



Teachify - Lernspiele für Kinder

Dokumentation, Spezifikation, Konstruktion

Angelina Scheler, Bastian Kusserow, Christian Pfeiffer,
Christian Pöhlmann, Johannes Franz, Marcel Hagmann, Maximilian Sonntag,
Normen Krug, Patrick Niepel, Philipp Dümlein, Carl Philipp Knoblauch

10.07.2018
Vorgelegt bei Prof. Dr. Sven Rill

Inhaltsverzeichnis

1 Pflichtenheft	1
1.1 Zielbestimmung	1
1.1.1 Musskriterien	1
1.1.1.1 Allgemein	1
1.1.1.2 Login	1
1.1.1.3 Schüler	1
1.1.1.4 Lehrer	2
1.1.1.5 Schnittstelle	2
1.1.2 Wunschkriterien	2
1.1.2.1 Allgemein	2
1.1.2.2 Login	2
1.1.2.3 Schüler	2
1.1.2.4 Lehrer	2
1.1.2.5 Schnittstelle	3
1.1.3 Abgrenzungskriterien	3
1.2 Produkteinsetz	3
1.2.1 Anwendungsbereiche	3
1.2.2 Zielgruppen	3
1.2.3 Betriebsbedingungen	3
1.3 Produktfunktionen	3
1.3.1 /F0010/ Einloggen	3
1.3.2 /F0020/ Verfügbare Spiele herunterladen (Schüler/Schnittstelle)	4
1.3.3 /F0030/ Spielinformationen anzeigen (Schüler)	4
1.3.4 /F0040/ Spiel spielen (Schüler)	4
1.3.5 /F0050/ Leistungen anzeigen (Schüler)	4
1.3.6 /F0060/ Schüler in Klassen einladen (Lehrer/Schnittstelle/Schüler)	4
1.3.7 /F0070/ Aufgaben Klassen zuteilen	4
1.3.8 /F0080/ Ergebnisse einzelner Schüler anzeigen	4
2 Schülerteam: MathePiano Spiel, Login & Schülerhauptmenü	5
2.1 Einleitung	5
2.1.1 Motivation	5
2.1.2 Ziele	5
2.2 Spezifikation	5
2.2.1 Schülerhauptmenü	5
2.2.1.1 Einleitung	5
2.2.1.2 Ziele	6
2.2.1.3 Einfache Loginmöglichkeiten	6
2.2.1.4 Simple Navigationsmöglichkeiten zu den Spielen	6
2.2.1.5 Unterscheidung zwischen Aufgaben und Endlosspielen	6

2.2.1.6	Darstellung von Hintergrundaktivitäten	6
2.2.1.7	User Interface	7
2.2.2	Mathe Piano Spiel	9
2.2.2.1	Game Engine	9
2.2.2.2	Herausforderungen	9
2.2.2.3	Testen des Spieles	9
2.2.2.4	Anbindungen an interne Schnittstellen	9
2.2.2.5	User Interface	9
2.3	Implementierungsphase	10
2.3.1	Mathe Piano Spiel	10
2.3.1.1	SpriteKit	10
2.3.1.2	Game Models	10
2.3.2	User Interface	11
2.4	Fazit	11
3	Schüler Game Teachify Bird	14
3.1	Konzept und Spielidee	14
3.2	Technische Umsetzung	15
3.3	Aufgaben	15
3.4	Animieren des Backgrounds	16
3.5	Navigieren des Vogels durch Neigen des Gerätes	16
3.6	Punkte und Leben	16
3.7	Animieren des Vogels	16
3.8	Quellenangaben	16
4	Das Spiel FeedMe	17
4.1	Der Entwurf	17
4.2	Die Umsetzung und Implementierung	17
4.2.1	Die Nodes	19
4.2.2	Der Sound	19
4.2.3	Die Touch-Events	19
4.2.4	Die Animation	20
4.2.5	Das Game Play	21
5	Lehrer-Seite	22
5.1	Lehrer UI	22
5.1.1	Homescreen	22
5.1.2	Erstellen von Aufgaben	22
5.2	Implementierung	23
5.2.1	Homescreen	23
5.2.1.1	Klassen- und Fächer-CVs	23
5.2.1.2	Dokumente-CV	23
5.2.2	Herunterladen der Clouddaten	23
5.2.2.1	Problematik	24
5.2.2.2	Lösung	24
5.3	Aufgaben erstellen	24
5.4	Fazit	25

6	Gemeinsames Datenmodell zur Speicherung der Schnittstellendaten	26
6.1	Ziele	26
6.2	Implementierung	26
6.2.1	TKModelSingleton	26
6.2.2	TKFetchController	27
6.2.2.1	Abrufen aus der Schnittstelle mit Operations	27
7	Schnittstelle iCloud	30
7.1	Einleitung	30
7.2	Warum iCloud/CloudKit?	30
7.3	Architektur	30
7.4	Features	32
7.4.1	Upload/Download/Delete/Fetch/Update	32
7.4.2	User Profile	32
7.4.3	Sharing	33
7.4.4	Push/Subscriptions	34
7.4.5	TKError	35
7.5	Aufgetretene Probleme	36
7.5.1	Subscription	36
7.5.2	Sharing	36
7.5.3	TKSolution	37

1 Pflichtenheft

Christian Pfeiffer, Normen Krug & Johannes Franz

1.1 Zielbestimmung

Es ist eine Lernspiel Software für Grundschulschüler zu entwickeln, welche auf iPads ab iOS Version 10 lauffähig ist. Schüler sollen Lernspiele bzw. Aufgaben bearbeiten können, welche von den Lehrern vorher generiert werden. Nach den Spielen können Schüler ihre eigenen Leistungen ansehen. Auch Lehrer sollen einen Überblick über die Leistungen der eigenen Schüler haben. Die Ausarbeitung der App ist auf ein Semester beschränkt, das bedeutet es stehen 3 Monate zur Umsetzung zur Verfügung. Kurz vor der Abgabe ist die Funktionsfähigkeit der Software durch Tests zu bestätigen.

1.1.1 Musskriterien

1.1.1.1 Allgemein

- a) Die App muss auf iPads mit iOS 11 laufen

1.1.1.2 Login

- a) Login für den Schüler
- b) Login für den Lehrer

1.1.1.3 Schüler

- a) Übersicht über alle freigegebenen Spiele (Spiele von der Schnittstelle abrufen)
- b) Übersicht über erreichte Punktzahlen
- c) Spielbeschreibung anzeigen
- d) Spiel spielen
- e) Ergebnisse anzeigen
- f) Ergebnisse über die Schnittstelle hochladen

1.1.1.4 Lehrer

- a) Übersicht über alle registrierten Schüler
- b) Einladen von Schülern in "Klassen"
- c) Anzeigen von Ergebnissen einzelner Schüler
- d) Aufgaben erstellen

1.1.1.5 Schnittstelle

- a) Abrufen der freigegebenen Spiele für einen Schüler
- b) Schüler zu Aufgaben einladen (durch den Lehrer)
- c) Abspeichern der Ergebnisse der gelösten Aufgaben
- d) Abrufen der Ergebnisse der gelösten Aufgaben (Lehrer)

1.1.2 Wunschkriterien

1.1.2.1 Allgemein

- a) Die App kann auch auf anderen iOS Geräten (iPhone) laufen

1.1.2.2 Login

- a) Alternative Login Methode für den Lehrer

1.1.2.3 Schüler

- a) Übersicht über die vergangenen Spiele
- b) Lösen von Aufgaben unter Zeitdruck
- c) Kindgerechte und einfache Menügestaltung
- d) Schüler soll es ermöglicht werden in eigenem Tempo zulernen

1.1.2.4 Lehrer

- a) Einteilung der Schüler in Klassen
- b) Ranking der Spielergebnisse der Klassen
- c) Individuelle Förderung von Schülern

1.1.2.5 Schnittstelle

- a) Detailliertes Speichern der Spiele für eine Auswertung (gebrauchte Antwortzeit, Statistiken für Klassen)
- b) Abspeichern von Bildern

1.1.3 Abgrenzungskriterien

Das System besitzt keine Schnittstellen zu anderen Produkten. Es existiert keine automatische Erfassung von Benutzern aus Fremddaten.

1.2 Produkteinsatz

Die App soll als Referenz für die Lehre der Fortgeschrittenen Swift 4 Entwicklung des Mobile Computing Studiengangs an der Hochschule Hof dienen.

Hypothetisch soll die App im Rahmen des Grundschulunterrichts eingesetzt werden. Hierbei hat jeder Schüler ein eigenes Tablet und kann mit diesem Aufgaben bearbeiten. Der Lehrer kann den Schülern Aufgaben zuweisen und die Aufgabenergebnisse einsehen.

1.2.1 Anwendungsbereiche

Primär Grundschulen, später Erweiterung für höhere Bildungseinrichtungen denkbar.

1.2.2 Zielgruppen

Schüler / Studenten und Lehrer bzw. Lehrbeauftragte

1.2.3 Betriebsbedingungen

Um die App in allen Funktionen nutzen zu können, wird ein iPad mit iOS 10 mit Internetverbindung vorausgesetzt. Beim Einloggen als Schüler müssen alle freigeschalteten und geteilten Aufgaben abgerufen werden. Eingeloggte Lehrer sollen eine Übersicht über die verfügbaren Aufgabentypen sowie Schüler bzw. deren Klassen haben. Die Pflege der Schnittstelle soll wartungsfrei sein. Die administrative Gewalt (Zuweisung der Aufgaben, sowie Einblick in die Statistik der Schüler) soll bei den Lehrern stehen.

1.3 Produktfunktionen

1.3.1 /F0010/ Einloggen

Ein beliebiger App Nutzer kann sich über den Startscreen einloggen. Als Kennung wird sein iCloud Account verwendet. Die App muss zwischen Schülerbenutzern und Lehrerbenutzern

unterscheiden können. Die Unterscheidung dieser zwei Benutzergruppen geschieht durch unterschiedliche Loginverfahren (Zum Login eines Lehrers muss ein Button gedrückt werden, bzw. ein QR Code verwendet werden). Nach dem Login wird der Benutzer in das seiner Benutzergruppe zugehörige Hauptmenü weitergeleitet (Schüler bzw. Lehrer).

1.3.2 /F0020/ Verfügbare Spiele herunterladen (Schüler/Schnittstelle)

Schülerbenutzer kann im Schülerhauptmenü seine ihm freigegebenen Aufgaben bzw. Spiele einsehen. Das Herunterladen dieser geschieht automatisch nach dem Login. Falls dem Schüler keine Aufgaben freigegeben wurden, wird ihm anstatt der Aufgaben ein Hinweis angezeigt.

1.3.3 /F0030/ Spielinformationen anzeigen (Schüler)

Wählt der Schüler ein ihm freigegebenes Spiel in dem Schülerhauptmenü aus, so werden ihm Informationen über das ausgewählte Spiel (Spielregeln, Schwierigkeit... etc. angezeigt). Über einen "Spiel starten" Button, kann der Schüler das Spiel starten. Mit einer Geste bzw. einem Zurückbutton kann er zurück in das Hauptmenü gelangen.

1.3.4 /F0040/ Spiel spielen (Schüler)

Wählt der Schüler den "Spiel starten" Button in den Spielinformationen aus, wird das Spiel gestartet und er kann die Aufgaben bearbeiten. Nach dem Bearbeiten der Aufgaben werden die Spielergebnisse über die Schnittstelle wieder hochgeladen.

1.3.5 /F0050/ Leistungen anzeigen (Schüler)

Wählt ein Schüler den "Erfolge Tab" in dem Schülerhauptmenü aus, so kann er seine Spielerfolge einsehen. Hierbei wird ihm eine Punktzahl angezeigt, welche die Summe der innerhalb der Lernspiele erreichten Punkte ist. Zudem kann er sich eine Übersicht über seine letzten Spiele und die darin richtig bzw. falsch gelösten Aufgaben anzeigen lassen.

1.3.6 /F0060/ Schüler in Klassen einladen (Lehrer/Schnittstelle/Schüler)

Bevor der Lehrer Aufgaben verteilen kann, muss er seine Schüler in eine Klasse einladen. Wenn die Schüler die Einladung der Lehrer annehmen, werden sie der Klasse hinzugefügt.

1.3.7 /F0070/ Aufgaben Klassen zuteilen

Ein Lehrer kann an einzelne Schüler spezielle Aufgaben stellen.

1.3.8 /F0080/ Ergebnisse einzelner Schüler anzeigen

Für den Lehrer muss es möglich sein, die Ergebnisse der Schüler einzusehen.

2 Schülerteam: MathePiano Spiel, Login & Schülerhauptmenü

Christian Pfeiffer, Normen Krug & Johannes Franz

2.1 Einleitung

In diesem Abschnitt werden die Ziele und die Motivation des Projektes definiert. Dabei werden unter anderem die Erwartungen an das Projekt genannt.

2.1.1 Motivation

Die Hauptmotivation des Projektes war das Lernen und Einarbeiten in neue Apple Frameworks (wie *SpriteKit* und *CloudKit*) und Erfahrungen sammeln in der Zusammenarbeit mit mehreren Entwicklerteams, welche gleichzeitig an einem Projekt arbeiten. Deshalb war es zwingend notwendig, sich mit anderen Teams zu verständigen und auszutauschen.

2.1.2 Ziele

Als Ziele der Studienarbeit wurden folgende Punkte definiert:

- Kinderfreundliches Design und Layout
- Erstellen eines Mathelernspiels
- Aufgaben, die von Lehrern erstellt werden, anzeigen und in eine spielbare Form überführen
- Die vom Schüler beantworteten Fragen an den Lehrer weiterleiten
- Den Schülern die Möglichkeit bieten die Spiele im Endlosmodus, unabhängig der von Lehrer zugewiesenen Aufgaben, zu spielen
- Lernen eines neuen Apple Frameworks (*SpriteKit*)
- Erfahrungen sammeln in Zusammenarbeit mit anderen Teams

2.2 Spezifikation

2.2.1 Schülerhauptmenü

2.2.1.1 Einleitung

Primäre Benutzerzielgruppe von Teachify sind Grundschüler. Deswegen ist es wichtig die Hauptmenüs so simpel und zielführend wie möglich zu gestalten. Weiterhin ist es nötig auch die Designsprache so kindgerecht wie möglich zu gestalten.

2.2.1.2 Ziele

- Einfache Loginmöglichkeiten
- simple Navigationsmöglichkeiten zu Spielen
- Unterscheidung zwischen Aufgaben und Endlosspielen
- Simple Darstellung von Hintergrundaktivitäten

2.2.1.3 Einfache Loginmöglichkeiten

Da es in Teachify zwei Benutzergruppen (Lehrer und Schüler) gibt, muss es eine Möglichkeit geben, wie sich diese authentifizieren und danach einloggen können. Als Grundlage unserer App dient das von dem Schnittstellenteam entwickelte "TeachKit". Dies stellt Teachify die von iCloud zur Verfügung gestellten Möglichkeiten zur Verfügung. Zur Authentifizierung des Benutzers verwenden wir daher die iCloud Accounts. Innerhalb dieser iCloud Accounts ist aber nicht definiert, ob ein Benutzer zu der Gruppe der Lehrer oder der Schüler gehört. Deswegen wird der Benutzer beim Start von Teachify mit der LoginView begrüßt, innerhalb derer er seine Rolle wählen kann.

2.2.1.4 Simple Navigationsmöglichkeiten zu den Spielen

Nach dem Login als Schüler wird der Benutzer zu dem Schülerhauptmenü weitergeleitet. Innerhalb dessen soll er eine kurze Übersicht über seine Statistiken und bereits erbrachten Erfolge bekommen. Prominent sollen die Spiele bzw. Übungen welche der Schüler spielen bzw. erarbeiten kann, dargestellt werden. Hierfür sollen die in iOS 11 neu hinzugekommenen Cards dienen, welche ebenso prominent im AppStore verwendet werden. Als weitere Interaktionsmöglichkeiten soll der Schüler in der Lage sein, neue Aufgaben aus der Schnittstelle herunterzuladen und Einladungen, welche in Form von QR-Codes verschickt werden, einzuscannen und somit anzunehmen

2.2.1.5 Unterscheidung zwischen Aufgaben und Endlosspielen

Die Schüler sollen nicht nur in der Lage sein, Aufgaben, welche von den Lehrern gestellt wurden zu bearbeiten, sondern die implementierten Spiele auch in einen Endlosmodus spielen können. Bei dem Endlosmodus kann der Benutzer endlos lange Aufgaben bearbeiten, welche vorher per Zufallsgenerator generiert wurden. Der Endlosmodus basiert nicht auf Aufgaben, welche von den Lehrern gestellt wurden.

2.2.1.6 Darstellung von Hintergrundaktivitäten

Da in Teachify oft zeitaufwendige Aufgaben im Hintergrund ausgeführt werden (wie das Abrufen von Daten von der Schnittstelle), muss der Benutzer auch über diese Abläufe informiert werden. Dies soll durch Einblendungen wie einem Progress Indicator, während die Download Operationen laufen, umgesetzt werden.

2.2.1.7 User Interface

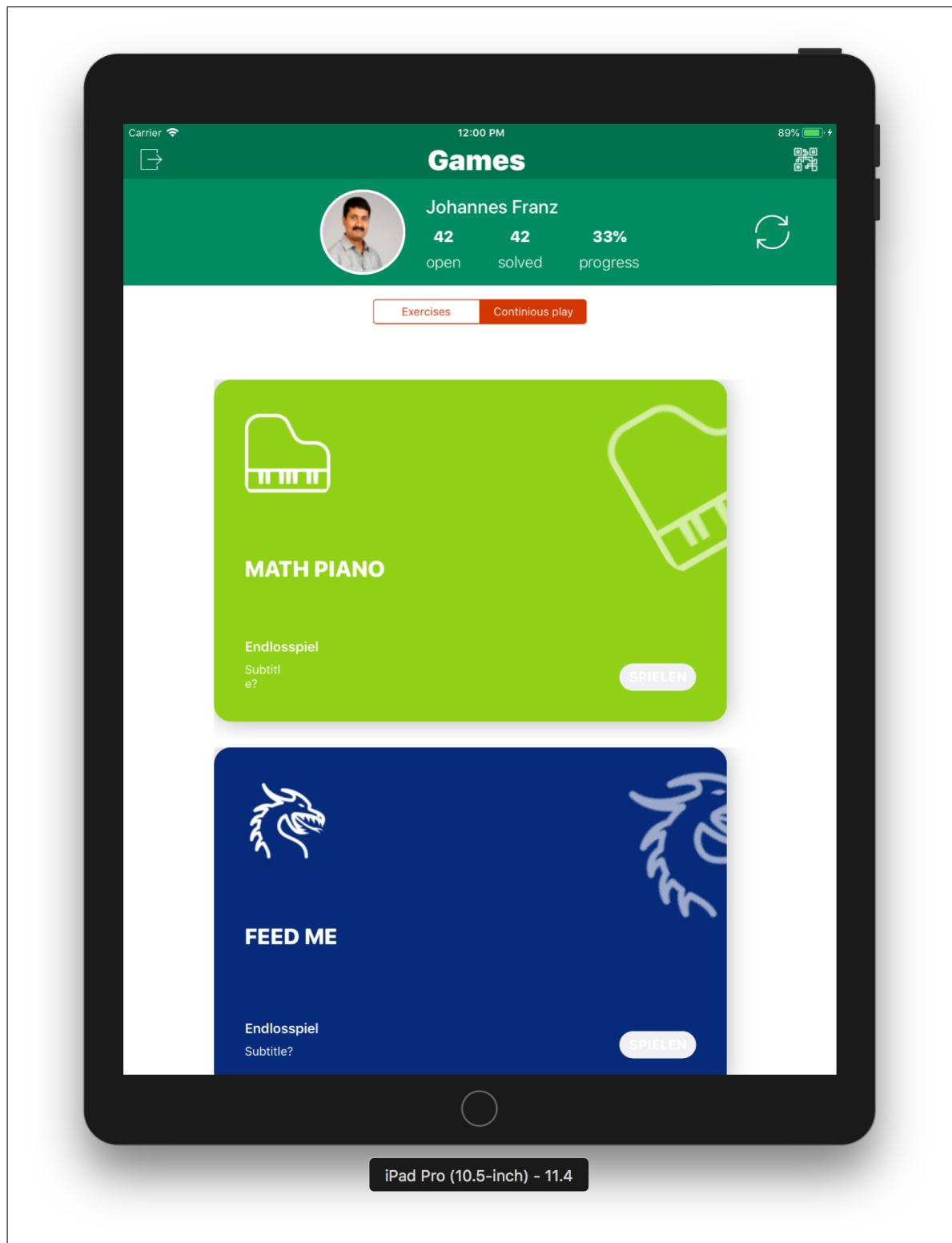


Abbildung 2.2: Das Schülerhauptmenü (Schüler)

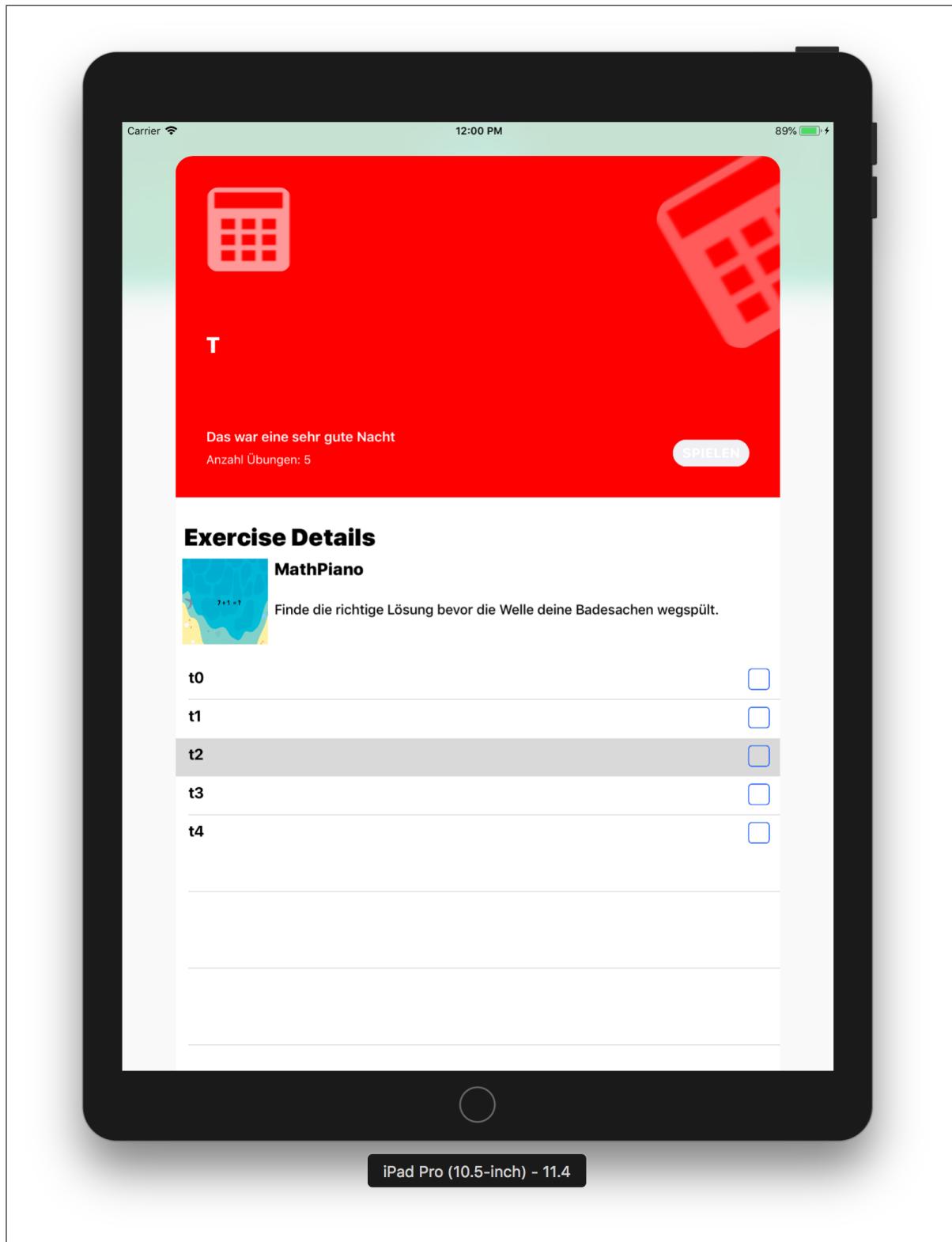


Abbildung 2.3: Detailansicht einer Übungsaufgabe (Schüler)

2.2.2 Mathe Piano Spiel

Bei der Entwicklung des Spiels war es wichtig, möglichst schnell einen funktionsfähigen Prototypen zu erstellen. Dieser wurde im Verlauf des Projektes immer weiter verbessert.

2.2.2.1 Game Engine

Als Grundlage wurde die von Apple entwickelte Game Engine namens *SpriteKit* verwendet. Diese Engine hat einige Vorteile gegenüber anderen Spielengines:

- Gute Dokumentation
- Wiederverwendungen bereits gelernter Paradigmen
- Einfache Integrationsmöglichkeit in die App
- Einfache Anbindung an andere iOS API's
- Swift als Programmiersprache
- Schnelles Entwickeln und Testen von Funktionen durch Swift Playgrounds

2.2.2.2 Herausforderungen

Nach der ausgiebigen Einarbeitung in das *SpriteKit Framework* haben sich einige Hürden ergeben. Das Verwenden von dynamischen Buttons ist nicht trivial, weil es keine Buttons per Default gibt. Deshalb muss eine eigene Button Klasse implementiert und mit der gewünschten Funktionalität erweitert werden. Des Weiteren war es schwierig den Code sinnvoll zu strukturieren, aufgrund der durch Spiel vorgegebenen skriptartigen Programmierung.

2.2.2.3 Testen des Spieles

Um das Spiel sinnvoll und schon während des Entwicklungsprozesses testen zu können, musste ein Generator entwickelt werden, der zufällige Aufgaben generiert. Dieser befindet sich in der *RandomQuestionGenerator.swift* Klasse.

2.2.2.4 Anbindungen an interne Schnittstellen

Von Anfang an musste darauf geachtet werden dass, das Spiel an die interne Schnittstelle angebunden werden muss, die von einem anderen Team entwickelt wurde. Da die Schnittstelle nicht von Beginn an verfügbar ist, muss eine temporäre Datenstruktur implementiert werden. Diese soll einfach austauschbar und erweiterbar sein.

2.2.2.5 User Interface

Bei der Gestaltung des User Interfaces muss explizit darauf geachtet werden, dass die Software primär von Kindern bedient wird. Das bedeutet, dass die Größe der Bedienelemente deutlich größer ausfallen muss als bei herkömmlichen Applikationen.

2.3 Implementierungsphase

An dieser Stelle wird die Implementierung der Aufgaben beschrieben. Dabei wird auf die Schnittstellenanbindung, das Mathe Piano Spiels sowie das User Interface eingegangen.

2.3.1 Mathe Piano Spiel

2.3.1.1 SpriteKit

Durch die Verwendung von den generischen Klassen *BasicNode*, *BasicButton* konnte schnell auf Änderungen reagiert werden. Besonders zu betrachten ist der *BasicButton*. Dieser implementiert das *BasicButtonDelegate*.

```
protocol BasicButtonDelegate: class {
    func basicButtonPressed(_ button: BasicButton)
}
```

Mit Hilfe dieses Protokolls können Buttons mit beliebigen Funktionen erstellt werden. Alle Nodes werden effizient in der *BasicScene* verwaltet. Die *BasicScene* wird vom *MathPianoGameViewController* erstellt und angezeigt. Wie bei anderen Spielen üblich wird der State der Scene mit jedem neu gerenderten Bild verändert. Dadurch ist es egal auf welchen Device das Spiel läuft. Schon am Beginn der Implementierungsphase wurde die Software so strukturiert dass sie in mehreren Modi gestartet werden kann. Aufgrund des Modi wird der interne State an den jeweiligen Modus angepasst. Diese Modi werden in einen *Enum* abgebildet. Dieses hat den Vorteil, dass es einfach erweiterbar ist, falls noch ein neuer Modus dazukommen würde. Die Schwierigkeit des Spiel war es, immer in einem konsistenten State zu behalten. Das war besonders schwierig bei den vielen verschiedenen Iterationsschritten die das Spiel durchlebt hat. Der größte Iterationsschritt war die Vorbereitung auf die Zahlenerkennung. Für diese Feature wurde parallel eine Testapplikation entwickelt. Bei diesen Prototypen war relativ schnell klar dass, das ausgewählte Modell aus mehreren Gründen, ungeeignet war. Die Zahlenerkennung funktioniert nicht ausreichend gut, um sie als Ersatz für die *Buttons* zu verwenden. Diese schlechte Erkennungsrate ist auf zwei Hauptfaktoren zurückzuführen.

Das Modell wurde mit amerikanischen Zahlen trainiert. Diese unterscheiden sich geringfügig von den deutschen. Durch diese Sprachunterschiede, vor allen bei der Zahl 4, ist die Erkennungsrate messbar schlechter. Der zweite und größere Faktor ist die Art der *Inputschicht* des *neuronalen Netzes*. Das neuronale Netz benötigt Bilder als Eingabequelle. Die Bilder müssen in einen Format von 28x28 vorliegen. Das hat den entscheidenden Nachteil das man das Bild, welches man aus der *UIView* generiert, stark skalieren muss. Durch diese Skalierung gehen viele Merkmale verloren.

2.3.1.2 Game Models

Der *MathPianoGameViewController* holt sich alle *TKExercises* aus den *TKModelSingleton*, welches in 6.2.1 *TKModelSingleton* beschrieben ist. Diese *TKExercise* werden in eine Spielinterne Datenstruktur überführt. Das *MathPianoQuestionModel* ist die interne Repräsentation

des *TKExercise* nur auf die Anforderung des Spiels optimiert. Die einzelnen *MathPianoQuestionModels* werden im *PianoGame* zusammengefasst. Für diese Umwandlung wurde ein Converter implementiert. Im Hintergrund werden, während der Benutzer spielt, die Antworten in ein Feedback Model gespeichert. Für diese Aufgabe wurde das *MathPianoGameFeedbackModel* implementiert.

Um einen sinnvollen Endlosspielmodus zu implementieren, werden generierte Fragen benötigt. Für diesen Zweck wurde der *RandomQuestionGenerator* erstellt.

2.3.2 User Interface

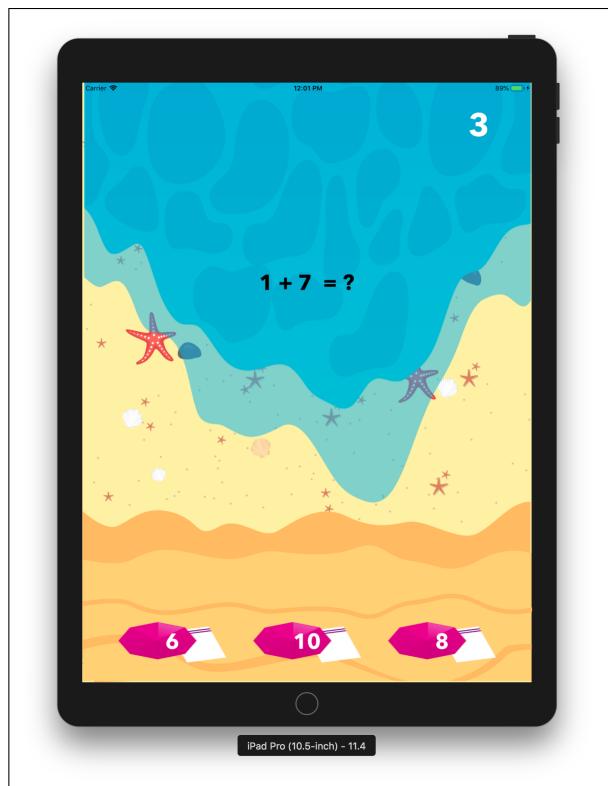


Abbildung 2.4: Das Mathe Piano Spiel (Schüler)

Um das Mathepianospiel so Zielgruppen freundlich wie möglich zu gestalten, wurden verschiedene Grafiken entwickelt, welche ein typisches Strandszenario abbilden. Bei der Farbwahl wurden helle freundliche Farbtöne gewählt. Inhalt des Spiels ist eine Welle, in welcher eine Matheaufgabe abgebildet ist. Der Spieler muss die richtige Antwort auf die Matheaufgabe auswählen, bevor die Welle zu nahe kommt und die Badesachen weg spült.

2.4 Fazit

Teachify war ein herausforderndes Projekt für das 6. Semester. Dies stellte das ganze Team immer wieder vor anspruchsvolle Aufgaben. Der Umgang mit Git (Teachify Projekt Link) brachte

zugleich viele Vorteile, aber auch Herausforderungen.

Durch die unterschiedlichen Herangehensweisen (Design vs. Funktion) und den damit weitgehend einhergehenden Verzicht auf einen Prototypen lenkte das Projekt gegen Ende des Projektzeitraums auf einen “Big-Bang“ Ansatz. Positiv zu erwähnen war das Zusammenwachsen des Teams und die Zusammenarbeit untereinander. So hatten die meisten Teams eine Domäne in die sie sich eingearbeitet hatten und mussten bei der Überschneidung ihrer Gebiete zusammenarbeiten.



Abbildung 2.1: Der Login Bildschirm (Schüler)

3 Schüler Game Teachify Bird

Christian Pöhlmann

3.1 Konzept und Spielidee

Die Grundidee war eine Spielvariante auf der Basis von FlappyBird zu entwickeln. Bei FlappyBird geht es darum einen Vogel an Hindernissen, die sich von rechts nach links über den Bildschirm bewegen vorbei zu navigieren. Durch tippen auf dem Bildschirm wird die Höhe des Vogels beeinflusst, der sich an einer festen Position befindet, und von den sich bewegenden Hindernissen getroffen werden kann. In unserer Spielvariante wird eine englische Vokabel vorgegeben, und der Spieler muss den Vogel von einer Auswahl an deutschen Vokabeln durch die richtig übersetze Vokabel navigieren. Bei der richtigen Antwort erhält der Spieler Punkte, bei der falschen verliert er Leben. Ziel des Spieles ist es soviel Punkte wie möglich zu sammeln.

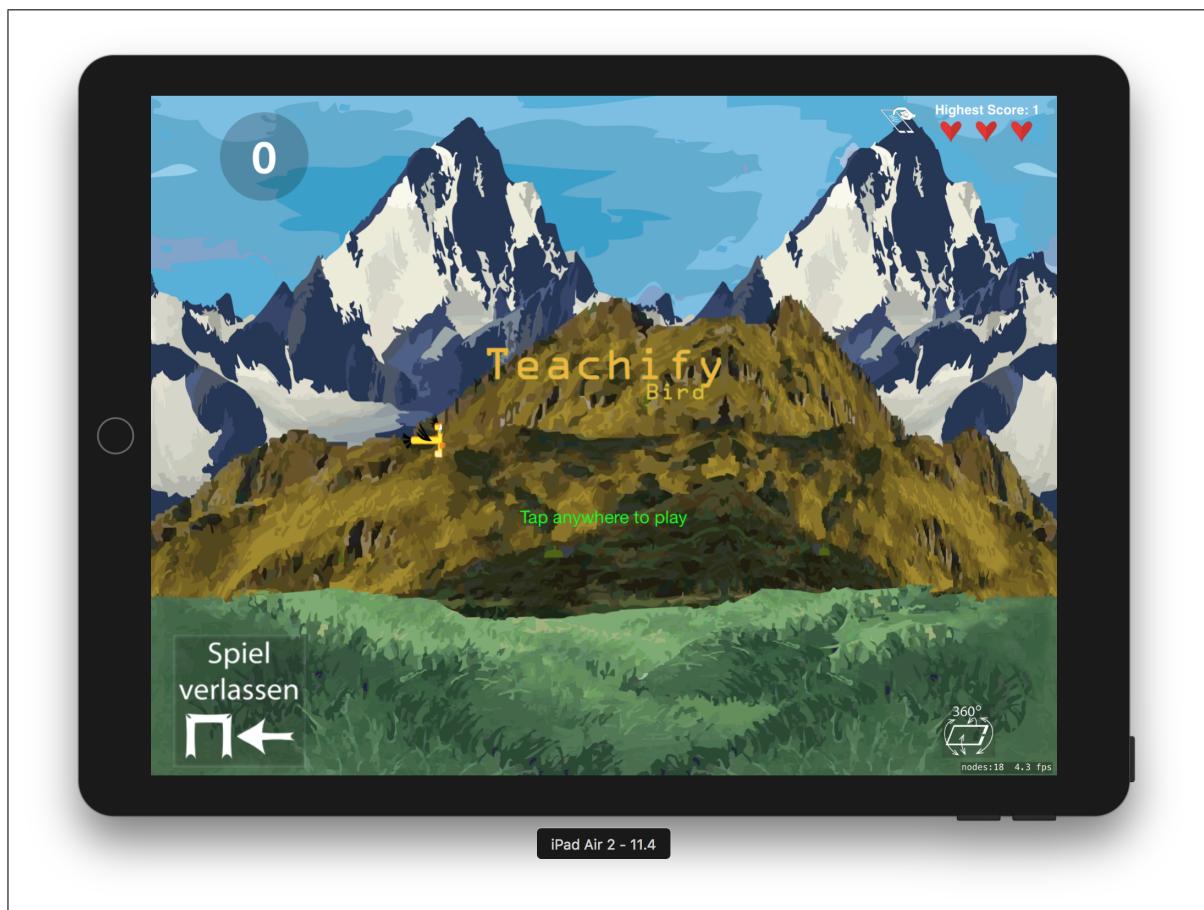


Abbildung 3.1: Teachify Bird Spiel StartScreen



Abbildung 3.2: Teachify Bird Spiel

3.2 Technische Umsetzung

Umgesetzt wurde dieses Spiel mit dem SpriteKit und Swift 4. Für die Spielvariante mit der Gerätebewegung wird ein physisches iPad benötigt.

3.3 Aufgaben

Es wird eine englische Vokabel angezeigt, als Übersetzung gibt es vier deutsche Begriffe, von denen einer die richtige Übersetzung ist. Die Antworten werden in einem geringen Abstand zueinander leicht versetzt übereinander angeordnet. Die richtige Antwort erhält eine zufälligen Position, um den Spieler zum nachdenken zu ermutigen. Die Antworten bewegen sich von rechts nach links, und der Spieler kann die richtige Antwort durchfliegen, und einen Punkt sammeln, oder an einer falschen Antwort ein Leben verlieren. Die Aufgabenstellung wird so lange die Aufgabe gültig ist unten links konstant angezeigt. Ist ein Aufgaben-Set zu drei viertel über den Bildschirm gelaufen, wird es gegen einer neuen Aufgabe ersetzt. Zu jedem Aufgaben-Set gibt es einen Bonuspunkt der eingesammelt werden kann.

3.4 Animieren des Backgrounds

Um eine Flugbewegung des Vogels zu verdeutlichen bewegt sich das Hintergrundbild in einer langsam Geschwindigkeit von rechts nach links. Um die Hintergrundlandschaft realistischer wirken zu lassen wird der Vordergrund der Landschaft langsamer bewegt als dieser Hintergrund. Für die Länge der Landschaft wiederholen sich die Hintergrundbilder.

3.5 Navigieren des Vogels durch Neigen des Gerätes

Als alternative zu der mühsamen Steuerung des Birds durch tippen auf dem Bildschirm wurde die Steuerung durch neigen des Gerätes eingeführt. Das Bird kann durch auf und abwärts neigen des Gerätes nach oben und unten gesteuert werden. Zusätzlich kann durch neigen nach links und rechts innerhalb des Bildschirms vorwärts und rückwärts geflogen werden. Beim Starten des Spieles wird die aktuelle Lage des Gerätes vom Sensor ausgelesen und als Nullstellung verwendet. Die Nullstellung wird nur bei der Horizontalachse je nach Gerätelage kalibriert. Die Steuerung kann über einen Buttons rechts unten vor starten eines Spieles zwischen Tab- und Sensor- Steuerung umgeschaltet werden. Der Aktive Bedienungsmodus wird oben rechts angezeigt. Ist im Hauptmenü der Sensor aktiv, wird das TeachifyBird Logo durch neigen des Gerätes bewegt. Für die Sensor-Steuerung wird CoreMotion verwendet.

3.6 Punkte und Leben

Ein Spiel hat vier Leben, welche oben rechts als Symbole angezeigt werden. Wird eine Antwort durch umfliegen der Aufgabe ausgelassen, werden zwei Punkte abgezogen. Bei weniger als zwei Punkten wird ein Leben abgezogen.

3.7 Animieren des Vogels

Um das Bird lebendig zu gestalten wurde es animiert und die Flügel in Bewegung versetzt.

3.8 Quellenangaben

- Backgroundmusic
<https://www.bensound.com/royalty-free-music/track/ukulele>
- Coin-Sound
<http://soundbible.com/free-sound-effects-3.html> Must Credit
- Died-Sound
<https://www.cayzland.de> Music Explosion

4 Das Spiel FeedMe

Angelina Scheler

4.1 Der Entwurf

Die Idee von FeedMe war es Multiple Choice Fragen interessanter zu gestalten, damit die Schüler motiviert bleiben. Eigenschaften des Spiels:

- Multiple Choice
- für alle Fächer geeignet
- Monster muss mit richtiger Antwort gefüttert
- bei falscher Antwort wird ein Leben abgezogen
- verliert der Spieler zu viele Leben, wird das Monster wütend

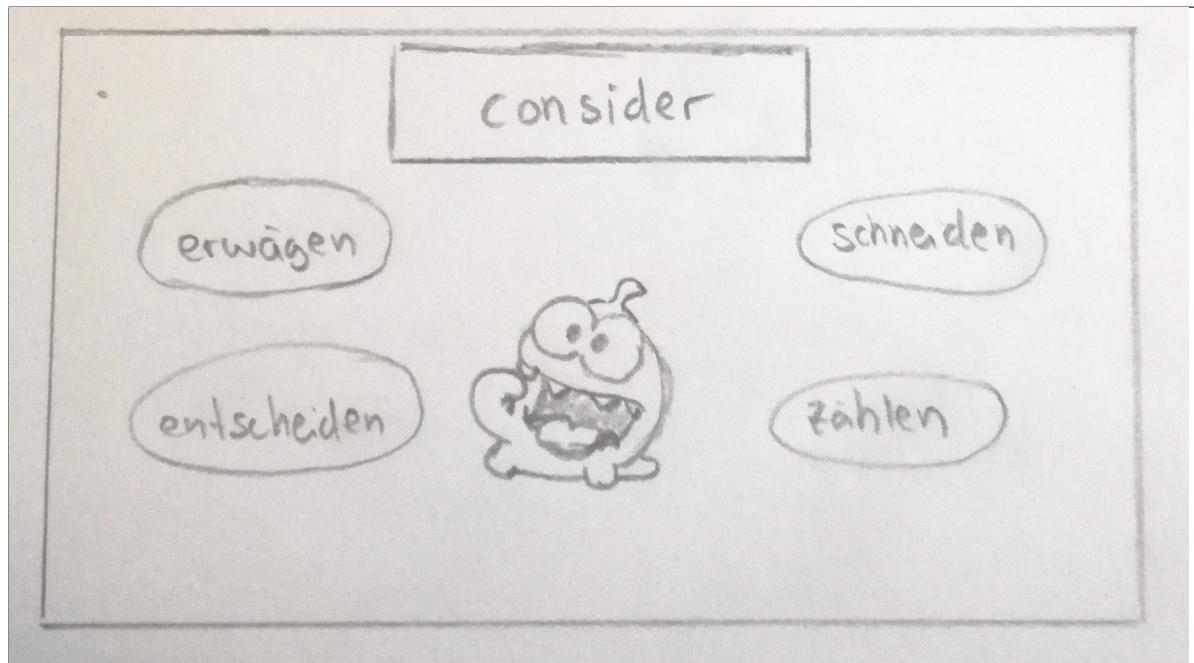


Abbildung 4.1: Erster Entwurf

4.2 Die Umsetzung und Implementierung

Das Spiel wurde mit dem SpriteKit umgesetzt



Abbildung 4.2: Game Screens

In einem SpriteKit Scene File wurden Color Sprites und Labels als Nodes eingefügt, den Sprites Assets zugewiesen, Skalierung und Position angepasst. (*GameFeedMeScene.sks*) In *GameFeedMeSwift.swift* wurden dann die Funktion umgesetzt.

4.2.1 Die Nodes

Für alle benötigten Nodes aus der Scene wurden Variablen angelegt z.B.

```
var frage: SKLabelNode?
```

und in der didMove() Methode initialisiert z. B.

```
self.frage = question?.childNode(withName: "frage") as? SKLabelNode
```

4.2.2 Der Sound

Über ein SKAudioNode wird der Sound initialisiert. Hintergrundmusik, Touch, falsche/richtige Antwort, Game Over und New Game. Beispielsweise die Hintergrundmusik als Loop in der didMove() Methode

```
let backgroundMusic = SKAudioNode(fileNamed: "sky-loop.wav") backgroundMusic.autoplayLooped = true addChild(backgroundMusic)
```

und beim Bewegen der „Früchte“ als einmaliger Ton in touchesBegan()

```
run(SKAction.playSoundFileNamed("beeps.wav", waitForCompletion: false))
```

4.2.3 Die Touch-Events

Um die Antworten bewegen zu können sind 3 Funktionen nötig:

- touchesBegan()
- touchesMoved()
- touchesEnded()

In der touchesBegan() Funktion wird eine location variable initialisiert, welche die Postion des „touch“ enthält. Befindet sich die location auf einer der Antworten wird einer Hilfsvariable „movableNode“ die Postion zugewiesen und die ursprüngliche Position in einer weiteren Hilfsvariable „originalposition“ zwischengespeichert und in der touchesMoved() Funktion ständig aktualisiert. Endet das Touch Event wird in der touchesEnded() Methode das „movableNode“ an die „originalPosition“ verwiesen und „movablenode“ zurückgesetzt.

```
//Auszug
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    if let touch = touches.first {
        let location = touch.location(in: self)
    }
}

override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
    if let touch = touches.first, movableNode != nil {
        movableNode!.position = touch.location(in: self)
    }
}

override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
    if let touch = touches.first, movableNode != nil {
        movableNode!.position = touch.location(in: self)
    }
}

movableNode?.position = originalPosition!
movableNode = nil}
```

Abbildung 4.3: Die Touch-Events

4.2.4 Die Animation

Der Drache wurde über eine SKAction animiert. Zunächst wurde ein Images.Atlas Ordner mit verschiedenen Bildern des Drachens angelegt.



Abbildung 4.4: Drachen Assets

Die Bilder wurden in ein SKTexture Array „dragonFrames“ bzw. „evilDragonFrames“ in der buildDragon() Funktion eingelesen. In der animateDragon() Funktion wird über dragon.run auf das Array zugegriffen und über ein Zeitintervall die Frames des SKSpriteNode des Drachen geändert.

```
dragon?.run(SKAction.repeatForever( SKAction.animate(with: dragonFrames, timePerFrame: 0.5, resize: false, restore: true)), withKey: "walkingInPlaceDragon")
```

Die Leben in Form von Herzen (siehe Abb. GameScreen) werden ebenfalls über eine SKAction animiert.

```
einblenden über „fadeIn()“ heart?.run(SKAction.fadeIn(withDuration: 2.0))
```

ausblenden über „fadeOut()“ `heart?.run(SKAction.fadeOut(withDuration: 1.0))`
mit „withDuration“, kann die Geschwindigkeit angegeben werden

4.2.5 Das Game Play

Der User startet mit 3 Leben welche durch die Herzen angezeigt wird, durch ziehen der „Früchte“ auf den Drachen wird die Antwort geprüft, bei falscher Antwort verliert der User ein Leben, befindet er sich kurz vor dem „Game Over“ ändert der Drache die Farbe „wird böse“.

Durch initialisieren eines Integer Arrays welches 4 zufällige Zahlen enthält, werden die Frage, die Antworten generiert und die Antworten den Nodes zufällig zugeordnet.

Ist der User „Game Over“ bzw. hat alle Leben verloren, kann er entweder das Spiel Neustarten oder dieses Beenden über den jeweiligen Button.

5 Lehrer-Seite

Philipp Dümlein, Bastian Kusserow, Maximilian Sonntag

5.1 Lehrer UI

Zu Beginn des Projekts musste zunächst eine funktionale und optisch ordentliche Benutzeroberfläche für den Lehrer entwickelt werden. Hierbei waren Ziele moderne UI-Elemente zu verwenden, sowie erste Erfahrungen bei der Implementierung von Views im Code zu sammeln. Ganz besonders zeigt sich dies auf dem Homescreen des Lehrers.

5.1.1 Homescreen

Auf dem Homescreen wurde durch die Kombination von mehreren CollectionViews (CVs), die unterschiedlich angepasst wurden, ein funktionales und übersichtliches UI geschaffen. Die Aufteilung wurde wie folgt implementiert:

- CV zur Anzeige, Auswahl und Erstellung von Klassen
- CV zur Anzeige, Auswahl und Erstellung des Faches
- Animierter Indikator zur Anzeige des aktuell gewählten Faches
- CV zur Anzeige, Auswahl und Erstellung von Dokumenten

Eine Besonderheit der CV der Dokumente ist die Sharing-Funktion, welche durch langen Druck auf ein Dokument aufgerufen werden kann. Nach erfolgreichem Sharing wird in einem Overlay der QR-Code mit dem Freigabelink angezeigt. Dieser kann dann von Schülern verwendet werden um Zugriff auf das geteilte Dokument zu bekommen.

5.1.2 Erstellen von Aufgaben

Bei der Erstellung von Aufgaben muss der Lehrer zunächst auswählen um welches Fach es sich handelt. Daraufhin muss ausgewählt werden, welche Art von Aufgaben erstellt werden sollen. Hierbei muss neben dem Fach auch eine genauere Angabe zum Aufgabentyp gemacht werden. Aktuell gibt es Folgende Auswahlmöglichkeiten:

- Englisch - Vokabeln
- Englisch - Grammatik
- Englisch - Synonyme
- Mathematik - Addition

- Mathematik - Subtraktion
- Mathematik - Division
- Mathematik - Multiplikation

Danach muss noch eine Auswahl getroffen werden zu welchem Spiel die Aufgaben auf Schülerseite werden sollen. Je nach Auswahl des Aufgabentyps werden unterschiedliche Views geöffnet, auf denen spezifische Einstellungen zur Auswahl gesetzt werden. Im Rahmen der Studienarbeit wurde bisher die Erstellung von Englisch Vokabeltests und von Mathematik-Aufgaben mit allen Grundrechenarten implementiert.

5.2 Implementierung

5.2.1 Homescreen

5.2.1.1 Klassen- und Fächer-CVs

Die CVs zur Anzeige und Auswahl der Klassen und Fächer verfügen neben den einzelnen Elementen auch Einträge zur Anzeige von allen Unterelementen und zum Hinzufügen von Klassen/Fächern. Letzteres geschieht über einen eigens geschriebenen ViewController, welcher modal als Overlay angezeigt wird. Hier kann der Name für die Klasse bzw. das Fach eingetragen werden. Wird der Save-Button betätigt, wird automatisch ein Objekt in die Cloud hochgeladen.

Die einzelnen Elemente der CVs können selektiert werden. Daraufhin werden die jeweils darunterliegenden CVs entsprechend angepasst. Als visuelles Feedback für den Nutzer wird bei Klassen der Name des ausgewählten Eintrags fett geschrieben. Für die Fächer gibt es einen weißen Indikator, welcher unterhalb des ausgewählten Eintrags angezeigt wird und beim Wechsel animiert zum neuen Eintrag springt.

5.2.1.2 Dokumente-CV

Die CV zeigt immer die Dokumente des aktuell ausgewählten Fachs einer Klasse. Die wohl wichtigste Funktion ist das Sharing einzelner Dokumente. Dies wurde durch eine Long-Tap-Action und einem sog. *MenuItem* umgesetzt. Wird auf dieses gedrückt, öffnet sich ein modaler SharingController, in dem alle Sharing-Optionen angezeigt werden.

Hier wird auch der eigene Share-Service '*Sharing*' angezeigt, welcher dafür sorgt, dass die Aufgabe in die Cloud geladen wird und der Link für die Freigabe zurückgeliefert und als QR-Code angezeigt wird. Dieser kann dann von Schülern verwendet werden um Zugriff zur geteilten Aufgabe zu erhalten.

5.2.2 Herunterladen der Clouddaten

Beim Start der Lehrerseite werden zunächst alle bereits durch die App hochgeladenen Daten auf dem iCloud Account des Nutzers heruntergeladen. Aufgrund des Datenmappings ist hierbei die Reihenfolge enorm wichtig. Zur Erklärung:

- Eine Klasse ist eigenständig
- Ein Fach gehört zu einer Klasse
- Ein Dokument gehört zu einem Fach
- Eine Aufgabe gehört zu einem Dokument

Da die jeweiligen Unterobjekte ohne deren Überobjekt nicht erreichbar sind, müssen also zunächst Klassen, dann deren Fächer, daraufhin die jeweiligen Dokumente und zuletzt die entsprechenden Aufgaben heruntergeladen werden. Für den Download selbst bietet die Schnittstelle zwar entsprechende Methoden, dennoch stellt die Reihenfolge der Downloads ein Problem dar.

5.2.2.1 Problematik

Beim Herunterladen der Daten ist es durch das Mapping zwingend notwendig eine bestimmte Reihenfolge einzuhalten. Da Downloads aber standardmäßig asynchron ablaufen, kann nie genau gesagt werden, wann welche Daten verfügbar sind. Eine Möglichkeit des Downloads ist die Verschachtelung der einzelnen Aufrufe. Das bedeutet, dass zunächst alle Klassen heruntergeladen werden, danach alle Fächer, alle Dokumente und alle Aufgaben. Dies ist zwangsläufig notwendig, im Code allerdings extrem unübersichtlich und unsauber. Daher wurde eine zwar etwas aufwändiger, aber deutlich schönere Lösung implementiert.

5.2.2.2 Lösung

Um die Verschachtelung sauber zu Implementieren wurde auf *Operations* zurückgegriffen. Es wurde eine *BaseOperation* implementiert, welche von *Operation* erbt. Hierdurch kann der genaue Zustand eines Downloads gesteuert/abgerufen werden. Es wurden für den Download von Klassen, Fächern, Dokumenten und Aufgaben eigene Operationen implementiert, welche von der *BaseOperation* erben. Beim Download werden nun lediglich die Completionblocks angegeben, um die heruntergeladenen Daten entsprechend zu setzen.

Die einzelnen Operationen werden dann in einer *OperationQueue* ausgeführt. Die Reihenfolge der Downloads wird durch sog. *Dependencies* gesteuert. Die Dependencies sorgen dafür, dass ein in der Reihe später stehender Block erst ausgeführt wird, wenn der vorhergehende beendet ist. Außerdem wird maximal eine Operation gleichzeitig ausgeführt, wodurch zusätzlich verhindert wird, dass Downloads gestartet werden, für die die erforderlichen Daten noch nicht vorhanden sind.

5.3 Aufgaben erstellen

Beim Erstellen von Aufgaben sind insbesondere die Mathematik-Aufgaben interessant. Es kann mit Hilfe zweier Picker ausgewählt werden, in welchem Bereich sich die Operanden der Rechenoperation befinden sollen. Zudem kann angegeben werden, ob diese Zahlen auch negative Werte annehmen dürfen. In Abhängigkeit der Auswahl werden die Picker entsprechend angepasst. Das bedeutet, negative Zahlen sind verfügbar bzw. nicht verfügbar und der Zahlbereich umfasst immer mindestens eins. Das bedeutet auch, dass beim Verändern der Picker die jeweils andere Seite entsprechend geändert wird, sodass ein positiver Zahlbereich entsteht.

5.4 Fazit

Der aktuelle Implementierungsstand umfasst den größten Teil der Grundfunktionalität. Das Erstellen und Teilen von Aufgaben ist grundsätzlich möglich und funktioniert. Im Rahmen der Studienarbeit konnten aus zeitlichen Gründen nicht mehr Aufgabentypen, Fächer und Operationen implementiert werden, die Rahmenbedingungen hierfür sind aber für eine zukünftige Erweiterung der Anwendung gegeben und funktionsfähig.

Die Implementierung bis zu diesem Punkt war umfangreich, die Aufgaben der Lehrerseite umfassten aber leider kaum neue Komponenten. Dennoch wurde erstmals mit XIB-Dateien gearbeitet, UI und Constraints im Code erzeugt und mit Operations gearbeitet. Somit konnten trotzdem neue Techniken und Funktionalitäten getestet und erlernt werden.

6 Gemeinsames Datenmodell zur Speicherung der Schnittstellendaten

Christian Pfeiffer & Bastian Kusserow

6.1 Ziele

Zur Umsetzung des temporären Datenmodells haben wir uns folgende Ziele gesetzt:

- Vermeidung von Redundanz
- Wiederverwendbarkeit von Code
- Ressourcenschonendes Speichern von heruntergeladenen Schnittstellendaten
- Vereinheitlichter Zugriff und Downloadabwicklung (Fetch) von Schnittstellendaten

6.2 Implementierung

6.2.1 TKModelSingleton

Um das Ziel des vereinheitlichten Zugriffs auf Schnittstellendaten zu erfüllen, wird eine Singleton Klasse zum Abspeichern der Schnittstellendaten verwendet. Das Besondere an einem Singleton ist, dass es immer nur ein Objekt der Klasse existiert, welches von der Singleton Klasse selbst erzeugt und verwaltet wird.

```
class TKModelSingleton {  
    static let sharedInstance = TKModelSingleton()  
    var downloadedClasses : [TKClass] = []  
    var downloadedSubjects : [TKSubject] = []  
    var myTKRank : TKRank?  
  
    private init (){}  
}
```

In unseren *TKModelSingleton* werden verschiedene Variablen gehalten, welche an verschiedenen anderen Stellen in der App benötigt werden. Die Konstante *sharedInstance* hält die Instanz des Singleton. Die Variable *downloadedClasses* hält die heruntergeladenen TKClass Objekte in einem Array, welche von der private Database des Nutzers gefetched wurden. In der Variable *downloadedSubjects* werden die TKSubjects abgespeichert, welche von der shared Database des Nutzers stammen. Die Variable *myTKRank* enthält den TKRank, auf welchen der Controller bei dem letzten Fetch initialisiert wurde.

6.2.2 TKFetchController

Als Zugriffsschicht auf den TKModelSingleton wurde der TKFetchController implementiert. In ihm sind einerseits Getter und Setter Methoden für den Zugriff auf den Singleton implementiert. Andererseits ist in diesem auch die Logik für die Fetch Funktionalitäten von der Schnittstelle implementiert.

Im folgenden Codeblock soll exemplarisch dargestellt werden, wie Daten aus dem TKFetchSingleton bereit gestellt werden:

```
func getSubjectAndDocumentForCollectionIndex(index : Int) -> (TKSubject,
    ↪ TKDocument) {
    var myindex = index
    for element in model.downloadedSubjects {
        if ((element.documents.count-1) < myindex) {
            myindex -= element.documents.count
        }
        else {
            return (element, element.documents[myindex])
        }
    }
    return (TKSubject(name: "Unknown Subject", color: TKColor.black),
    ↪ TKDocument(name: "nf", deadline: nil))
}
```

Um die Daten aus dem Singleton abzurufen, wurden verschiedenartige Getter und Setter Methodiken implementiert. Exemplarisch soll hier eine etwas komplexere Get-Methode dargestellt werden: Die Methode: `getSubjectAndDocumentForCollectionIndex()` liefert beispielsweise für die *CardViews* im Schülerhauptmenü das benötigte *TKSubject* & *TKDocument* für die View.

6.2.2.1 Abrufen aus der Schnittstelle mit Operations

Da die Schnittstelle die Daten mit vier verschiedenen verschachtelten Datentypen bereitstellt, soll im Folgenden dargestellt werden, wie die Schnittstellendaten mithilfe von Operations abgerufen werden können.

```
func fetchAll(notificationName : Notification.Name? = nil, rank : TKRank) {
//    1: Zuruecksetzen der des Controllers und initialisieren der
    ↪ Operations
    resetWithRank(newRank: rank)
    let classesOperation      = ClassOperation(opRank: self.getRank())
    let subjectOperation      = SubjectOperation(opRank: self.getRank())
    let documentOperation     = DocumentOperation(opRank: self.getRank())
    let exerciseOperation    = ExerciseOperation(opRank: self.getRank())
    var subjects              = [TKSubject]()
//    2: Definieren der Completion Blocks nach der Operations
```

```

    classesOperation.completionBlock = {
        subjectOperation.classes = TKModelSingleton.sharedInstance.
    ↵ downloadedClasses
    }

    subjectOperation.completionBlock = {
        if self.getRank() == TKRank.teacher {
            for element in TKModelSingleton.sharedInstance.
    ↵ downloadedClasses {
                subjects.append(contentsOf: element.subjects)
            }
        }
        else if self.getRank() == TKRank.student{
            subjects = TKModelSingleton.sharedInstance.downloadedSubjects
        }
        documentOperation.subjects = subjects

        self.debugPrintAfterFetch()
    }

    documentOperation.completionBlock = {
        if self.model.myTKRank == TKRank.teacher {
            subjects = []
            for element in TKModelSingleton.sharedInstance.
    ↵ downloadedClasses {
                subjects.append(contentsOf: element.subjects)
            }
        }
        else if self.model.myTKRank == TKRank.student {
            subjects = self.model.downloadedSubjects
        }

        for element in subjects {
            exerciseOperation.documents.append(contentsOf: element.
    ↵ documents)
        }
    }

    exerciseOperation.completionBlock = {
        if let notificationName = notificationName {
            if exerciseOperation.isInitialized == false {
                DispatchQueue.main.async {
                    NotificationCenter.default.post(Notification(name:
    ↵ notificationName))
                }
            }
            else {
                print("Completion Block")
            }
        }
    }
}

```

```

        self.debugPrintAfterFetch()
        DispatchQueue.main.async {
            NotificationCenter.default.post(Notification(name:
        ↳ notificationName))
        }
    }
}

// 3: Setzen der Operation Dependencies und aufsetzen der Queue
subjectOperation.addDependency(classesOperation)
documentOperation.addDependency(subjectOperation)
exerciseOperation.addDependency(documentOperation)

let queue = OperationQueue()
queue.maxConcurrentOperationCount = 1
queue.addOperations([classesOperation, subjectOperation,
    ↳ documentOperation, exerciseOperation], waitUntilFinished: false)

}

```

1: Zurücksetzen des Controllers und Initialisieren der Operations Bevor die Operations initialisiert werden können, werden mit der Methode `resetWithRank()` die in dem `TKModelSingleton` gehaltenen Variablen zurückgesetzt und der `TKRank` mit den neuen `TKRank` aktualisiert. Danach werden alle Operations mit den neuen `TKRank` initialisiert.

2: Definieren der Completion Blocks nach der Operations Anschließend werden die Completion Blocks der Operations definiert. Hierbei werden die jeweiligen heruntergeladenen Objekte von der Schnittstelle in die darauf folgende Operation eingefügt. In dem `Completion Block` der Exercise Operation wird eine Notification an die `UIThreads` geschickt, welche anschließend die neuen von der Schnittstelle abgerufenen Daten abrufen können.

3: Setzen der Operation Dependencies und Aufsetzen der Queue Hier werden die Dependencies (Abhängigkeiten) der Operations zueinander gesetzt. Es muss immer die darüberliegende Operation den `Completion Block` abgehandelt haben, bevor die darunterliegende Operation beginnen kann. Anschließend werden die Operations mithilfe einer Queue in die richtige Reihenfolge gebracht.

Weitere Informationen über die Funktionsweise von den Operations ist in dem Lehrerteil dieser Dokumentation enthalten.

7 Schnittstelle iCloud

Patrick Niepel, Marcel Hagmann, Carl Philipp Knoblauch

7.1 Einleitung

In diesem Abschnitt wird die Schnittstelle mit iCloud beschrieben. Dabei wird erklärt wie die Architektur aufgebaut ist, wie mit der Schnittstelle kommuniziert wird und welche Probleme aufgetreten sind.

7.2 Warum iCloud/CloudKit?

Wenn man bei der Entwicklung einer iOS App auf Cloud Services zurückgreifen will, bietet sich natürlich das Apple eigene CloudKit für iCloud an. Es ergab sich dadurch auch die Möglichkeit eine neues Framework kennenzulernen, da wir zuvor noch nicht mit CloudKit gearbeitet hatten. Über die Cloud-Schnittstelle sollen zwischen Lehrer und Schüler alle Aufgaben geteilt werden. Der Lehrer kann seine Aufgaben/Spiele in iCloud laden und diese mit seinen Schülern teilen. Die Schüler sollen dann diese Aufgaben/Spiele erledigen und ihre Lösungen wieder in iCloud laden. Dadurch wird dem Lehrer wiederum ermöglicht sich einen Überblick über die Lösungen seiner Klasse zu machen. Mit CloudKit konnten wir diese Ziele alle umsetzen.

7.3 Architektur

Auch TeachKit ist strikt nach der Model-View-Controller - Architektur aufgebaut. Hierbei erben alle Models, die in der Cloud gespeichert werden, von ihrer Superklasse TKCloudObject. Die Controller die diese Models verwalten, sind mit Hilfe eines generischen Controllers TKGenericCloudController implementiert worden. Alle Cloud-Models und Controller bedienen sich von verschiedenen Enumerations, die die Funktionalität übersichtlicher gestalten.

7 Schnittstelle iCloud

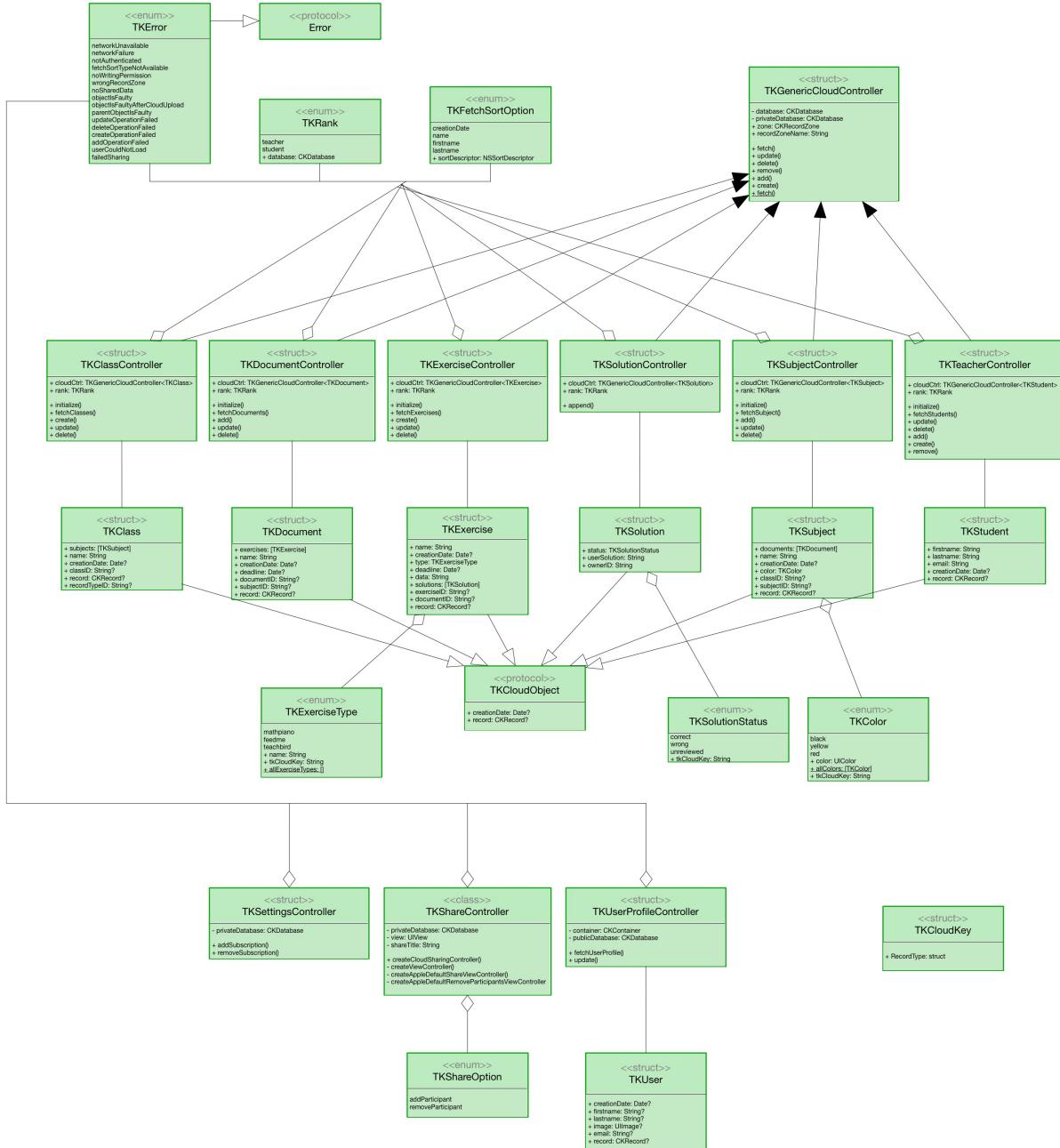


Abbildung 7.1: TeachKit Schnittstellen Klassendiagramm

7.4 Features

7.4.1 Upload/Download/Delete/Fetch/Update

Für jeden Datentyp in TeachKit, gibt es einen Controller der für die Operationen auf den Datentyp verantwortlich ist. Somit gibt es folgende Controller für die Datentypen:

- TKClassController
- TKSubjectController
- TKDocumentController
- TKEExerciseController
- TKSolutionController

Alle Controller sind ähnlich aufgebaut. Der Grund weshalb der Controller über die Methode initialize(...) funktionsfähig gemacht werden muss ist der, dass der Student auf die Shared-Database zugreift und diese zuerst gefetched werden muss.

Um doppelten Code zu vermeiden, arbeiten alle der oben genannten Controller mit dem TKGenericCloudController, der die Grundfunktionen übernimmt und in den jeweiligen Controllern dann spezialisiert werden.

7.4.2 User Profile

Zu jedem Nutzer kann der Vorname, Nachname und ein Profilbild gespeichert werden. Der TKUserController der für die Nutzerverwaltung verantwortlich ist, arbeitet auf der Public-Database. Das bedeutet, dass alle Nutzer diese Informationen sehen können. Die aktuelle implementation erlaubt nur den download der Daten für den derzeit eingeloggten Nutzer. Diese kann erweitert werden, hätte aber momentan keine Verwendung gefunden.

7.4.3 Sharing

Eines der wichtigsten Features unserer App ist das Teilen von Daten zwischen Teacher und Student. Nach dem Teilen gemeinsamer Daten haben beide Zugriff auf das Subject und alles was darunter angelegt ist.

CloudKit arbeitet mit drei verschiedenen Datenbanken.

Private-Database

Der aktuell angemeldete Nutzer ist der Inhaber der Daten, nur dieser hat Zugriff auf diese Datenbank und hat das Recht zu lesen und zu schreiben.

Shared-Database

Der aktuelle Nutzer ist nicht der Besitzer der geteilten Daten, und hat die beim Teilen zugewiesenen lese und/oder schreib Rechte.

Public-Database

Der aktuelle Nutzer ist nicht der Besitzer der geteilten Daten, und hat die beim Teilen Jeder App Nutzer hat lese Recht auf diese Daten, auch ohne aktiven iCloud Account.

Legt der Teacher seine Daten an, befinden diese sich in seiner private-Database. Nachdem dieser das Subject geteilt hat, befindet sich dies immer noch in der private-Database. Bei jedem Student der nun berechtigt ist, das geteilte Subject einzusehen, wird eine Referenz in der shared-Database gespeichert. Diese Referenz zeigt auf die Daten des Teachers in der private-Database.

Der Teacher teilt mit dem Student ein Subject, damit der Student die neuen Arbeitsblätter einsehen kann. Mit dem Student wird der Zugriff auf Class nicht geteilt, weil er diese Information nicht benötigt.

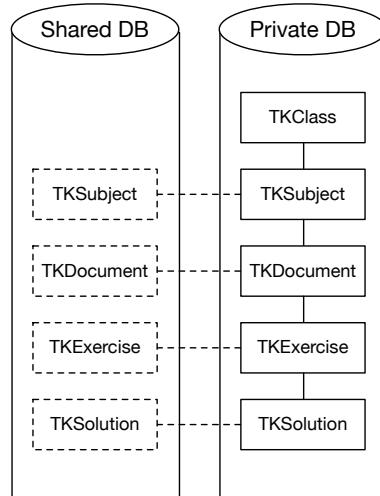


Abbildung 7.2: hared DB / Private DB

Das Teilen findet über den TKShareController statt. Mit der Methode `createCloudSharingController(...)`, wird der ViewController der für das Teilen verantwortlich ist erstellt und kann angezeigt werden. Bei der Verwendung des Controllers müssen keine weiteren Bedingungen beachtet werden, der Controller kümmert sich um alles was zum Teilen benötigt wird.

7.4.4 Push/Subscriptions

Eine Besonderheit des CloudKits ist die einfache Implementierung von Push Benachrichtigungen bei Datenbankänderungen. PushNotifications in einer App zu implementieren, die über einen eigenen Server laufen, setzen PHP Kenntnisse voraus und können einiges an Zeit in Anspruch nehmen. CloudKit Subscriptions sind jedoch einfacher zu implementieren und können auf Insert, Update, Delete und Create in der Datenbank reagieren.

Der Verwendungszweck der Subscriptions war dafür gedacht, dass der Student über Änderungen zu seinem Fach informiert wird. Dies könnte beispielsweise der Fall sein, wenn der Professor ein neues Arbeitsblatt seinem Fach hinzufügt.

Wir implementierten die CloudKit Subscriptions, die darauf reagiert, wenn der Teacher ein neues Document einem Subject hinzufügt. Die ersten Tests auf zwei unterschiedlichen Geräten mit der gleichen Apple ID waren erfolgreich. Beim Testen über zwei unterschiedliche Apple ID's, bei dem der Teacher ein Subject mit dem Student teilt, kamen wir allerdings an unsere Grenzen. Der Student wurde beim Hinzufügen eines neuen Objekts nicht benachrichtigt. In der CloudKit API wurden wir dann auf die folgende Anmerkung aufmerksam, die besagt, dass Subscriptions auf der Shared-Database nicht unterstützt werden.

7.4.5 TKError

Bei den meisten Aktionen die innerhalb des TeachKits Fehler auslösen können, werden Fehler vom Datentyp TKError erstellt. Jeder Fehler beinhaltet eine genauere Beschreibung des aufgetretene Problems. Folgende Fehler existieren:

networkUnavailable Es besteht keine Internetverbindung.

networkFailure Es besteht eine Internetverbindung, es konnte allerdings keine Verbindung zur Cloud hergestellt werden.

wrongRecordZone Die Operation wird auf der falschen RecordZone ausgeführt oder existiert nicht. An Schnittstelle wenden.

failedSharing Das Objekt konnte nicht geteilt werden.

parentObjectIsFaulty Operation konnte nicht ausgeführt werden, da das Parent-Objekt Fehlerhaft ist.

objectIsFaulty Operation konnte nicht ausgeführt werden, da das Objekt Fehlerhaft ist.

objectIsFaultyAfterCloudUpload Auf dem Objekt wurde eine Operation in der Cloud ausgeführt. Das von der Cloud erhaltene Objekt ist inkonsistent.

userCouldNotLoad Beim Zugriff auf die Nutzer Informationen ist ein Fehler unterlaufen.

updateOperationFailed Der Datensatz konnte nicht geupdated werden.

deleteOperationFailed Der Datensatz konnte nicht gelöscht werden.

createOperationFailed Der Datensatz konnte nicht erstellt werden.

addOperationFailed Der Datensatz konnte nicht hinzugefügt werden.

fetchSortTypeNotAvailable Das Attribut nach dem sortiert werden soll existiert nicht.

noWritePermission Der Nutzer hat keine Berechtigung die Daten zu ändern.

noSharedData Die Operation konnte nicht ausgeführt werden, da noch keine Daten geteilt werden.

7.5 Aufgetretene Probleme

7.5.1 Subscription

Da auf das Problem mit den Subscriptions bereits im Abschnitt Push/Subscriptions eingegangen wurde, erwähnen wir an dieser Stelle nur noch einmal, dass Subscriptions nicht auf der Shared-Database unterstützt werden.

7.5.2 Sharing

Während der Implementation der Sharing Funktion, hatten wir über einen längeren Zeitraum das Problem, dass das Record zwischen den Nutzern geteilt wurde. Der darunter hängende Baum war allerdings nicht beim Student einzusehen.

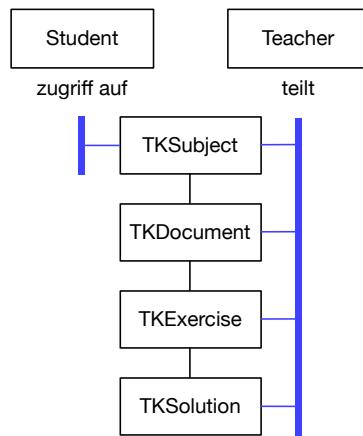


Abbildung 7.3: Zugriff Teacher/Student

Letztendlich stellte sich heraus, dass die Verbindung zwischen Parent- und dem Child-Record nicht hergestellt wurde. Dieses Problem konnte ganz einfach mit der Methode `setParent(CKRecord?)` gelöst werden, allerdings musste man dafür erst wissen, dass so eine Funktion überhaupt existiert.

7.5.3 TKSolution

Die Idee hinter TKSolution war die, dass ein Student seine Lösung darin erstellt und anschließend zu der TKEexercise hinzufügt. Allerdings ist es dem Student nicht möglich, weitere Records in der geteilten Datenbank des Teachers zu erstellen und hinzuzufügen.

```
TKGenericCloudController-create-Error: Optional<CKError 0x103781b20: "Permission Failure" (10/2007); server message = "CREATE operation not permitted"; uuid = 6A7512C1-4170-4D82-A2BF-23564DADBCA7; container ID = "icloud.iosapps.hof-university.teachify">
solution upload error: Optional(TEachify_TKError.objectIsFaultyAfterCloudUpload)
```

Abbildung 7.4: Error

Da diese Fehlermeldung auftritt obwohl wir die publicPermission des CKShare Objektes richtig gesetzt haben und immer noch kein Record erzeugen konnten, mussten wir eine andere Lösung dafür suchen. Wir entschlossen uns die Lösungen in ein Data-Objekt zu serialisieren und diese in TKEexercise hinzuzufügen. Unseren neuen Lösungsansatz testen wir zuerst mit einem Attribut vom Datentyp String und nicht den eigentlich benötigten Datentyp Data. Beim ersten Upload in die iCloud werden die Records in der Cloud automatisch angelegt. Der erste Test hat geklappt und anschließend wollten wir unseren benötigten Daten vom Typ Data sichern. Da allerdings die angelegten Constraints nicht mehr für den neuen Datentyp gestimmt haben, konnten keine Records mehr hochgeladen werden. Die Constraints müssen zuerst im iCloud-Dashboard gelöscht und erneut hochgeladen werden.

Abbildungsverzeichnis

2.2	Das Schülerhauptmenü (Schüler)	7
2.3	Detailansicht einer Übungsaufgabe (Schüler)	8
2.4	Das Mathe Piano Spiel (Schüler)	11
2.1	Der Login Bildschirm (Schüler)	13
3.1	Teachify Bird Spiel StartScreen	14
3.2	Teachify Bird Spiel	15
4.1	Erster Entwurf	17
4.2	Game Screens	18
4.3	Die Touch-Events	20
4.4	Drachen Assets	20
7.1	TeachKit Schnittstellen Klassendiagramm	31
7.2	hared DB / Private DB	33
7.3	Zugriff Teacher/Student	36
7.4	Error	37