downloads-1296006/>, letzter Zugriff am 12.02.2020.
- [29] Looploop IVS: Tody. <http://www.todyapp.com>, letzter Zugriff am 12.02.2020.
- [30] Natalja Menold, Cornelia Züll: Handbuch Methoden der empirischen Sozialforschung. Springer VS; 2014. ISBN 978-3-531-18939-0.
- [31] MySQL AB: MySQL. <https://www.mysql.com>, letzter Zugriff am 12.02.2020.
- [32] Neeritech GbR: Haus Segen Die WG Planer APP. <https://haus-segen.com>, letzter Zugriff am 12.02.2020.
- [33] Ninatic, Inc.: Pokémon Go. <https://www.pokemongo.com/de-de/>, letzter Zugriff am 12.02.2020.
- [34] Suwichai Phunsa, Suwich Tirakoat: A case study of developing game edutainment: “addictive danger”. DIMEA '08: 3<sup>rd</sup> international conference on Digital Interactive Media in Entertainment and Arts; September 2008; S. 58-61.
- [35] Catherine Plaisant, Ben Shneiderman: Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison-Wesley; 2009. ISBN: 978-0-321-53735-5.
- [36] Benedikt Plass-Fleßenkämpfer , Sebastian Weber: Gamification: Das ganze Leben ist ein Spiel. Medienagentur Plassma. <http://www.plassma.de/972/>, letzter Zugriff am 12.02.2020.
- [37] Pokémon Go Team: 200.000 Reisen um die Welt!. <https://pokemongo.nianticlabs.com/de/post/milestones/>, letzter Zugriff am 12.02.2020.
- [38] Rolf Porst: Fragebogen – Ein Arbeitsbuch. Springer VS; 2014. ISBN 978-3-658-02117-7.

- 
- [39] Ben R. Rich: Clarence Leonard (Kelly) Johnson 1910-1990: A Biographical Memoir. National Academies Press; 1995.
- [40] Runtastic GmbH: Facts & Figures. <https://www.runtastic.com/career/facts-about-runtastic/>, letzter Zugriff am 12.02.2020.
- [41] Runtastic GmbH: Runtastic. <https://www.runtastic.com>, letzter Zugriff am 12.02.2020.
- [42] Statista GmbH: Anzahl der Computerspieler in Deutschland von 2013 bis 2019 (in Millionen). <https://de.statista.com/statistik/daten/studie/712928/umfrage/anzahl-der-computerspieler-in-deutschland/>, letzter Zugriff 12.02.2020.
- [43] Statista GmbH: Prognose zur Anzahl der eSports-Zuschauer weltweit in den Jahren 2016 bis 2022 (in Millionen). <https://de.statista.com/statistik/daten/studie/586871/umfrage/prognose-zur-anzahl-der-esports-zuschauer-weltweit/>, letzter Zugriff am 12.02.2020.
- [44] Statista GmbH: Umsatz im eSport-Markt weltweit in den Jahren 2015 bis 2018 und Prognose für 2019 und 2022 (in Millionen US-Dollar). <https://de.statista.com/statistik/daten/studie/677986/umfrage/prognose-zum-umsatz-im-esports-markt-weltweit/>, letzter Zugriff am 12.02.2020.
- [45] Statista GmbH: Verteilung der Nutzer von Gaming-Apps nach Altersgruppen in Deutschland im Jahr 2017. <https://de.statista.com/statistik/daten/studie/878340/umfrage/nutzung-von-gaming-apps-nach-altersgruppen-in-deutschland/>, letzter Zugriff am 12.02.2020.
- [46] Statista GmbH: Verteilung der Nutzer von Gaming-Apps nach Geschlecht in Deutschland im Jahr 2017. <https://de.statista.com/statistik/daten/studie/878329/umfrage/nutzung-von-gaming-apps-nach-geschlecht-in-deutschland/>, letzter Zugriff am 12.02.2020.
- [47] Statistisches Bundesamt: Bevölkerung – Einwohnerzahl in Deutschland nach Geschlecht von 1995 bis 2018 (in 1.000). <https://de.statista.com/statistik/daten/studie/161868/umfrage/entwicklung-der-gesamtbevoelkerung-nach-geschlecht-seit-1995/>, letzter Zugriff 12.02.2020.
- [48] The PHP Group: PHP. <https://www.php.net/docs.php>, letzter Zugriff am 12.02.2020.
- [49] Tiny Giant UG: Hubble: Die App für euren Haushalt. <http://hellohubble.com>, letzter Zugriff am 12.02.2020.
- [50] Woohoo Software: Clean My House. <http://www.woohoosoftware.com/clean-my-house/>, letzter Zugriff am 12.02.2020.
- [51] 6 Wunderkinder GmbH: Wunderlist. <https://www.wunderlist.com>, letzter Zugriff am 12.02.2020.

## Anhang

### A.1 Fragebogen

2:17 PM Tue Feb 11 100%

### Benutzerstudie - Fragebogen (1/5)

Spielst du gelegentlich Videospiele? Dazu gehören auch Spiele-Apps.

Ja      Nein

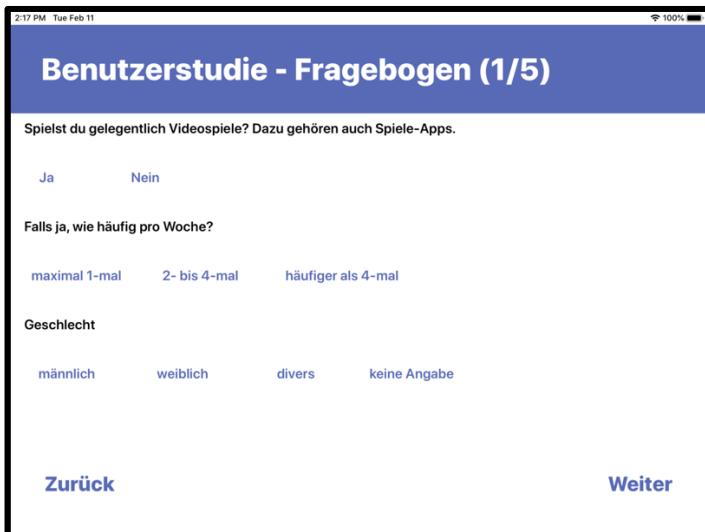
Falls ja, wie häufig pro Woche?

maximal 1-mal    2- bis 4-mal    häufiger als 4-mal

Geschlecht

männlich    weiblich    divers    keine Angabe

**Zurück**      **Weiter**



2:18 PM Tue Feb 11 100%

### Benutzerstudie - Fragebogen (2/5)

Die App war einfach zu bedienen.

Stimme voll zu    Stimme eher zu    Stimme eher nicht zu    Stimme gar nicht zu

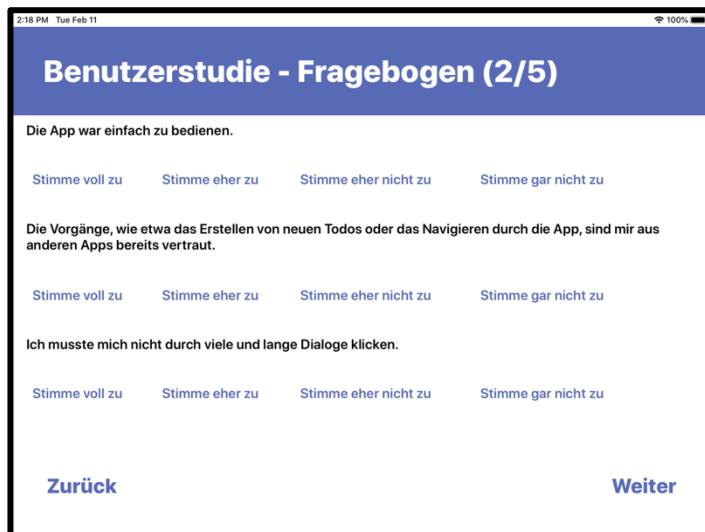
Die Vorgänge, wie etwa das Erstellen von neuen Todos oder das Navigieren durch die App, sind mir aus anderen Apps bereits vertraut.

Stimme voll zu    Stimme eher zu    Stimme eher nicht zu    Stimme gar nicht zu

Ich musste mich nicht durch viele und lange Dialoge klicken.

Stimme voll zu    Stimme eher zu    Stimme eher nicht zu    Stimme gar nicht zu

**Zurück**      **Weiter**



2:19 PM Tue Feb 11 100%

### Benutzerstudie - Fragebogen (3/5)

Welche Gamification-Mechaniken werden in der App eingesetzt?

Rangliste    Zeitdruck    Auszeichnungen    Kampfsystem

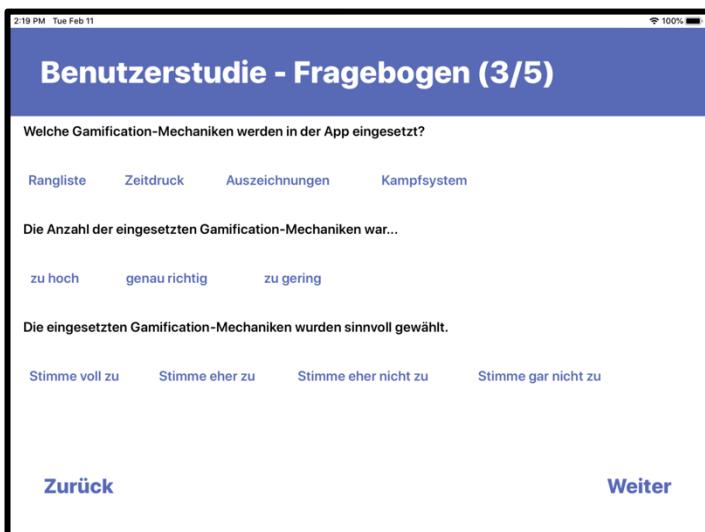
Die Anzahl der eingesetzten Gamification-Mechaniken war...

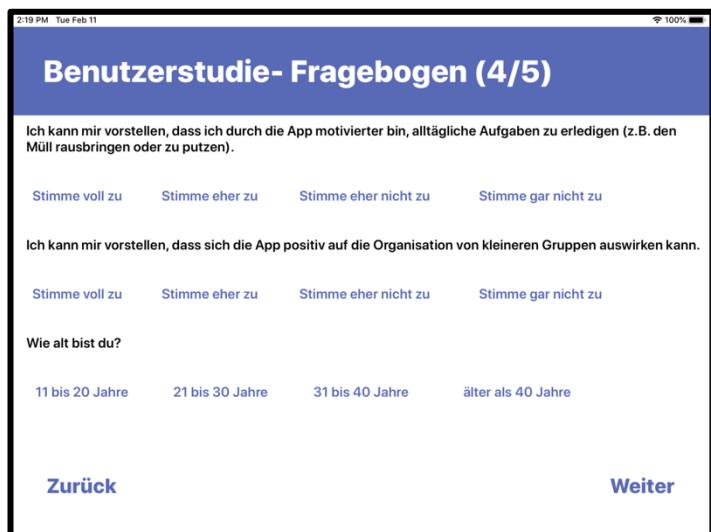
zu hoch    genau richtig    zu gering

Die eingesetzten Gamification-Mechaniken wurden sinnvoll gewählt.

Stimme voll zu    Stimme eher zu    Stimme eher nicht zu    Stimme gar nicht zu

**Zurück**      **Weiter**





## A.2 Antworten auf die offene Frage des Fragebogens (im Wortlaut)

„Ausloggen Button unten ist ungewohnt (deswegen nicht unbedingt schlecht-nur vielleicht gewöhnungsbedürftig)

„Bestätigungen“ sind grundsätzlich nicht schlecht aber auf Dauer vielleicht zu viel, wenn man mit der App vertraut ist. Kann man das ggf. abschalten?“

„Bezeichnungen für gewisse Level von Punkten mit Krone oder ähnliches die das Level auch im Benutzerprofil für andere bildlich ausweisen, mit steigendem Level glorreichere Titel/Symbole“

„Echte photos, punktanzahl von bis angeben, oder vglswerte: klo putzten bringt in den meisten Weg 10 Punkte“

„Es wäre hilfreich zu wissen, in welchen Zahlenbereich die Punkte liegen sollten 1-10/1-100, oder einen Mechanismus einzubauen, durch den über die angemessene Punktzahl abgestimmt wird, sonst ist es einfach (siehe Achim), sich auf der Rangliste nach oben zu schummeln.“

Mir wurde noch nicht klar, wie ich eine Todo, die von anderen erstellt wurde annehmen kann. Aber das mag an mir liegen.

Coole Idee!“

„Die Rote Zahl bei den Auszeichnungen sollte dann auch wieder verschwinden, wenn man draufklickt. Hilft aber schon um Aufmerksamkeit zu erzeugen. :)“

„Kooperative Elemente einbauen - Teamarbeit stärken?“

„Gute Idee, weiter so!“

„Kann man erkennen welche Todos innerhalb einer Gruppe zwar von Nutzern angenommen aber noch nicht erledigt sind? Wenn ja besteht die Möglichkeit diese Todos zu „klauen“? Das würde eventuell den Wettbewerb noch erhöhen.“

„In der Rangliste wäre es eventuell interessant mehr Informationen über die anderen Teilnehmer zu erfahren. Z.B was er für Aufgaben erledigt hat, welche die war mit der höchsten Punktzahl, wie lange er schon Mitglied ist usw.

In der Ansicht der Ranglisten ist mir nicht direkt klar was die Zahlen vor den Lorbeerkränzen bedeutet. Ich nehme an wieviel Todos die Person bereits erledigt hat. Vielleicht wäre da das Bild Jobs done eindeutiger.

Punktevergabe für ein Todo ohne Relation oder Vergleichsmaßstab kann zu „unfairen“ Verhältnissen führen.“

„Den Wettbewerb steigern indem öfter auf den Punktestand der anderen Gruppenmitglieder und den eigenen Rang hingewiesen wird, z.B. beim drücken von „erledigt“ bei einem Todo um zu motivieren gleich das nächste Todo zu greifen.“