ulm university universität ulm

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Diplomarbeit
im Studiengang Informatik

Technische Konzeption und Realisierung einer mobilen Anwendung zur Durchführung von Lernübungen anatomischer Strukturen am Beispiel des Apple iOS

vorgelegt von

Daniel Kopf

April 2013

Erstgutachter	Prof. Dr. Manfred Reichert
Zweitgutachter	Prof. Dr. Peter Dadam
Betreuer	Dipl. Inf. Rüdiger Pryss
Arbeit vorgelegt am	04. April 2013

Danksagung

Ich möchte mich an erster Stelle bei Rüdiger Pryss für die tatkräftige Unterstützung bei der Erstellung meiner Diplomarbeit bedanken.
Darüber hinaus bedanke ich mich bei Herrn Professor Reichert und Herrn Professor Dadam für die schnelle Zusage, mir als Gutachter zur Verfügung zu stehen.
Ganz besonders Bedanken möchte ich mich für die Bereitstellung der für die Entwicklung benötigten Apple Geräte durch das Institut für Datenbanken und Informationssysteme und das Institut für Anatomie und Zellbiologie.
Weiterhin möchte ich mich bei Barbara Eichner bedanken, da Sie mir bei der Erstellung der Anforderungen für das System sehr geholfen hat.
Nicht zuletzt möchte ich mich auch bei meinen Eltern bedanken, denn ohne sie wäre dieses Studium niemals möglich gewesen.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sinngemäße Übernahmen aus anderen Werken sind als solche kenntlich gemacht und mit genauer Quellenangabe (auch aus elektronischen Medien) versehen.

Ulm, den 04. April 2013 Daniel Kopf

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation . . . 2
1.2 Aufbau der Arbeit . . . 2

2 Grundlagen **5**
2.1 Präpmappe . . . 5
2.1.1 Vorstellung der Präpmappe . . . 5
2.2 E-Learning und M-Learning . . . 10
2.2.1 3D4Medical Skeleton System Pro III . . . 11
2.2.2 Science360 for iPad . . . 12
2.2.3 Snapguide . . . 13
2.2.4 Vergleich der vorgestellten Apps . . . 14

3 Anforderungsanalyse **17**
3.1 Anforderungen der beteiligten Benutzergruppen . . . 17
3.1.1 Administratoren . . . 18
3.1.2 Student . . . 19
3.2 Weitere Anforderungen . . . 20
3.3 Anforderungen an die Hardware . . . 22
3.4 Zusammenfassung . . . 22

4 Entwurf **25**
4.1 Informationsarchitektur . . . 26
4.2 Screens . . . 28
4.2.1 Anmeldung . . . 28
4.2.2 Benutzersichten . . . 29
4.2.2.1 Studentensicht . . . 31
4.2.2.2 Administratorsicht . . . 32
4.3 Zusammenfassung . . . 34

5 Anwenderszenarien **35**
5.1 Szenario 1: Administrator erstellt eine Übung . . . 35
5.1.1 Login . . . 35
5.1.2 Administratorsicht: Testate . . . 36
5.1.3 Administratorsicht: Übungen . . . 37
5.1.4 Administratorsicht: Werkzeuge . . . 37
5.1.5 Administratorsicht: Labels . . . 38

5.2 Szenario 2: Student führt eine Übung durch . . . 39
5.2.1 Login . . . 39
5.2.2 Studentensicht: Testate . . . 40
5.2.3 Studentensicht: Übungen . . . 40
5.2.4 Studentensicht: Pins . . . 40
5.3 Zusammenfassung . . . 43

6 Implementierung **45**
6.1 Grundlagen der iOS-Programmierung . . . 45
6.1.1 Xcode . . . 45
6.1.1.1 Interface Builder . . . 45
6.1.1.2 Instruments . . . 46
6.1.1.3 Simulator . . . 46
6.1.2 iOS SDK . . . 46
6.1.2.1 Core OS . . . 46
6.1.2.2 Core Services . . . 46
6.1.2.3 Media . . . 47
6.1.2.4 Cocoa Touch . . . 48
6.1.3 Automatic Reference Counting . . . 48
6.1.4 Design Patterns . . . 50
6.1.4.1 Model-View-Controller . . . 50
6.1.4.2 Delegation . . . 50
6.1.4.3 Target-Action . . . 51
6.1.4.4 Observer-Pattern . . . 51
6.2 Besonderheiten der iPad-Programmierung . . . 51
6.2.1 UIPopoverController . . . 52
6.2.2 UISplitViewController . . . 52
6.2.3 Modale Sichten . . . 53
6.3 Architektur im Überblick . . . 56
6.4 Verwendete APIs, Frameworks und Libraries . . . 58
6.4.1 AFNetworking . . . 58
6.4.2 Core Data . . . 59
6.4.3 QuartzCore . . . 59
6.4.4 Core Graphics . . . 60
6.5 Web Service . . . 60
6.6 Datenmodell . . . 62
6.7 Synchronisierung der Daten . . . 63
6.8 Die grafische Benutzeroberfläche . . . 64
6.8.1 PASplitViewController . . . 65
6.8.2 LoginViewController . . . 65
6.8.3 Studentensicht . . . 67
6.8.4 Administatorsicht . . . 72
6.8.4.1 Bild auswählen/ Bild aufnehmen . . . 74
6.8.4.2 Werkzeuge . . . 75
6.8.4.3 Pins . . . 78
6.9 Zusammenfassung . . . 80

7 Diskussion **83**

8 Zusammenfassung **87**
8.1 Ausblick 87
8.2 Fazit 89

1 Einleitung

Der anfängliche Ansatz, um E-Learning in Hochschulen zu etablieren, bestand darin, die Präsenzlehre durch sogenannte virtuelle Universitäten zu ersetzen. Jedoch stellte sich bald heraus, dass dies in der Realität nicht umzusetzen ist. Aus diesem Grund verlagerte sich der Fokus von E-Learning immer mehr auf die Unterstützung der Präsenzlehre, anstatt diese zu ersetzen [31]. So werden auch mehrere Veranstaltungen an der medizinischen Fakultät der Universität Ulm durch digitale Inhalte ergänzt, die Studenten online zur Verfügung gestellt werden. Die Inhalte werden dabei in Kooperation mit dem Kompetenzzentrum eLearning über deren Plattform "Moodle" angeboten.
Eine der Veranstaltungen, bei der ergänzende E-Learning Inhalte für Studenten angeboten werden, ist der Präparierkurs der makroskopischen Anatomie. Er ist Teil der vorklinischen ärztlichen Ausbildung und wird deshalb auch an der medizinischen Fakultät der Universität Ulm durch das Institut für Anatomie und Zellbiologie angeboten. In ihm wird den Studenten ein umfangreiches Wissen im Bereich der makroskopischen Anatomie vermittelt. Dies geschieht dadurch, dass direkt mit anatomischen Strukturen gearbeitet wird, indem konservierte Leichen seziert und präpariert werden.
Um sich darauf vorzubereiten, wird ein System angeboten, dass mehrere digitale Testate enthält, durch die die verschiedene Körperregionen und deren anatomische Strukturen in mehreren Übungen erlernt werden können. Dieses System ist allerdings nicht sehr ausgereift und birgt einige Nachteile für die Studenten. So bleibt es beispielsweise dem Studenten selbst überlassen, in welcher Form er die Testate bearbeitet, da er die Strukturen nicht innerhalb des Systems beschriften, beziehungsweise auswerten kann. Informationssysteme sind generell sehr komplex, da sie einerseits aus einer großen Anzahl an Elementen bestehen und diese andererseits auf vielfältige Art und Weise zusammenhängen [30]. So auch das mobile Informationssystem, das zur Ergänzung des bestehenden Systems entwickelt wurde und dessen Nachteile beheben soll.

1.1 Motivation

Die Anwendung des bestehenden Systems geschieht mithilfe eines Webbrowsers. Da dieses allerdings nicht auf mobile Geräte beziehungsweise mobile Webbrowser angepasst ist, ist bisher von einer mobilen Nutzung eher abzuraten. Durch die Einführung einer Anwendung für mobile Geräte (App) soll diesem Umstand entgegengewirkt werden. Ein Vorteil einer solchen App ist, dass die Studenten die Vorbereitung auf den Kurs besser in ihren Alltag integrieren können. Sie sind viel flexibler, wenn es darum geht, wo und wann sie sich vorbereiten wollen und können beispielsweise während einer Zugfahrt Übungen bearbeiten. Dabei soll das neue System die Funktionalität des bestehenden Systems bieten und diese weiter optimieren. Ein möglicher Optimierungsansatz, welcher nach Analyse des alten Systems erkannt wurde, ist, dass eine Unterstützung bei der Auswertung von Übungen benötigt wird. Zudem wäre es auch wünschenswert neue Inhalte einpflegen zu können. Diese Funktion wird bisher nicht direkt durch das System abgedeckt und soll in der App eingeführt werden. Als Plattform wurde das iPad von Apple gewählt, da es alle nötigen Voraussetzungen erfüllt, um eine adäquate Anwendung zur Vermittlung der Lehrinhalte zu realisieren, und zudem eine Anschaffung mehrerer solcher Geräte durch die medizinische Fakultät geplant war.

1.2 Aufbau der Arbeit

Im nachfolgenden Kapitel wird der Leser mit den Grundlagen vertraut gemacht, welche zum Verständnis dieser Arbeit benötigt werden. Dafür wird zuerst das bisher bestehende System der "Präpmappe" genauer betrachtet. Danach erfolgt ein Überblick über das Themengebiet des M-Learnings beziehungsweise E-Learnings und es werden drei themenverwandte Apps vorgestellt. Am Ende des Kapitels werden diese anhand von von M-Learning Kriterien verglichen.
Im dritten Kapitel werdend die Anforderungen, welche an das System gestellt werden, analysiert. Dabei werden zuerst verschiedene Benutzergruppen definiert, die anfänglichen Anforderungen, welche sich durch ein Interview und dem bestehenden System ergaben, formuliert und schließlich weitere Anforderungen, welche während der Entwicklung hinzukamen. Abschließend werden die Anforderungen an die Hardware festgelegt.
Das vierte Kapitel beschäftigt sich mit dem Entwurf der App. Zunächst wird der Leser mit der Informationsarchitektur der App vertraut gemacht. Darüber hinaus werden die verschiedenen Entwürfe der Benutzeroberfläche im Detail vorgestellt.
Das fünfte Kapitel führt durch die App anhand von Anwenderszenarien. Dabei werden zwei Szenarien definiert und genauer betrachtet. Am Ende des Kapitels erfolgt eine Zusammenfassung in der weitere mögliche Szenarien vorgestellt werden.
Die Implementierung der App wird in Kapitel sechs behandelt. Vorab werden die Grundlagen für die Entwicklung in iOS vermittelt und häufig genutzte Entwurfsmuster vorgestellt. Danach werden die Besonderheiten bei der Entwicklung auf einem iPad beschrieben. Des Weiteren wird das Zusammenspiel der verschiedenen Komponenten anhand der Softwarearchitektur verdeutlicht. Im Anschluss werden die verwendeten Frameworks und Bibliotheken skizziert und die einzelnen Komponenten im Detail vorgestellt.
Im nächsten Kapitel erfolgt eine Diskussion über die Anforderungen und deren Realisierung im System.
Am Schluss der Arbeit erfolgt ein Ausblick auf mögliche Erweiterungen des Systems und es wird ein

Fazit gezogen. In Abbildung 1.1 ist eine grafische Übersicht sämtlicher Kapitel und deren Inhalt zu sehen.

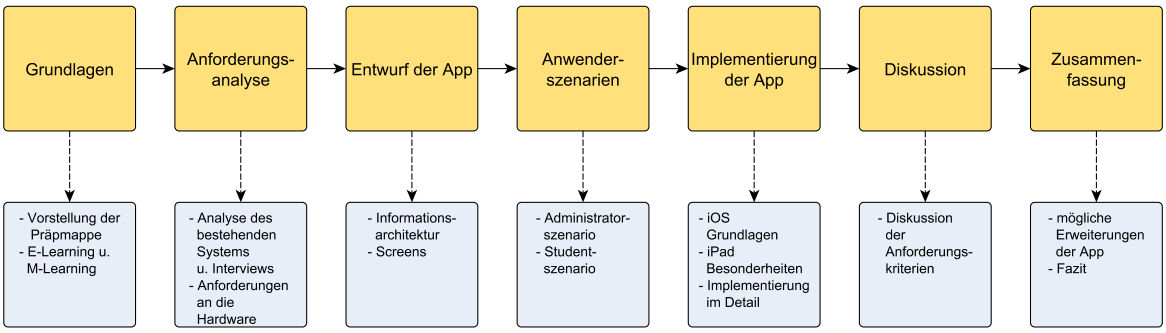


Abbildung 1.1: Übersicht über die Arbeit

2 Grundlagen

Im Verlauf dieses Kapitels wird das bestehende System zur Vorbereitung ("Präpmappe") vorgestellt. Anschließend wird der Leser an das Themenfeld des *E-Learnings* beziehungsweise des *M-Learnings* herangeführt. Um dies an praktischen Beispielen zu verdeutlichen, werden weiterhin drei mobile Anwendungen vorgestellt und abschließend, anhand der Definitionskriterien für *M-Learning*, miteinander verglichen.

2.1 Präpmappe

Die Präpmappe ist eine Sammlung von digitalen Testaten, die für Studenten an der medizinischen Fakultät Ulm verfügbar ist. Sie wird zum Zweck der Vorbereitung in der Veranstaltung *Makroskopische Anatomie (Präparierkurs)* angeboten. Der Zugang erfolgt über die Lernplattform "Moodle", welche von der medizinischen Fakultät der Universität Ulm in Kooperation mit dem Kompetenzzentrum eLearning verwaltet wird. Die Plattform wird in Abbildung 2.1 gezeigt.

2.1.1 Vorstellung der Präpmappe

Um Zugriff auf die Präpmappe zu erhalten, muss ein Student zunächst über gültige Benutzerinformationen verfügen, um sich bei der Lernplattform "Moodle" anzumelden. Weiterhin muss er sich dort für die Veranstaltung *"Makroskopische Anatomie (Präparierkurs)"* anmelden. Befindet er sich auf der Seite des Kurses, kann er dort im Bereich E-Learning die Präpmappe durch Klicken des entsprechenden Links aufrufen. Zum weiteren Schutz des in der Präpmappe eingesetzten Bildmaterials, ist diese mit einem Passwort versehen, welches man von den Organisatoren der Veranstaltung erhält. Abbildung 2.2 zeigt die Ansicht, welche dem Studenten nach Starten der Präpmappe geboten wird.
Auf der linken Seite der Ansicht befindet sich ein Inhaltsverzeichnis, um sich durch die verschiedenen Übungen navigieren zu können. Jede Übung wird dabei einem Testat zugeordnet (beispielsweise

Abbildung 2.1: Die Moodle-Lernplattform der Universität Ulm [37]

befindet sich die Übung "CT Abdomen" in dem Testat "3.Testat: Brust- und Bauchsitus, Retrositus, kleines Becken").
Auf der rechten Seite der Ansicht wird die ausgewählte Übung angezeigt. Es handelt sich dabei entweder um ein Bild (beispielsweise eine Röntgenaufnahme eines Schultergelenkes), bei dem anatomische Strukturen benannt werden sollen, oder das Bild ist bereits komplett beschriftet.

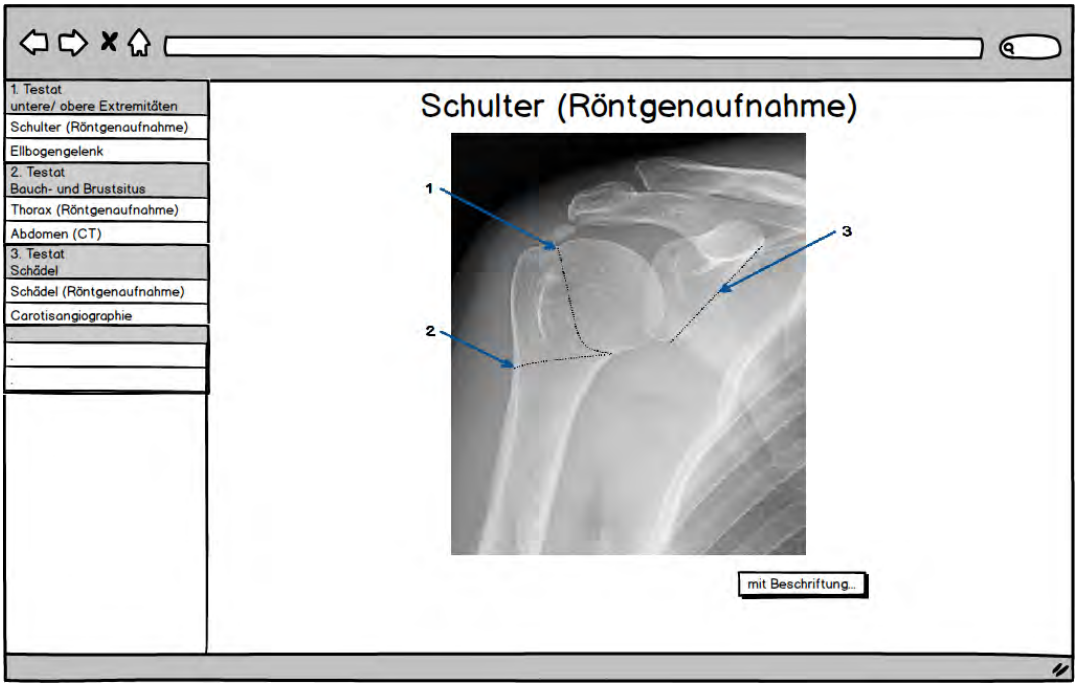


Abbildung 2.2: Die Ansicht nach Öffnen der Präpmappe [27]

Um eine Grundlage für die Entwicklung eines neuen Systems zu schaffen, wurde das bisherige System analysiert. Dafür wurden auch die Mittel betrachtet, die im bestehenden System eingesetzt werden, um die Lehrinhalte zu kennzeichnen. Diese werden im Folgenden genauer beschrieben:

- *Pfeile und Linien*
  Pfeile und Linien werden zum direkten Zeigen auf bestimmte Bildbereiche benutzt oder auch um nah beieinanderliegende anatomische Strukturen zu markieren, ohne dabei die wichtigen Details des Bildes zu verdecken (siehe Abbildung 2.3).

Abbildung 2.3: Übung mit Pfeilen und Linien [27]

- *Asterisk*
  Sie werden zur Markierung einer ergänzenden Information eingesetzt 2.4.
- *Ziffern*
  Ziffern werden zur Markierung großer beziehungsweise detailliert dargestellter anatomischer Strukturen benutzt. In den Übungen werden sowohl Buchstaben als auch Zahlen verwendet, welche bei der beschrifteten Version der Übung als Legende dienen (siehe Abbildung 2.5).
- *Gestrichelte Linien*
  Gestrichelte Linien werden benutzt, um zu vermitteln, dass ein nicht direkt sichtbarer Teil des Bildes beschriftet werden soll (beispielsweise die Bezeichnung der Rückseite eines Knochens, wie in Abbildung 2.6 dargestellt wird)

Die unbeschrifteten Übungen liegen im PDF-Format vor. Bei diesen hat der Student die Möglichkeit, durch Betätigen eines Buttons, eine neue PDF-Datei zu laden, welche die Beschriftungen enthält. Die

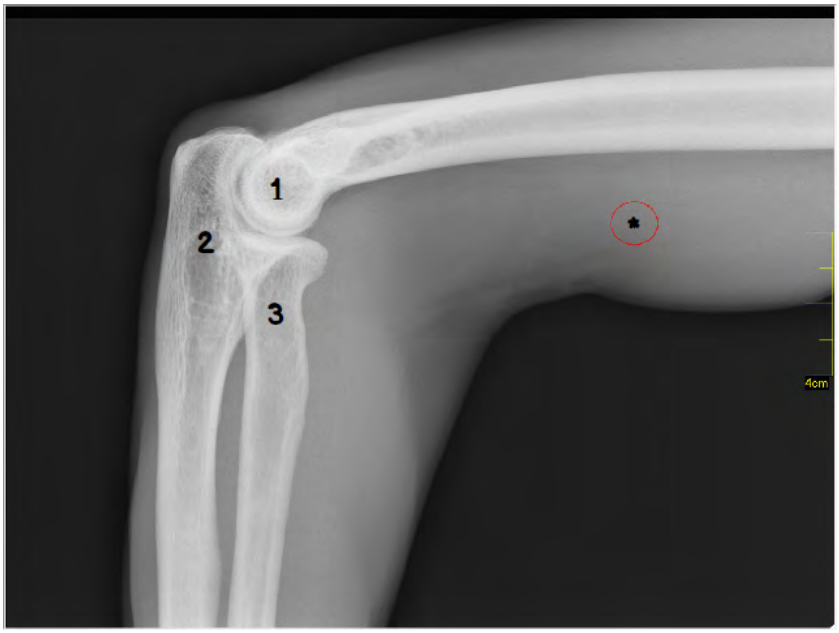


Abbildung 2.4: Asterisk zur Markierung ergänzender Informationen [27]

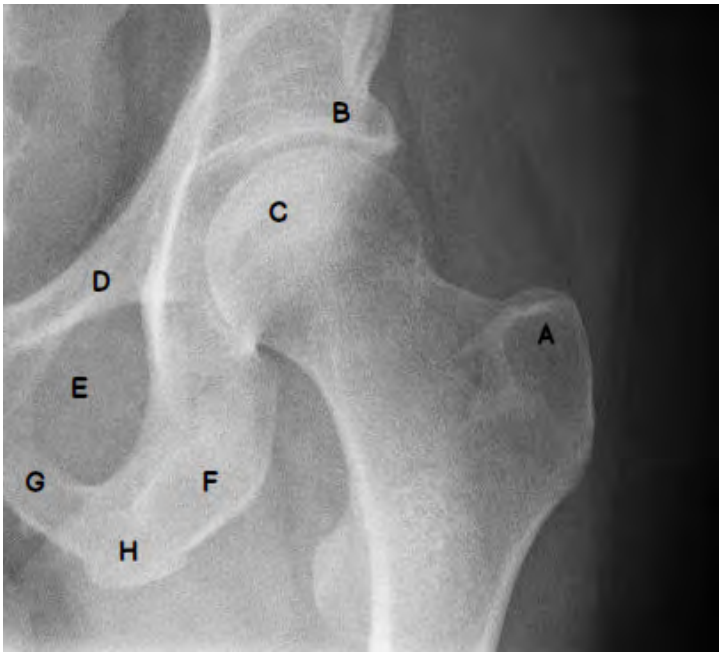


Abbildung 2.5: Beschriftung einer Übung mit Ziffern

Verbindung zwischen unbeschrifteten und beschrifteten Übungen wird in Abbildung 2.7 dargestellt.

Zudem ist in Testat fünf eine gemeinsame Übung zu Horizontal-, Frontal- und Sagitalschnittbildern des Gehirns aufgelistet , welche nach Auswahl in Form eines HTML-Dokuments in einem eigenen

Abbildung 2.6: Gestrichelte Linien deuten auf nicht direkt sichtbare Strukturen

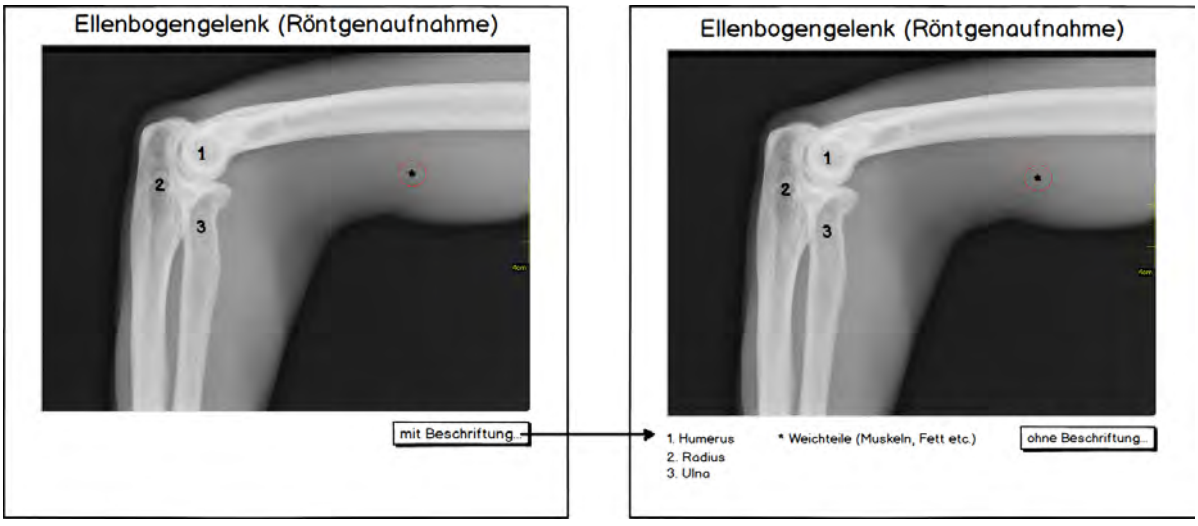


Abbildung 2.7: Verbindung zwischen unbeschrifteten und beschrifteten Übungen

Fenster präsentiert wird. Die verschiedenen Schnitte werden durch Klicken von Buttons im oberen Bereich des Fensters aufgerufen.

2.2 E-Learning und M-Learning

Zwar werden die Begriffe *E-Learning* (Electronic Learning) und *M-Learning* (Mobile Learning) heute in vielen verschiedensten Zusammenhängen benutzt, es existiert jedoch noch keine eindeutige Definition, welche auf die Begriffe angewandt werden könnte. Eine mögliche Definition des E-Learnings lautet: „E-Learning kann begriffen werden als Lernen, das mit Informations- und Kommunikationstechnologien (Basis- und Lerntechnologien) respektive mit darauf aufbauenden (E-Learning-) Systemen unterstützt bzw. ermöglicht wird.“ [21]. Im Allgemeinen könnte man sagen, dass alle, durch Computer vermittelten Lerninhalte, zum Bereich des E-Learnings zählen.
M-Learning kann als konsequente Weiterentwicklung des klassischen E-Learning betrachtet werden, welche von Georgiev/Georgieva/Smrikarov [24] folgendermaßen definiert wird: "M-Learning must include the ability to learn everywhere at every time without permanent physical connection to cable networks". In dieser Betrachtung liegt der wesentliche Vorteil des mobilen Lernens darin, dass Lernen beispielsweise auch im Auto oder im Zug möglich ist.
Diese Definition beschränkt sich aber auf die Verwendung von mobilen Geräten. Aus diesem Grund ist eine genaue Abgrenzung zwischen M-Learning und E-Learning nicht möglich, wie in Abbildung 2.8 verdeutlicht wird. Um den Begriff genauer abzugrenzen, schlägt Traxler [36] vor, dass eine M-Learning-Anwendung die folgenden Charakteristiken aufweisen sollte:

- *Spontan* Das Lernen sollte ungeplant und unbewusst stattfinden.
- *Privat* Der Lernvorgang erfordert keinen expliziten Lehrer.
- *Tragbar* Das Gerät, welches die Lehrinhalte vermittelt, muss portabel sein.
- *Nahegelegen* Das Lernen findet im selben Kontext statt, in dem es angewandt wird.
- *Informell* Das Lernen sollte außerhalb des formellen Bildungsbereiches und in verschiedenen Lebenszusammenhängen stattfinden.
- *"Mundgerecht"* Es dürfen nicht zu viele Lehrinhalte auf einmal vermittelt werden.
- *Leicht anwendbar* Die Anwendungen müssen ohne Vorkenntnisse sofort nutzbar sein.
- *Kontextbezogen* Die Lernsoftware sollte beispielsweise Faktoren wie die Gemütslage des Anwenders oder die Umgebung, in der er sich befindet, miteinbeziehen (beispielsweise durch GPS oder Sensoren) und dazu passende Lehrinhalte vermitteln.

Weiter schreibt er, dass zukünftig auch folgende Aspekte beachtet werden müssten:

- *Verbunden* Es findet ein aktiver Wissensaustausch zwischen den Anwendern statt.
- *Personalisiert* Verschiedene Aspekte, wie beispielsweise das Befinden des Anwender, müssen miteinbezogen werden.
- *Interaktiv* Das gelernte kann praktisch (beispielsweise in interaktiven Übungen) angewandt werden.

Die meisten Anwendungen, die sich selbst unter der Kategorie des M-Learnings einordnen, erfüllen aber bei weitem nicht alle dieser Anforderungen.
Zur weiteren Erläuterung der Thematik werden nachfolgend verschiedene E-Learning beziehungsweise M-Learning Apps präsentiert.

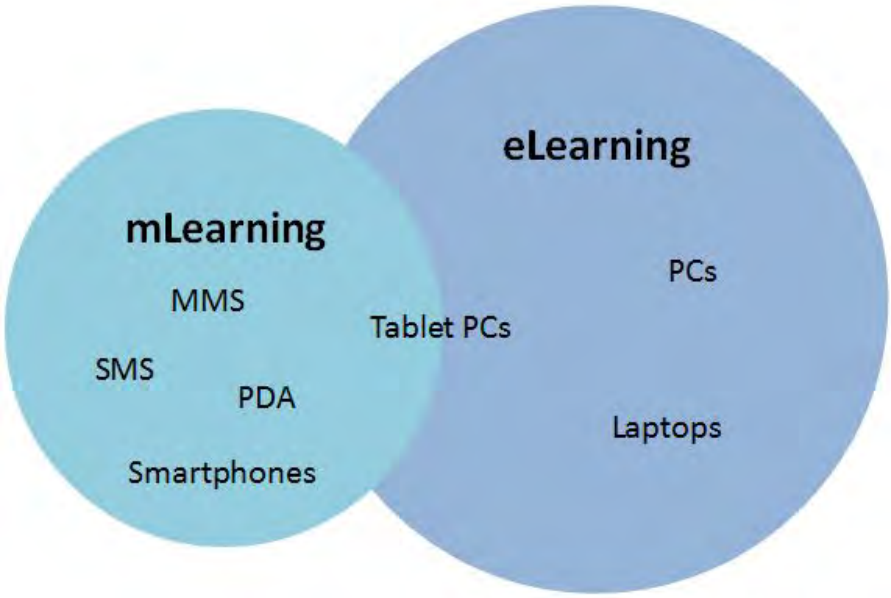


Abbildung 2.8: Problem der Abgrenzung zwischen M- und E-Learning [36]

2.2.1 3D4Medical Skeleton System Pro III

3D4Medical [1] ist ein Unternehmen, dass sich auf die Entwicklung von iOS-Apps in den Bereichen Medizin, Pädagogik und Gesundheit spezialisiert hat. Ihr Sortiment in dem Fachgebiet Medizin umfasst Apps, welche das Lernen von anatomischen Strukturen unterstützen. Eine dieser Apps ist das "Skeleton System Pro III", welches für das iPad, iPod Touch und das iPhone erhältlich ist. Im Folgenden wird die iPad-Version der App genauer betrachtet.
Nach Starten der App wird dem Benutzer das 3D-Modell eines Skelettes präsentiert. Zudem wird dem Benutzer eine Ansicht mit drei verschiedenen Reitern geboten, welche jeweils verschiedene Funktionen zur Navigation durch die einzelnen Bestandteile des Skelettes bieten.
Man hat außerdem Zugriff auf verschiedene Werkzeuge, mithilfe derer man beispielsweise am 3D-Modell einen Schnitt vornehmen kann, um so die Sicht zwischen *koronalen* (Ansicht von vorne), *sagittalen* (Ansicht von der Seite) und *transversalen* (Ansicht von oben) Schnitt zu wechseln. Darüber hinaus ist es möglich, die Bezeichnungen der anatomischen Strukturen einzublenden. Dies erfolgt, indem auf dem 3D-Modell Stecknadelköpfe angezeigt werden, welche bei Tippen die Bezeichnung des darunterliegenden Teils des Skelettes anzeigen. Zudem können bei manchen Strukturen Videos und Animationen - zum besseren Verständnis ihrer Funktion im Körper - angezeigt werden.
Zur verbesserten Aneignung des Lehrinhaltes wird ein Quiz angeboten. Nachdem der abzufragende Teil des Skelettes angegeben wurde, kann das Quiz in zwei verschiedenen Modi ausgeführt werden. Die erste Möglichkeit ist das Ziehen und Ablegen von Bezeichnungen über den entsprechenden Teilen des Skelettes. Bei dem zweiten Modus wird dem Benutzer ein markierter Teil des Skelettes angezeigt und er muss dafür aus mehreren Begriffen die richtige Bezeichnung auswählen.
Skeleton System Pro III bietet dem Benutzer darüber hinaus an, eigene Pins zu setzen und in das 3D-Modell zu zeichnen. Diese Ergänzungen können zudem über die gängigen Social Media Plattformen geteilt werden. Abbildung 2.9 zeigt die Benutzeroberfläche der App im Querformat.

Abbildung 2.9: Die Benutzeroberfläche der Skeleton System Pro III App [1]

2.2.2 Science360 for iPad

Die "National Science Foundation" (NSF) bietet mit ihrer App "Science360" Zugriff auf wissenschaftliche und technische Bilder und Videos. Die Lehrinhalte werden von der NSF und verschiedenen Wissenschaftlern beziehungsweise Universitäten bereitgestellt und decken verschiedene Bereiche der Naturwissenschaften ab. Zudem erhält man Neuigkeiten in Form der neuesten Forschungserkenntnisse, welche von den durch die NSF unterstützten Institutionen gesammelt werden.
Artikel mit Bildern sind dabei offline verfügbar, während Videoartikel über eine Internetverbindung gestreamt werden müssen. Zum Stöbern durch die Inhalte wird eine sogenannte "360 View" angeboten (siehe Abbildung 2.10). Um gezielt Inhalte zu finden, wird darüber hinaus eine Suchfunktion angeboten. Zudem können Bilder und Videos mithilfe der gängigen Social Media Plattformen (Twitter, Facebook) geteilt werden.
Der M-Learning-Ansatz besteht hier also darin, dass der Anwender sich jederzeit, auch ohne konstante Internetverbindung (außer für Videoartikel), durch die App neues Wissen aneignen kann.

Abbildung 2.10: "360 View" der Science360 App [25]

2.2.3 Snapguide

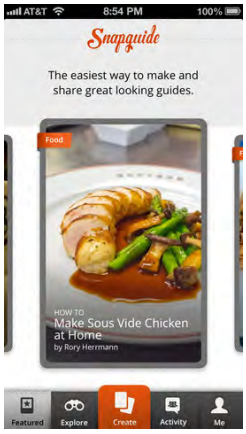


Abbildung 2.11: Übersicht von Snapguide [33]

Mit Snapguide hat der Anwender die Möglichkeit, Schritt-für-Schritt Anleitungen zu erstellen, und kann diese auch mit anderen Anwendern teilen. Das System besteht aus einem Web Service und einer App für alle mobilen Geräte, die mindestens iOS in der Version 5.0 verwenden.
Um die Anleitungen zu teilen, hat man die Möglichkeit, diese, via Web Service, auf einen Server hochzuladen. Dabei werden die Anleitungen in verschiedenen Kategorien, wie etwa Haustiere, Essen und Automobile zusammengefasst. Anleitungen können Texte, Fotos und Videos enthalten und es können zu jeder Anleitung Kommentare verfasst oder Fragen gestellt werden.
Diese App ist ein gutes Beispiel für M-Learning, da es die meisten Kriterien, um M-Learning von E-Learning abzugrenzen, erfüllt. So tauschen beispielsweise mehrere Benutzer ihr Wissen untereinander aus. Zudem ist sie mobil, leicht anwendbar, und das Lernen von neuen Inhalten findet spontan statt, indem der Benutzer beispielsweise durch die Kategorien stöbert und dabei auf etwas stößt, das ihn interessiert. Abbildung 2.11 zeigt die Benutzeroberfläche von Snapguide.

2.2.4 Vergleich der vorgestellten Apps

In der folgenden Tabelle 2.1 werden die vorgestellten Apps anhand der grundlegenden Kriterien, welche von Traxler [36] vorgeschlagen wurden, miteinander verglichen.

Kriterium	Skeleton System Pro III	Science360 for iPad	Snapguide
Spontan	✗	✓	✓
Privat	✓	✓	✓
Tragbar	✓	✓	✓
Nahegelegen	✗	✗	✓
Informell	✓	✓	✓
"Mundgerecht"	✓	✓	✓
Leicht anwendbar	✗	✓	✓
Kontextbezogen	✗	✗	✗

Tabelle 2.1: Erfüllte M-Learning Kriterien der Apps

Viele M-learning-Apps setzen eine konstante Internetverbindung für die uneingeschränkte Nutzung sämtlicher Funktionen voraus. In dem Vergleich wird deshalb angenommen, dass diese vorhanden ist und so das Kriterium der Tragbarkeit von allen drei Apps, aufgrund ihrer Plattform, dem iPad, erfüllt wird. Bei einer kritischeren Betrachtung erkennt man, dass dies nur bei "Skeleton System Pro III" der Fall wäre (außer für das Teilen der Inhalte über Social Media Plattformen).
Ein weiteres Kriterium, dass von allen verglichenen Apps erfüllt wird, ist das keine expliziten Lehrer zur Vermittlung der Lehrinhalte nötig sind (selbständiges beziehungsweise privates Lernen). Die Erfüllung der noch ausstehenden Kriterien wird bei jeder App nachfolgend im Detail betrachtet.

Im Vergleich wird klar, dass das App "Skeleton System Pro III" am weitesten von der Definition eines M-Learning Apps, im Sinne von Traxler [36], entfernt ist. Das Spontanität-Kriterium wird nicht erfüllt, da die App gezielt zum Erlernen von anatomischen Strukturen eingesetzt wird. Das Lernen findet zudem nicht im selben Kontext statt, in dem es angewandt wird, da die App sonst beispielsweise in direktem Zusammenhang mit einer Pathologie oder ähnlichem verwendet werden müsste, wo der Anwender direkt mit anatomischen Strukturen in Berührung kommt. Durch ihren Umfang an Funktionen braucht die App eine gewisse Einarbeitungszeit und verfehlt somit das Kriterium der leichten Anwendbarkeit. Die Umgebung wird bei Anwendung der App ebenfalls nicht miteinbezogen.
Erfüllt werden hingegen, zusätzlich zu den von allen Apps erfüllten Kriterien, das informelle und "mundgerechte" Lernen. Die App kann als informell angesehen werden, da der Benutzer sich "spielend" neues Wissen aneignen kann, indem er ein Quiz ausführt. Durch dieses Quiz wird zudem nur ein Teil der Lehrinhalte vermittelt, so dass der Anwender nicht damit überfordert wird, sich zu viel neues Wissen auf einmal aneignen zu müssen.

Die "Science for iPad"-App erfüllt nur zwei der Kriterien nicht. Dazu gehört, dass durch die App vermittelte Wissen nicht unbedingt unmittelbar einsetzbar ist. Beispielsweise ließt man einen Artikel über neueste Forschungserkenntnisse während einer Zugfahrt. Als weiteres Kriterium wird nicht erfüllt, dass die Lehrinhalte einen Bezug zum aktuellen Kontext (etwa dem aktuellen Standort) des Anwenders herstellen.
Die restlichen Kriterien werden hinreichend erfüllt. So erhält man beispielsweise eine einfache

Übersicht der angebotenen Artikel mithilfe der "360 View" und lernt unbewusst beziehungsweise spontan in seiner Freizeit, in dem man sich Artikel durchliest, an denen man interessiert ist. Darüber hinaus wird durch die Aufteilung der Lehrinhalte in Artikeln nicht zu viel Wissen auf einmal vermittelt.

Die meisten Kriterien erfüllt die "Snapguide" App. Das einzige Kriterium, welches auch hier nicht umgesetzt wird, ist, dass keine Informationen durch Sensoren oder GPS miteinbezogen werden, um die Lehrinhalte auf den Anwender anzupassen.
Ansonsten werden alle Kriterien erfüllt. Die Lehrinhalte werden beispielsweise leicht einprägsam vermittelt, da sie in Form von Anleitungen und Tipps vorliegen. Diese werden für sämtliche Lebensbereiche geboten werden (Kochrezepte, Tipps zur Tiererziehung, usw.) und sind deshalb sowohl spontan als auch informell. Die vermittelten Lehrinhalte werden zumeist auch sofort eingesetzt und sind durch die Benutzeroberfläche der App leicht auffindbar.

3 Anforderungsanalyse

Die anfänglichen Anforderungen der Anwender wurden hauptsächlich aus dem bestehenden System der Präpmappe herausgearbeitet. Zudem gab es noch kurze Interviews mit den beteiligten Lehrkräften des Instituts für Anatomie und Zellbiologie der Universität Ulm.

3.1 Anforderungen der beteiligten Benutzergruppen

Bevor die Anforderungen formuliert werden können, müssen die verschiedenen Benutzerrollen definiert werden, da das System, je nach Benutzer, unterschiedliche Funktionen erfüllen muss.

Benutzerrolle	**Beschreibung**
Administrator	Administratoren sind alle Personen, die Lehrinhalte für das System bereitstellen und Zugriff auf entsprechende Funktionen erhalten, um diese zu erstellen (beispielsweise Dozenten oder Professoren beziehungsweise wissenschaftliche Mitarbeiter, die für das Einpflegen von Daten in das System verantwortlich sind). Solche Funktionen sind etwa das Hinzufügen neuer Testate oder das Einpflegen von Bildmaterial in das System.
Student	Studenten sind alle Personen, die lediglich daran interessiert sind, Testate beziehungsweise Übungen durchzuführen und auswerten zu lassen, um sich die Lehrinhalte anzueignen.

Aufgrund dieser zwei unterschiedlichen Benutzerrollen, werden zunächst die gemeinsamen Anforderungen dargestellt und danach die benutzerspezifischen Anforderungen behandelt. Zu den gemeinsamen Anforderungen gehören:

Mobilität
Der Benutzer sollte die App unabhängig von seinem derzeitigen Standort einsetzen können. So könn-

ten die Anwender zuhause, an der Universität, oder auch während dem Pendeln (beispielsweise bei einer Zugfahrt) die App im vollen Funktionsumfang verwenden.

Speicherung der Daten
Die erstellten Übungen müssen in geeigneter Form lokal auf dem Gerät gespeichert werden. Dadurch soll der dauerhafte Zugriff auf sämtliche Inhalte, auch ohne Internetverbindung, gewährleistet werden. Darüber hinaus sollen Ladezeiten durch eventuelle Begrenzungen des Netzwerkes, beispielsweise aufgrund geringer Bandbreite oder schlechter WLAN-Verbindung, minimiert werden.

Orientierung Es sollte sowohl die Orientierung im Hochformat als auch im Querformat unterstützt werden.

Login
Bisher konnte sich jeder Student mit gültigen Logininformationen bei der Moodle Lernplattform (siehe Abbildung 2.1) anmelden und die Veranstaltung "Makroskopische Anatomie (Präparierkurs)" abonnieren, um Zugang zu den Testaten zu erhalten. Da allerdings das verwendete Bildmaterial nicht frei verfügbar sein sollte, wurde der Zugang zu der Präpmappe durch ein Passwort geschützt. Dementsprechend wäre auch in der App eine gezielte Begrenzung des Zugangs zu den Daten erwünscht. Daher soll der Anwenderkreis auf Studenten und Administratoren begrenzt werden, welche die entsprechenden Veranstaltungen organisieren, beziehungsweise besuchen. Um dies zu realisieren, empfiehlt sich den Studenten und Administratoren einen Benutzernamen und Passwort zuzuweisen, mit denen sie Zugang zu den Testaten erlangen können.

Synchronisierung der Daten
Sämtliche Testate liegen auf einem Server der Universität Ulm und die Anwender greifen über ihren Webbrowser auf dieselben Inhalte zu. Da in einer mobilen App die Inhalte verteilt auf den Geräten vorliegen, aber alle Benutzer über dieselben Daten verfügen sollten, ist eine Synchronisierung zwischen den Geräten notwendig [29, 28]. Genauer gesagt benötigt ein Administrator die Möglichkeit, nach dem Erstellen neuer oder nach dem Bearbeiten bestehender Testate, diese mit externen Daten zu synchronisieren, während bei einem Studenten sicher gestellt werden muss, dass er immer mit den aktuellen Daten arbeitet.

3.1.1 Administratoren

Bisher wurden die Übungen der Testate in Form von PDF-Dateien und HTML-Dokumenten angeboten. Beim Anlegen einer Übung im PDF-Format müssen dabei zwei Dateien erstellt werden. Die erste der PDF-Dateien enthält ein Bild mit Markierungen, wie Pfeile, Klammern, Buchstaben, Ziffern usw., aber keine Beschriftungen, während in der zweiten die Beschriftungen vorhanden sind. Beide Dateien sind über einen Button miteinander verlinkt und so kann zwischen beschrifteter und unbeschrifteter PDF-Datei gewechselt werden. Die Übungen, welche in Form eines HTML-Dokumentes dargestellt werden, sind dagegen bereits beschriftet. Für die Navigation zu den verschiedenen Übungen werden sie unter bestimmten Testaten als Links im HTML-Format aufgelistet (siehe Abbildung 3.1).
Insgesamt war es also notwendig, zuerst die PDF-Dateien beziehungsweise HTML-Dokumente zu erstellen, und danach die Auflistung dementsprechend anzupassen. In der App, welche im Kontext dieser Arbeit entwickelt wurde, sollten die Formate der Übungen vereinheitlicht und das Anlegen und Bearbeiten von Übungen vereinfacht werden.

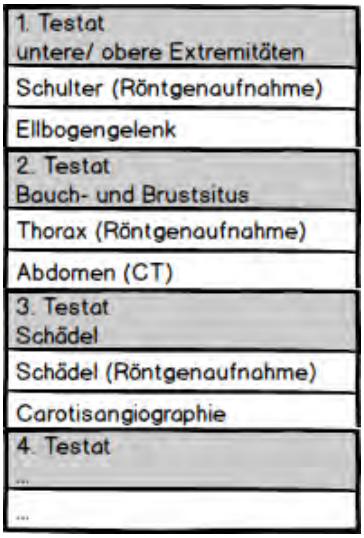


Abbildung 3.1: Bisherige Auswahl von Übungen in der "Präpmappe"

Anlegen einer Dateistruktur
In der Rolle des Administrators muss es eine Möglichkeit geben, neue Testate und Übungen anzulegen. Zusätzlich ist es notwendig, dass Übungen bestimmten Testaten zugewiesen werden können.

3D-Modelle
Als weitere Anforderung wurde der Wunsch nach 3D-Modellen der anatomischen Strukturen geäußert. Diese sollten beliebig schwenkbar sein, um so von allen Seiten betrachtet werden zu können.

Bildmaterial einfügen
Als Grundlage jeder Übung soll ein Bild gewählt werden können. Daher ist es sinnvoll, verschiedene Möglichkeiten anzubieten, um eine Bilddatei in das System einzupflegen. Dabei sollten Bilddateien von einer Webressource aus geladen und direkt aus den auf dem iPad gespeicherten Bildern ausgewählt werden können.

Bildmaterial bearbeiten
Das eingefügte Bildmaterial sollte mit Pfeilen versehen werden können, um dem Studenten eindeutig zu vermitteln, auf welche Bildbereiche sich die Bezeichnungen beziehen.

Beschriftungen einfügen
Das Bildmaterial soll nicht durch die Beschriftungen überlagert werden. Es sollten also keine direkt sichtbaren Beschriftungen enthalten sein, wenn die Übung geöffnet wird. Vielmehr sollte es möglich sein, Zeiger einzufügen, welche beschriftet werden und die zugeordnete Beschriftung bei Berührung anzeigen. Zudem sollten verschiedene Arten von Zeigern, die verschiedene Semantiken repräsentieren, verfügbar sein. Dies dient beispielsweise dazu, punktgenaue Zeiger von Zeigern, welche einem größeren Bereich des Bildes zugewiesen werden, zu unterscheiden.

3.1.2 Student

Um eine Übung zu bearbeiten, muss sich ein Student bisher zuerst an der Lernplattform Moodle der Universität Ulm anmelden. Danach kann er, wenn er bei der dafür vorgesehenen Veranstaltung *"Makroskopische Anatomie (Präparierkurs)"* angemeldet ist, die Übungen aufrufen. Eine Übung besteht

dabei aus entweder zwei PDF-Dokumenten oder einem HTML-Dokument. Bei HTML-Dokumenten ist das Bild bereits voll beschriftet. Bei Übungen im PDF-Format ist die PDF-Datei, welche der Student zuerst angezeigt bekommt, unbeschriftet. Sie enthält lediglich Markierungen und Zeiger auf die zu beschriftenden Bereiche beziehungsweise Punkte. Bisher hat der Student keine Möglichkeit, seine Antworten direkt in das PDF-Dokument zu schreiben, und muss sich daher seine Antworten extern notieren (beispielsweise in eine Textdatei oder auf ein Blatt Papier). Nach Klicken eines Buttons wird ein neues PDF-Dokument geöffnet, welches die korrekten Beschriftungen enthält. Mit diesem muss der Student jetzt die notierten Antworten Schritt für Schritt vergleichen. Diesem umständlichen Testablauf sollte mit der App entgegengewirkt werden. Im Folgenden werden die Anforderungen, welche für die Studenten gelten, dargestellt:

Gezielte Auswahl von Übungen
Der Student muss in der Lage sein, gezielt und ohne großen Aufwand zu der gewünschten Übung zu gelangen.

Zwischenergebnis anzeigen
Aus Sicht des Studenten ist das Primärziel des Systems das Selbststudium. Deshalb sollte es ihm möglich sein, seine Eingaben vorläufig auswerten zu lassen. Er bekommt dadurch ein Feedback zu seinen bisherigen Eingaben und kann diese gegebenenfalls verbessern.

Übung auswerten
Will der Student die Übung endgültig auswerten lassen, soll ihm angezeigt werden, welche seiner Eingaben korrekt waren, und bei falschen Eingaben, welches die korrekten Beschriftungen gewesen wären. Nach Abschluss einer Übung sollte eine weitere Bearbeitung nicht mehr möglich sein.

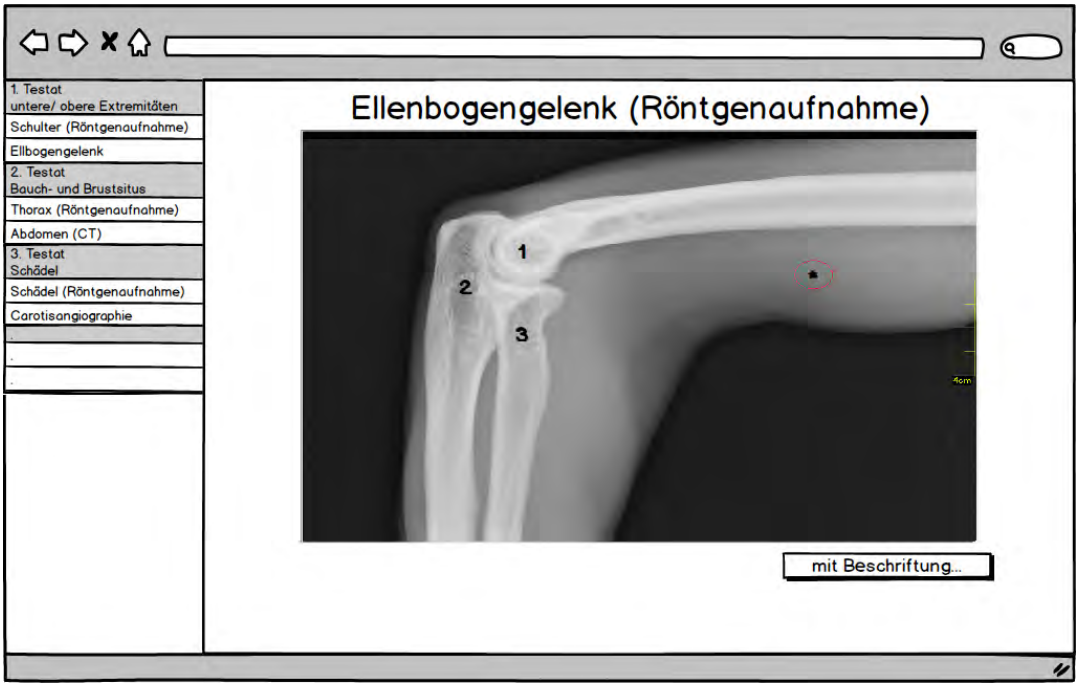
3.2 Weitere Anforderungen

Nachdem es einen ersten Entwurf der App gab, fand ein erneutes Treffen mit den beteiligten Lehrkräften des Instituts für Anatomie und Zellbiologie statt. Daraus entstanden drei weitere Anforderungen, welche in den Entwicklungsprozess eingebunden werden sollten. Dabei handelte es sich um:

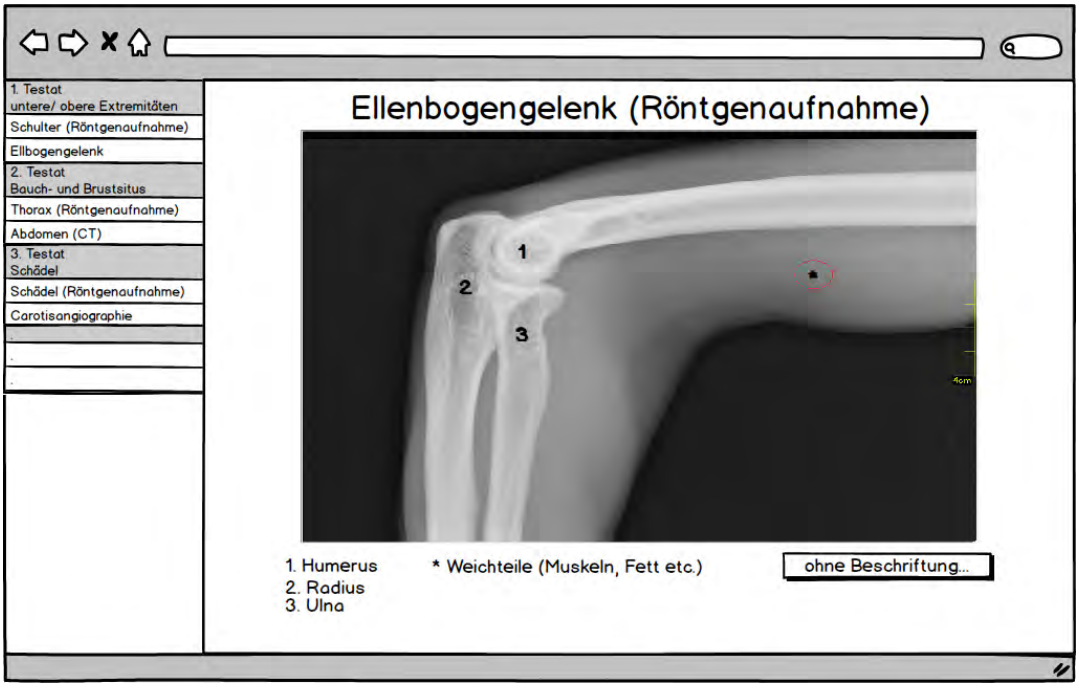
Begriffslisten
Einem Zeiger sollten mehrere gültige Beschriftungen zugeordnet werden können. Die Anforderung nach mehreren gültigen Beschriftungen leitete sich daraus ab, dass im System nur zweidimensionales Bildmaterial zu Verfügung steht und so anatomische Strukturen nicht immer eindeutig bestimmt werden können, wie dies in einem dreidimensionalem Modell möglich wäre. Zudem haben manche der anatomische Strukturen mehrere gültige Bezeichnungen.

Mehrsprachigkeit
Da es auch nicht deutschsprachiger Medizinstudenten an der Universität gibt, sollte es im System die Möglichkeit geben, verschiedene Sprachen auszuwählen. Dies wirkt sich auf die Dialoge und den Buttons aus, welche dem Anwender angezeigt werden. Zudem müssten, bei den bereits bestehenden Übungen, die Begriffslisten in den angebotenen Sprachen eingepflegt werden.

Bereichsmarkierungen
Das System sollte die Möglichkeit bieten, Bildbereiche zu markieren. So kann man nicht nur gezielt auf Bilddetails aufmerksam machen, indem man dementsprechend Pfeile in das Bild zeichnet, sondern auch gröbere Details umranden, beziehungsweise vom Rest des Bildes abheben.



(a)



(b)

Abbildung 3.2: Übung ohne (a) und mit (b) Beschriftung

Zusätzlich zu diesen Anforderungen wurde eine weiterer Weg, um Bilddateien in das System einzupflegen, erarbeitet. Die Idee war, direkt mit der Kamera des iPads Bilder aufzunehmen und diese

für die Übungen zu verwenden. So haben Administratoren später die Möglichkeit, beispielsweise direkt Röntgenbilder abzufotografieren, und auf Basis derer neue Übungen zu erstellen.

3.3 Anforderungen an die Hardware

Die Testate sollten ortsunabhängig und zu jeder Zeit ausführbar sein. So kann sich der Student selbst einteilen, wann er eine Übung bearbeiten möchte. Andererseits soll eine intuitive Benutzeroberfläche für die Administratoren geschaffen werden, um schnell neue Testate und Übungen erstellen zu können. Da in den Übungen Bilder benutzt werden, empfahl es sich außerdem, auf einem Gerät zu entwickeln, dass die nötige Bildschirmgröße für die detaillierte Darstellung der Bilder bietet. Zudem war es wichtig, dem Benutzer möglichst intuitive Möglichkeiten zur Bearbeitung und Beschriftung der Übungen zu bieten.
Deshalb bot sich die Entwicklung der Anwendung (*App*) auf einem Tablet-Computer an. Als mögliche Alternativen bot sich die Entwicklung für *Android*-, *iOS*- oder *Windows*-Geräte an. Da, zum Zeitpunkt dieser Arbeit, für das Institut für Zellbiologie und Anatomie die Anschaffung von neuen iPad Geräten für die mobile Lehre geplant war, bot sich die Entwicklung auf dieser Plattform an.
Das von Apple entwickelte Gerät bietet, mit einer Bilddiagonalen von 24,63 cm und einer Auflösung von bis zu 2048 x 1536 Pixeln (iPad 4), intuitiver Benutzereingaben durch das Multi-Touch Display[1], und seinem darauf abgestimmten Betriebssystem (*iOS*) [13], die nötigen Voraussetzungen. Zudem unterstützt sein geringes Gewicht den ortsunabhängigen Einsatz.

3.4 Zusammenfassung

In diesem Kapitel wurden die Anforderungen an das zu entwickelnde System anhand des bisherigen Systems und Interviews mit den beteiligten Lehrkräften definiert.
Nach Analyse des bestehenden Systems konnten verschiedene Schwachstellen aufgedeckt werden. Diese werden in Tabelle 3.1 zusammengefasst dargestellt. Diese Schwachstellen rühren vor allem daher, dass es bisher kein gemeinsam genutztes Werkzeug beziehungsweise Programm gab, um die Übungen zu entwerfen. Um eine gewisse Konsistenz der Testate zu gewährleisten, sollte also für jede Übung dieselben Werkzeuge zur Beschriftung und Markierung der Bildbereiche bereitgestellt werden.
Aus Sicht des Studenten ist es vor allem wichtig, das ihm die Möglichkeit geboten wird, die Übungen direkt zu beschriften und auswerten zu lassen. Eine weitere Anforderung an das System, die durch die Interviews definiert wurde, ist das Auswerten der Übungen noch während sie von dem Studenten bearbeitet werden. So hat er die Möglichkeit, falsche Antworten nochmals zu überarbeiten, bevor er die Übung endgültig auswerten lässt.
Nach der Analyse der Anforderungen wurde deutlich, dass es sich anbieten würde, die Anwendung um die Aspekte Darstellung und Mobilität zu vereinen, auf einem iPad zu entwickeln.

[1] berührungsempfindlicher, kapazitiver Bildschirm, der sich mittels Multi-Touch-Gesten bedienen lässt.

Schwachstelle	Beschreibung
Unterschiedliche Beschriftungs- und Markierungselemente	Es existiert keine einheitliche Darstellung der Übungen. Zu bezeichnende Bereiche sind beispielsweise in manchen Übungen mit Buchstaben und in anderen Übungen mit Zahlen markiert.
Verschiedene Dateiformate der Übungen	Übungen werden sowohl im PDF-Format als auch im HTML-Format angeboten.
Kompliziertes Erstellen neuer Übungen	Das Erstellen neuer Übungen ist nur mit dem Einsatz externer Programme möglich. Beispielsweise wird zuerst ein Bildbearbeitungsprogramm benötigt, um das Übungsbild entsprechend anzupassen und danach ein Programm, um daraus eine PDF-Datei zu erstellen.
Fehlende Unterstützung bei der Auswertung von Übungen	Bei der Auswertung der Übungen ist der Student auf sich allein gestellt. Es könnte leicht passieren, dass eine Antwort nicht ausgewertet wird, da sie übersehen wird.
Eingeschränkte Mobilität	Das System wird zwar in einem Webbrowser ausgeführt, ist aber nicht für mobile Geräte optimiert und daher auf diesen nicht effizient einsetzbar.
Unterschiedliche Präsentation der Übungen	Die meisten Übungen werden auf der rechten Seite des Browsers angezeigt. Andere wiederum öffnen ein neues Fenster und bieten weiter Navigationsmöglichkeiten an, was die Anwendung des Systems unnötig kompliziert macht.
Schlechte Qualität der Bilder	Die bisher im System für die Übungen eingesetzten Bilder sind teilweise verschwommen oder liegen nur in einer schlechten Auflösung vor.
Abprüfen der Übungen	Die Ergebnisse einer Übung können im bisherigen System nicht zentral erfasst und ausgewertet werden. Infolge dessen kann keine einheitliche Benotung beziehungsweise Bewertung von Studenten erfolgen.

Tabelle 3.1: Schwachstellen des bisherigen Systems

4
Entwurf

In diesem Kapitel erfolgt, anhand der in Kapitel 3 genannten Anforderungen, ein Designentwurf für den später in Kapitel 6 umgesetzten Prototyp. Bei der Umsetzung eines Softwareprojektes hat man die Wahl aus einer Vielzahl von Softwareentwicklungsprozessen (siehe [39]). In dieser Arbeit wurde der Prozess des Prototyping eingesetzt. Unter Prototyping versteht man die "Entwicklung einer vorläufigen Version eines Software-Systems, damit bestimmte Aspekte des Systems genauer untersucht werden können" [22]. Hierbei wird oft zwischen horizontalen und vertikalen Prototypen unterschieden (siehe Abbildung 4.1). Ein horizontaler Prototyp realisiert nur einen ausgewählten Bereich einer Architektur, wie beispielsweise GUI-Prototypen, die nur die Benutzerschnittstelle demonstrieren, ohne eine darunterliegende technische Funktionalität umzusetzen. Vertikale Prototypen stellen dagegen nur einen kleinen Teil der Funktionalität bereit, realisieren diesen aber in sämtlichen Bereichen der Architektur. Da der erste Einsatz des Prototypen ursprünglich noch während den Arbeiten an dieser Diplomarbeit geplant war, wurde eine Mischform aus vertikalem und horizontalem Prototypen entwickelt, welche den Großteil der am Ende angestrebten Funktionalität in einem angemessenen Rahmen bietet. Außerdem war es notwendig, einen umfangreichen Prototypen zu entwickeln, da so die Konzepte direkt auf der am Ende dafür ausgelegten Hardware getestet werden konnten.

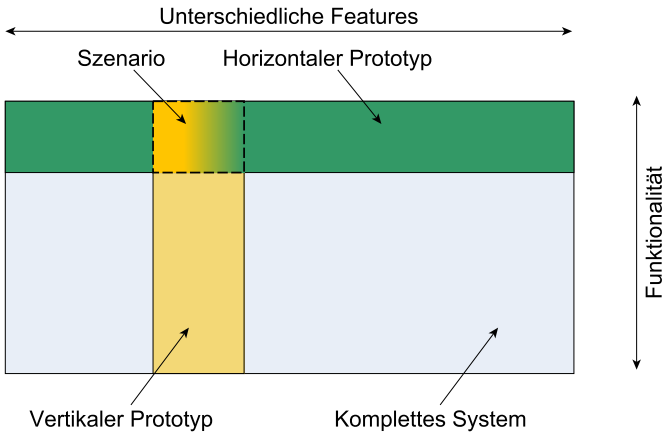


Abbildung 4.1: Horizontale vs. vertikale Prototypen. [26]

4.1 Informationsarchitektur

Im Folgenden wird die Informationsarchitektur der App dargestellt. Aus den Anforderungen ging hervor, dass, je Benutzerrolle, eine eigene Sicht implementiert werden sollte. Um die Benutzerrolle zu bestimmen, bekommt man zuerst einen Anmeldungsbildschirm angezeigt, wo mithilfe der Benutzerdaten eine Rolle ermittelt wird. Danach wird eine Sicht angezeigt, welche in eine Hauptsicht auf der linken, und eine Detailsicht auf der rechten Seite, aufgeteilt ist. Die dort gebotenen Funktionen richten sich nach der Rolle des Benutzers.
In der Abbildung 4.2 auf der nächsten Seite wird die grundlegende Informationsarchitektur der App dargestellt. Rechtecke stellen dabei Sichten dar. Die durchgezogenen Kanten symbolisieren die Reihenfolge der Sichten. Gestrichelte Kanten repräsentieren die Möglichkeit auf die vorherige Sicht zurückzuspringen und gepunktete Kanten zeigen an, dass die Sichten in anderen Sichten enthalten sind. Kreise stellen Funktionen dar, welche in den Sichten geboten werden.

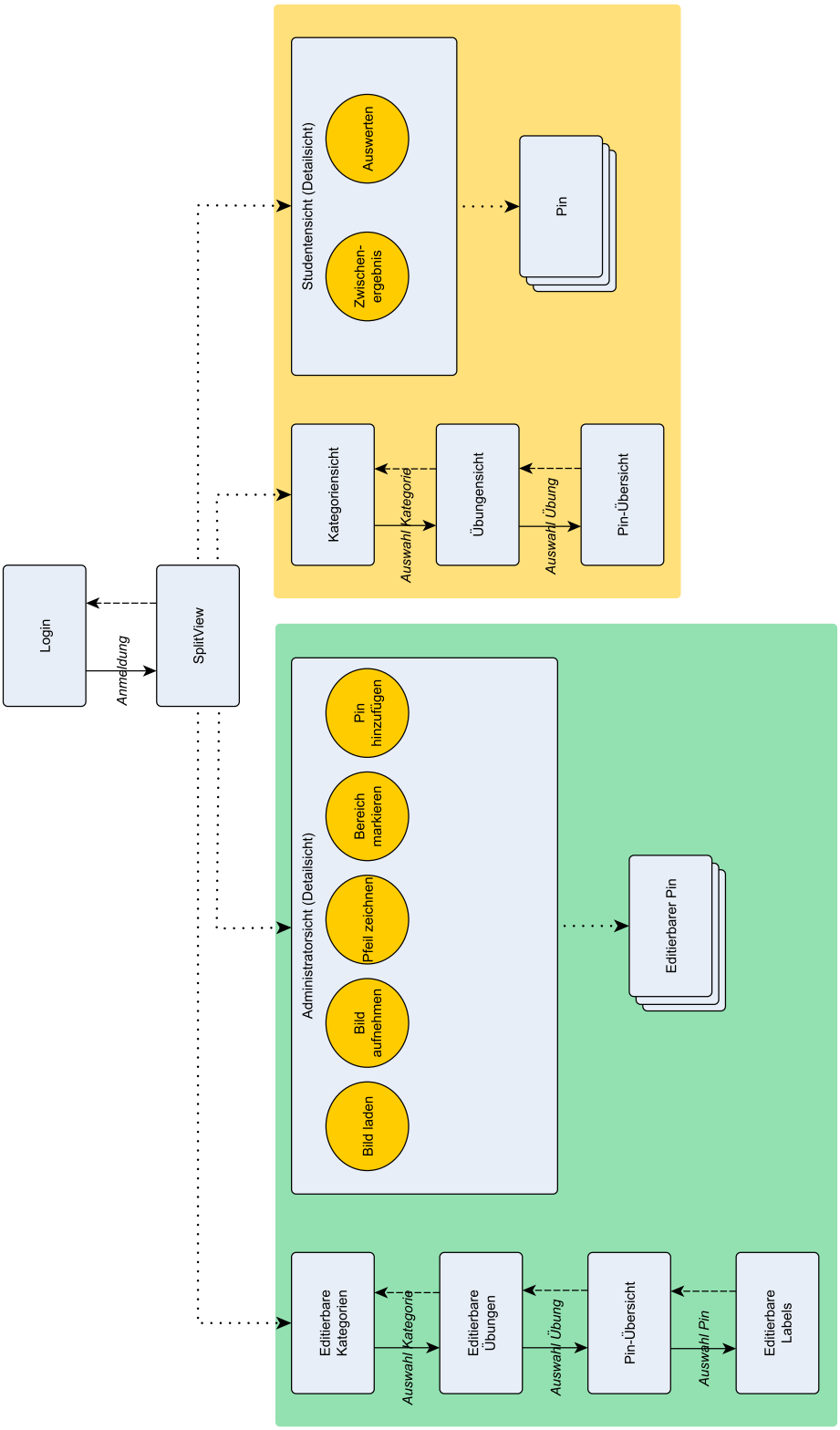


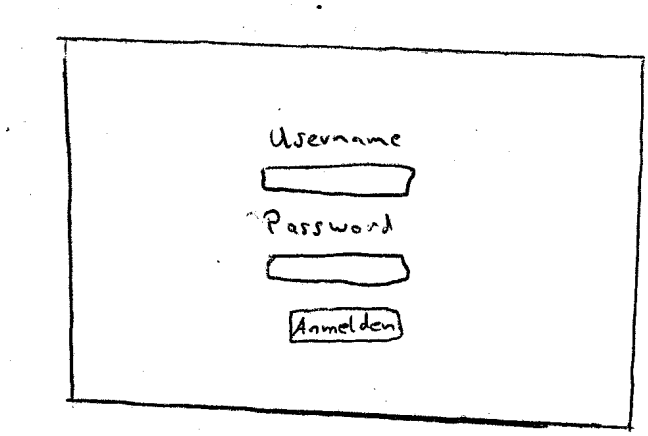
Abbildung 4.2: Die Informationsarchitektur der App

4.2 Screens

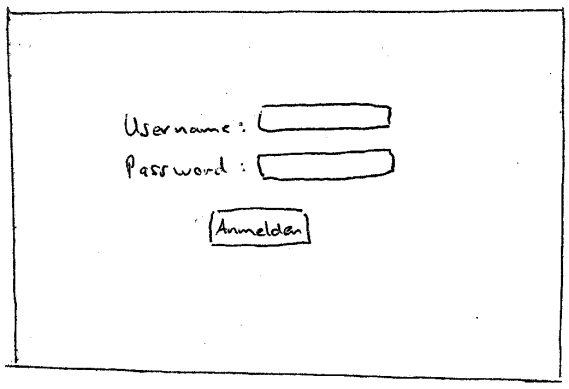
Nachdem die Anforderungen formuliert wurden, stellte sich schnell heraus, dass für die zwei Benutzerrollen (siehe Kapitel 3, Abschnitt 3.1) Student und Administrator, jeweils ein Bildschirm für das Ausführen einer Übung beziehungsweise das Erstellen und Bearbeiten einer Übung existieren muss. Für die Authentifizierung der Benutzer wurde ein Anmeldungsbildschirm vorgesehen.

4.2.1 Anmeldung

Nach dem Start der App soll dem Benutzer als erstes der Anmeldungsbildschirm angezeigt werden. Dieser sollte auf jeweils ein Eingabefeld für den Benutzernamen, ein Eingabefeld für das dazugehörige Passwort, deren Beschriftungen und einen Anmelde-Button beschränkt sein. Abbildung 4.3 stellt die in Betracht gezogenen Mockups dar.



(a)



(b)

Abbildung 4.3: In Betracht gezogene Anmeldungsbildschirme

4.2.2 Benutzersichten

Die Studentensicht und Administratorsicht sollten sich möglichst gleichen, so dass ein Administrator während dem Erstellen eines Testates bereits weiß, wie das Testat später einem Studenten präsentiert wird. Beide Sichten werden in eine Hauptsicht und in eine Detailsicht aufgeteilt, wie in Abbildung 4.4 dargestellt wird. Nachfolgend werden die Gemeinsamkeiten der beiden Sichten weiter erläutert:

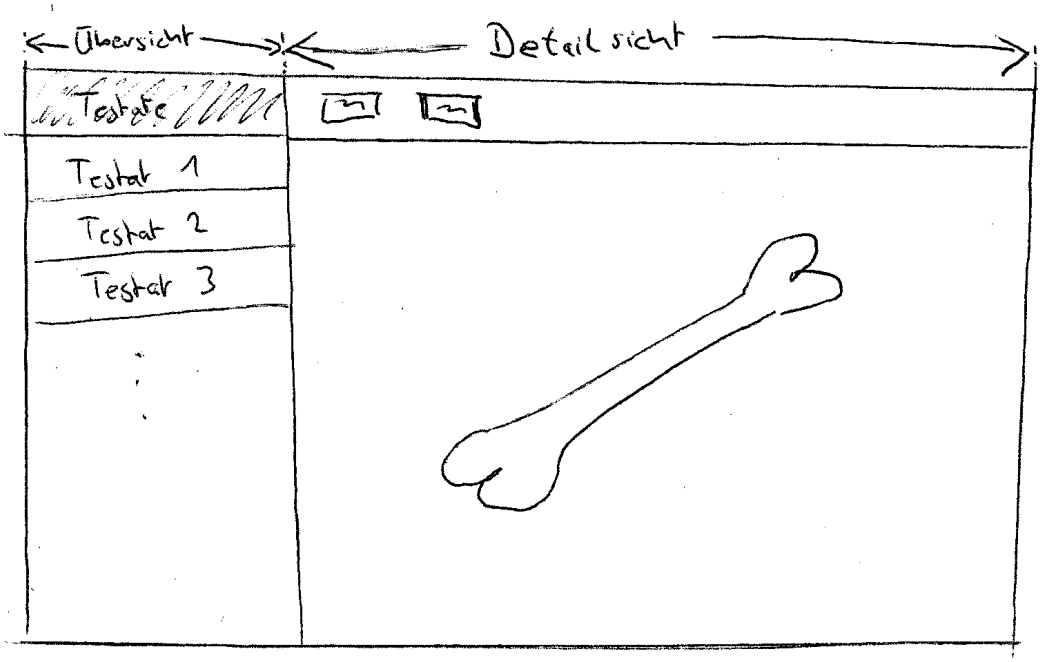


Abbildung 4.4: Das Basisdesign der Anwendung

Hauptsicht
Auf der linken Bildschirmseite wird dem Benutzer eine Übersicht über die vorhandenen Daten in Form einer Tabellenansicht geboten. Mit ihr hat er die Möglichkeit, sich durch die Datenhierarchieebenen zu navigieren. Hierbei wurden zwei Varianten angedacht, welche in Abbildung 4.5 dargestellt werden. In der ersten Variante sollte dem Benutzer nur eine Tabellensicht angezeigt werden. In dieser Sicht sollten die Bezeichnungen der Testate als Überschriften für die verschiedenen Abschnitte der Tabelle angezeigt werden und die dazugehörenden Übungen als Einträge in den Abschnitten. Die Darstellung sollte also analog zu der Darstellung im bisherigen System sein. Beim Entstehen dieses Entwurfs war allerdings die Anforderung, dass mehrere korrekte Beschriftungen für einen Pin in das System eingepflegt werden können, noch nicht vorgesehen. So sollte es ursprünglich nur möglich sein, Pins direkt in dem *Popover*, welcher beim Berühren eines Pins geöffnet wird, zu beschriften. Da sich das Hinzufügen und Anzeigen verschiedener Beschriftungen in dem *Popover* aber als zu kompliziert herausstellte, wurde eine neue Variante herausgearbeitet.
In dieser zweiten Variante sollte eine Tabellensicht angezeigt werden, in der sich ausschließlich eine Datenebene befindet (beispielsweise nur Testate oder nur Übungen) und erst nach dem Wählen einer dieser Daten, soll eine neue Tabellensicht angezeigt werden, bei deren Einträge es sich um die dazugehörigen Daten der nächsten Hierarchieebene handelt (beispielsweise alle Übungen des ausgewählten Testats). Um wieder auf eine höhere Hierarchieebene zu wechseln, bietet diese Sicht am oberen Rand eine Leiste in der sich, neben der Bezeichnung der aktuell angezeigten Ebene, ein "zurück"-Button

befindet.
In der Informationsarchitektur (siehe Abbildung 4.2) sind die verschiedenen Tabellensichten und die Navigationsmöglichkeiten zwischen ihnen zu sehen. Die Hauptsichten der Administratoren unterscheiden sich dabei von denen der Studenten, indem durch einen Button in der Leiste der Tabellensicht, neue Einträge hinzugefügt oder gelöscht werden können, und es zudem möglich sein sollte, die bestehenden Einträge umzubenennen.

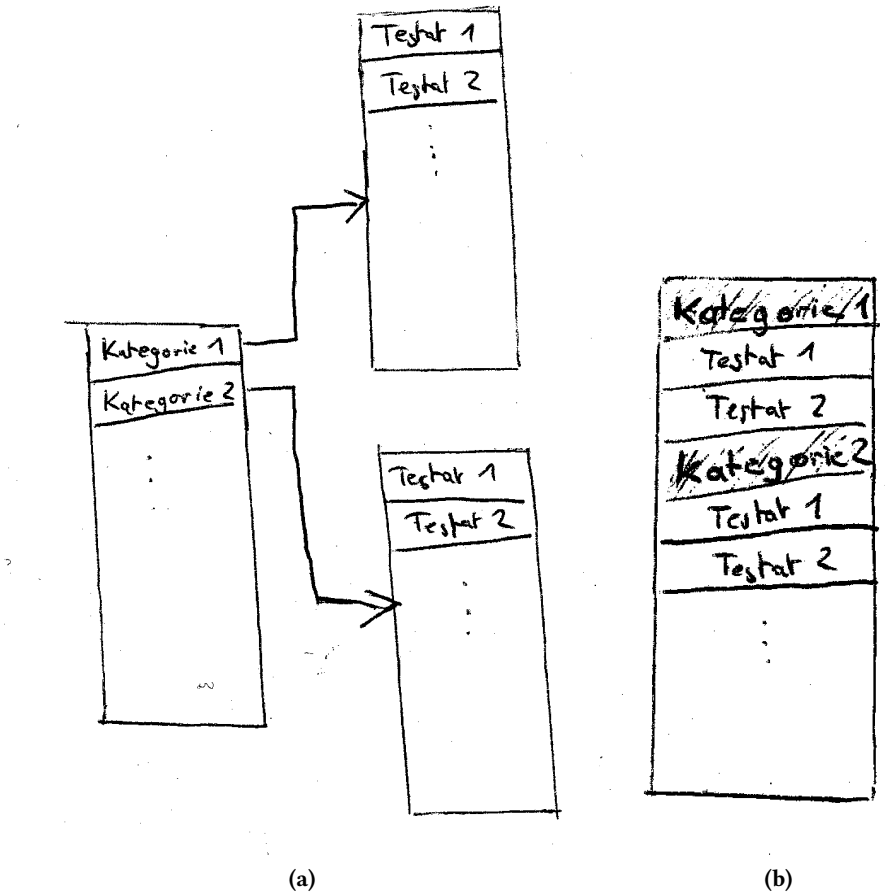


Abbildung 4.5: In Betracht gezogene Konzepte zur Übersicht über die Datenhierarchie

Detailsicht
Auf der rechten Bildschirmseite wird das ausgewählte Testat und eine Leiste (*Toolbar*), welche je nach Benutzerrolle verschiedene Buttons bietet, angezeigt. Während in der Studentensicht das bereits erstellte Testat angezeigt wird und der Fokus der Toolbar-Buttons auf Auswertungsfunktionen liegt, müssen in der Administratorensicht verschiedene Werkzeuge zum Anlegen und Editieren von Testaten vorhanden sein.

Pins Zur Bestimmung der zu beschriftenden Punkte wurden Bilder von Stecknadelköpfen (Pins) gewählt. Abbildung 4.6 zeigt die Pin-Grafiken, welche als Auswahl in Frage kamen. Letztendlich fiel die Entscheidung auf Pin 4.6c.

(a) (b) (c)

Abbildung 4.6: In Betracht gezogene Pin-Grafiken

4.2.2.1 Studentensicht

Das Hauptziel beim Entwurf der Studentenansicht war es, eine möglichst intuitive und komfortable Oberfläche zu bieten, um Testate zu beschriften und auszuwerten. Die Navigation zu dem gewünschten Testat sollte dem Studenten möglichst leicht fallen. Beim Beschriften der Pins war es vor allem wichtig, dass der Zusammenhang zwischen der Beschriftung und dem zu beschriftenden Punkt betont wird. Es sollte also eine klar zu erkennende Auswahl eines Pins möglich sein. Dies sollte mit dem Färben angewählter Pins und einer überlappenden Sicht (*Popover*), welche einen Zeiger auf den zu beschriftenden Pin besitzt, ermöglicht werden (siehe Abbildung 4.7).

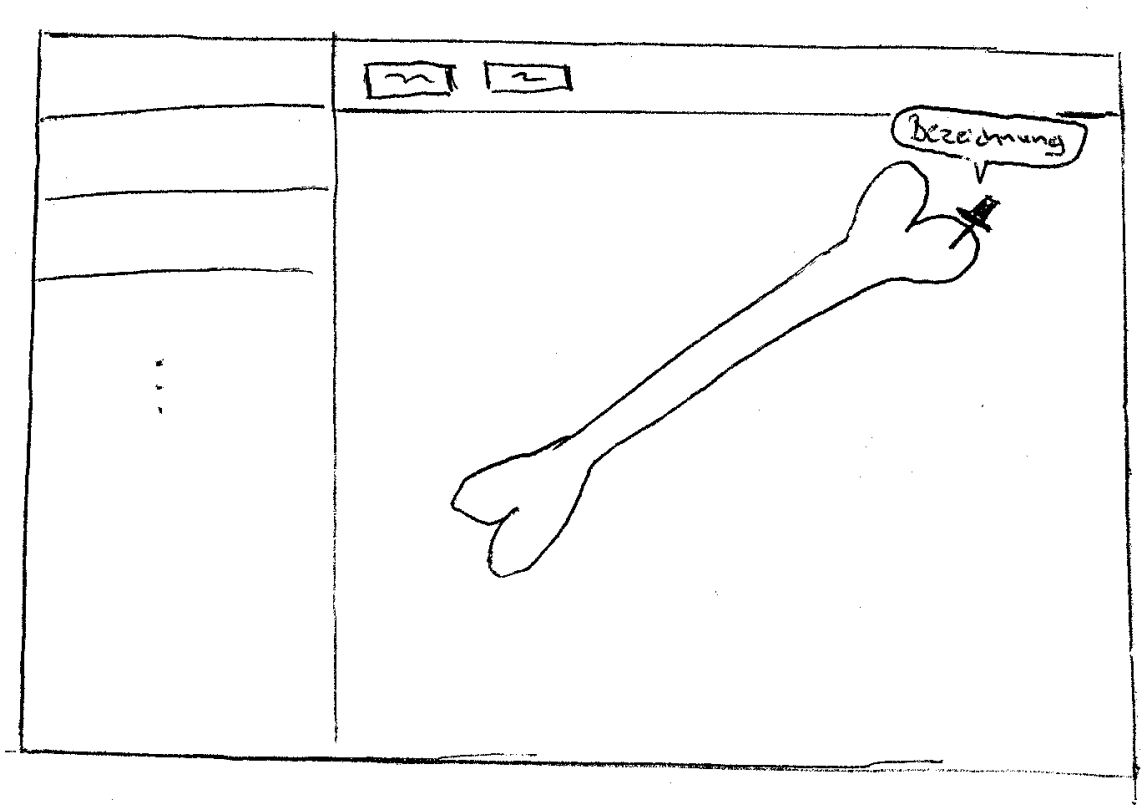


Abbildung 4.7: Ansicht bei Auswahl eines Pins in der Studentensicht

4.2.2.2 Administratorsicht

Die Administratorsicht bietet den größten Funktionsumfang. In dieser Sicht werden die Beschriftungen der Pins als eine weitere Hierarchieebene zu der Hauptsicht hinzugefügt. Sobald der Benutzer einen Pin anwählt, wechselt die Hauptsicht am linken Bildschirmrand zu den Beschriftungen. Außerdem sollte es nicht nur möglich sein, Daten auszuwählen, man sollte sie auch direkt in der Tabellensicht erstellen und bearbeiten können. Dies führte zu der Frage, wie sich eine solche Funktion auf einem Tablet möglichst benutzerfreundlich realisieren lässt. Es wurden zwei verschiedene Varianten herausgearbeitet, um dieses Problem zu lösen.
In der ersten Variante sollte dem Benutzer am linken Bildschirmrand eine Tabelle mit leeren Zellen angezeigt werden. Bei Tippen auf eine leere Zeile öffnet sich ein Kontextmenü, welches, je nachdem, auf welcher Datenebene sich der Benutzer befindet, das Anlegen eines neuen Testats, einer neuen Übung oder einer neuen Beschriftung ermöglichen soll. Wenn in der angetippten Zelle bereits ein Eintrag vorhanden war, soll sich der Benutzer zwischen dem Bearbeiten oder Löschen des Eintrages entscheiden können (siehe Abbildung 4.8a).
Die zweite Variante, welche letztendlich auch implementiert wurde, verzichtet auf das Einsetzen eines Kontextmenüs und ermöglicht stattdessen das Einfügen neuer Bezeichnungen mithilfe eines "Hinzufügen-Button" und das Umbenennen beziehungsweise Löschen existierender Bezeichnungen, indem der Benutzer eine Touch-Geste einsetzt. Die zum Umbenennen beziehungsweise Editieren geeignete Geste ist laut [8] das Gedrückthalten ("Touch and hold") der Bezeichnung. Für das Löschen wurde die von Apple dafür vorgesehene Wischgeste eingesetzt, welche einen "Löschen"-Button sichtbar macht, durch dessen Betätigung der Eintrag schließlich gelöscht werden kann (siehe Abbildung 4.8b).

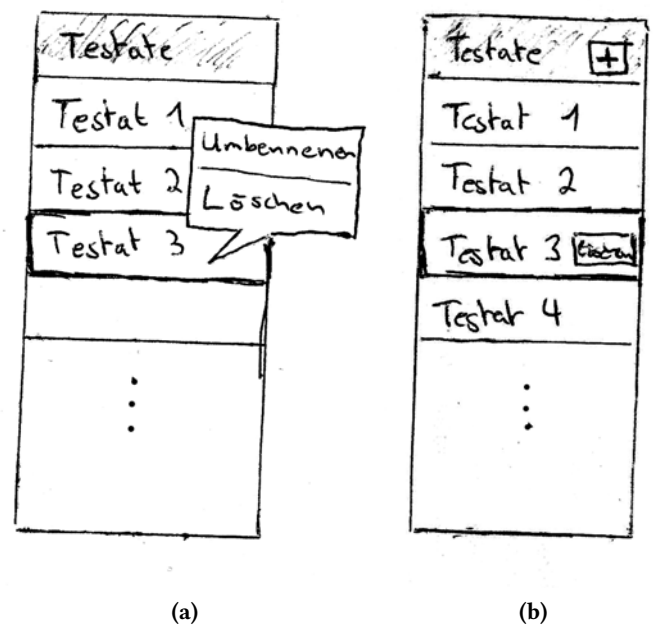


Abbildung 4.8: Bearbeitung von Einträgen in der Hauptsicht

Nachdem ein Testat und eine dazugehörige Übung angelegt und bezeichnet wurden, benötigt der Benutzer eine Möglichkeit, Bildmaterial als Grundlage für die Übung einzufügen. Wie in den Anforderungen definiert, sollten folgende Quellen, aus denen Bildmaterial geladen werden kann, verwendet werden:

1. Externes Bildmaterial aus einer Webressource.
2. Lokales Bildmaterial, welches auf dem iPad gespeichert ist.
3. Bilder, die mit der Kamera des iPads aufgenommen worden sind.

Dafür soll für jede Quelle ein Button implementiert werden, welches ein *Popover* öffnet, das Vorschaubilder zu den verfügbaren Bildmaterialien beziehungsweise das Vorschaubild zu dem Bild, welches mit der Kamera aufgenommen wird enthält (siehe Abbildung 4.9). Nach Auswahl eines Bildes wird dieses in der Detailsicht angezeigt.

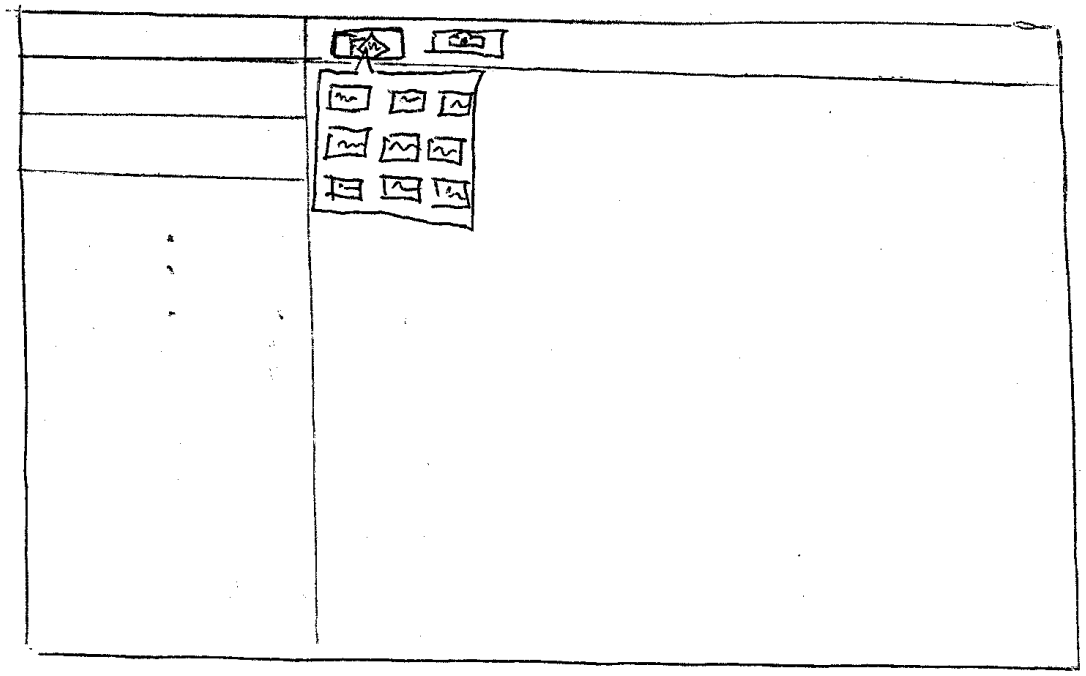


Abbildung 4.9: Vorschaubilder bei der Auswahl eines Bildes

Ist das Bildmaterial geladen, müssen gegebenenfalls Anpassungen stattfinden, um bestimmte Bildbereiche deutlicher hervorzuheben. Dazu sollten folgende Werkzeuge, welche über die Toolbar auswählbar sind, implementiert werden:

- *Pfeil-Werkzeug* Ein Werkzeug, dass dem Benutzer erlaubt mithilfe von Gesten Pfeile zu ziehen.
- *Bereichsmarkierungen* Mithilfe von Gesten können Rechtecke gezogen werden, welche die Farbe des Bildes verändern und so einen Bildbereich hervorheben.

Beispielanwendungen der Werkzeuge sind in Abbildung 4.10 dargestellt.

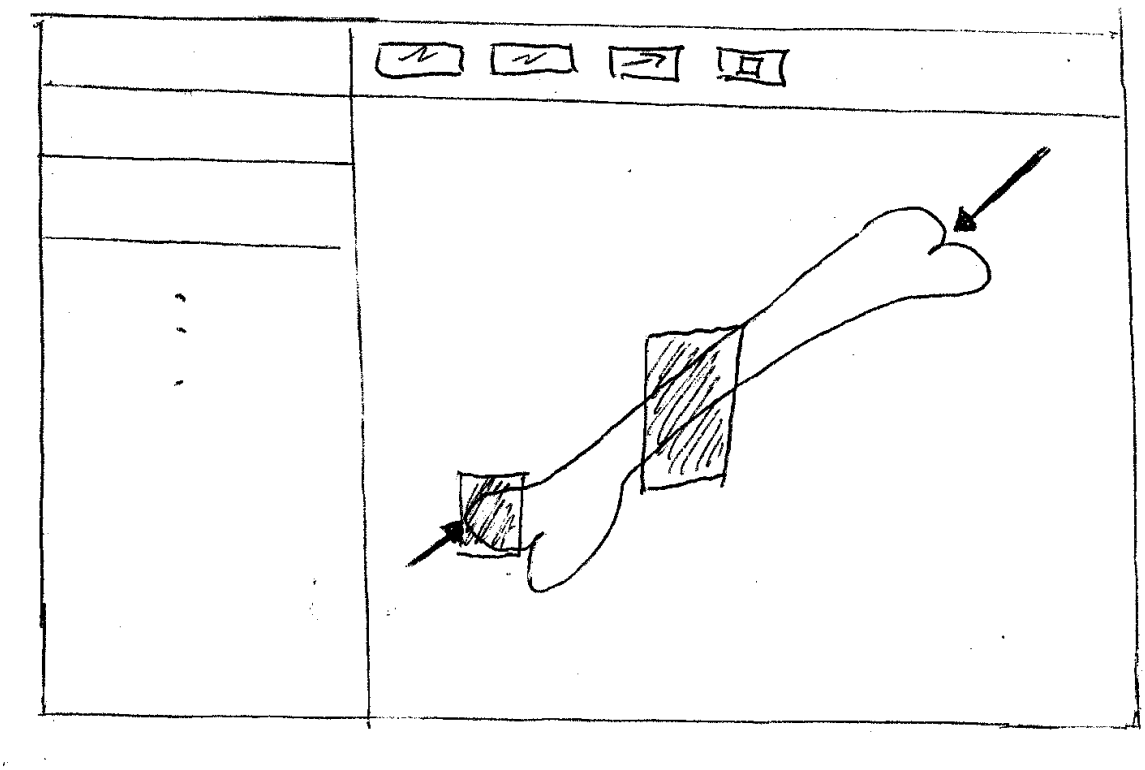


Abbildung 4.10: Beispielhafte Anwendung der Werkzeuge

Die wohl wichtigste Funktion umfasst das Einfügen der Beschriftungen in das System, welche in der Übung abgefragt werden. Die Zuordnung der Bildbereiche zu den Beschriftungen geschieht mithilfe von Pins. Um einen Pin zu setzen, sollte es in der Toolbar ein Button geben, welcher ein *Popover* anzeigt, in dem mehrere Pins zur Auswahl stehen. Der Sinn dahinter mehrere Arten von Pins anzubieten ist, dass man so jedem Pin eine eigene Semantik zuordnen kann. Sobald sich das *Popover* mit den Pins öffnet, sollte es möglich sein, den gewünschten Pin mit einer Wischgeste auf den gewünschten Bildbereich zu ziehen und sobald der Finger gehoben wird, diesen dort abzulegen. Nachdem der Pin gesetzt wurde, wechselt die Hauptsicht am linken Bildschirmrand zu den gültigen Beschriftungen. Hier können, wie auch schon bei den anderen Daten, neue Beschriftungen hinzugefügt und vorhandene bearbeitet werden.

4.3 Zusammenfassung

In diesem Kapitel wurde eine Informationsarchitektur entworfen, mit deren Hilfe das Navigationskonzept zwischen den Sichten dargestellt wird und es wurden die Funktionen definiert, welche in den Sichten implementiert werden sollten. Zudem wurden die Sichten und ihre Funktionen genauer spezifiziert. Die detaillierte Ausarbeitung der Anforderung ist die Grundlage, um mit der Implementierung des Systems zu beginnen. Mit diesem Thema befasst sich das Kapitel 6 dieser Arbeit.

5 Anwenderszenarien

In diesem Kapitel werden mögliche Szenarien aufgeführt, die ein Anwender durchlaufen könnte. Es wird ein Szenario aus Sicht eines Administrators und ein weiteres aus Sicht eines Studenten vorgestellt. Dabei werden die verschiedenen Elemente der Benutzeroberfläche und deren Funktionsumfang genauer betrachtet.

5.1 Szenario 1: Administrator erstellt eine Übung

Die Voraussetzung für dieses Szenario ist ein System, in dem bisher noch keine Daten angelegt wurden. Zudem ist der Anwender noch nicht in dem System angemeldet und verfügt über eine konstante Internetverbindung. Unter diesen Voraussetzungen werden in diesem Anwenderszenario folgende Schritte durchgeführt:

1. Der Anwender erstellt ein Testat
2. Zu diesem Testat wird eine Übung angelegt.
3. Als Grundlage für die Übung wählt er ein Bild aus der Fotobibliothek des iPads aus.
4. Dem Bild werden ein Pfeil und ein markierter Bereich, sowie Pins hinzugefügt.
5. Die Pins werden beschriftet.
6. Abschließend werden die erstellten Daten in die externe Datenbank hochgeladen, indem der Synchronisationsvorgang gestartet wird.

5.1.1 Login

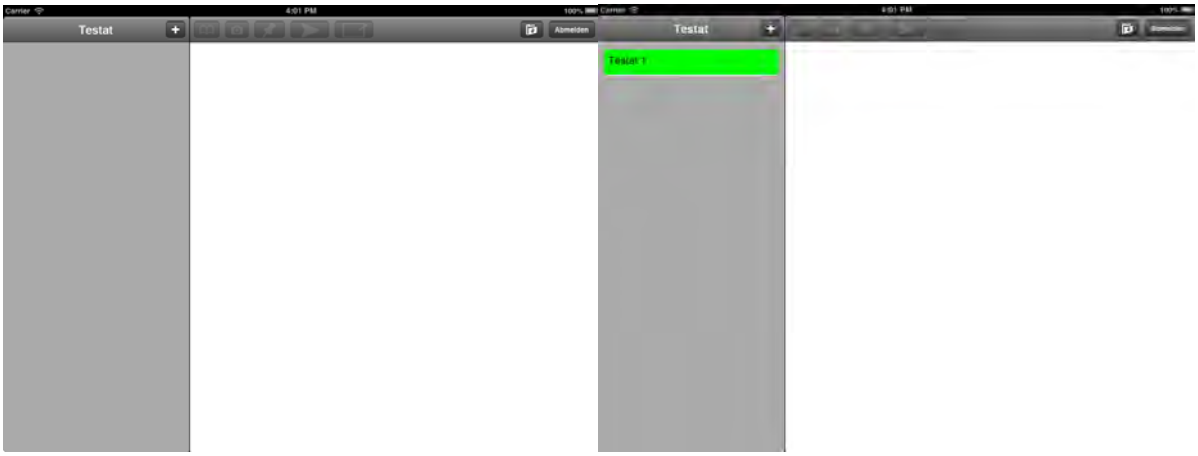
Nach Programmstart befindet sich der Anwender im Anmeldebildschirm. Dort werden zuerst die lokalen Daten mit den externen Daten synchronisiert. Während diesem Vorgang werden keine Eingaben des Benutzers von der GUI entgegengenommen. Dies wird mit einem sogenannten *Activity-*

Indicator[1] angezeigt. Nachdem die Synchronisation abgeschlossen wurde, muss der Anwender seine Benutzerdaten in die dafür vorgesehenen Textfelder eingeben. Um den Anmeldevorgang zu starten, drückt der Benutzer auf den "Anmelden"-Button. Wenn die Benutzerdaten korrekt waren und es sich bei ihm um einen Administrator handelt, wird er zu der Administratorsicht weitergeleitet.

(a) Die Synchronisierung bei Programmstart **(b)** Die Loginsicht der App

5.1.2 Administratorsicht: Testate

Die Administratorsicht ist aus einer Haupt- und einer Detailsicht aufgebaut. Beim ersten Anzeigen der Sicht sieht der Benutzer sämtliche Testate, auf die er Zugriffsrechte besitzt. Um ein Testat zu erstellen, drückt er auf den "Plus"-Button. Es wird ein neuer Eintrag zu der Tabelle hinzugefügt und eine virtuelle Tastatur präsentiert. Jetzt kann der Anwender eine Bezeichnung für das Testat eingeben und dies mit der "Return"- oder "Tastatur verbergen"-Taste bestätigen. Die Bezeichnung ist dabei grün hinterlegt, da sie noch nicht synchronisiert wurde.



(a) Die Administratorsicht ohne vorhandene Testate **(b)** Hinzufügen eines neuen Testats

[1] Ein Objekt zur Darstellung einer animierten Grafik, die dem Anwender vermitteln soll, das ein Ladevorgang ausgeführt wird.

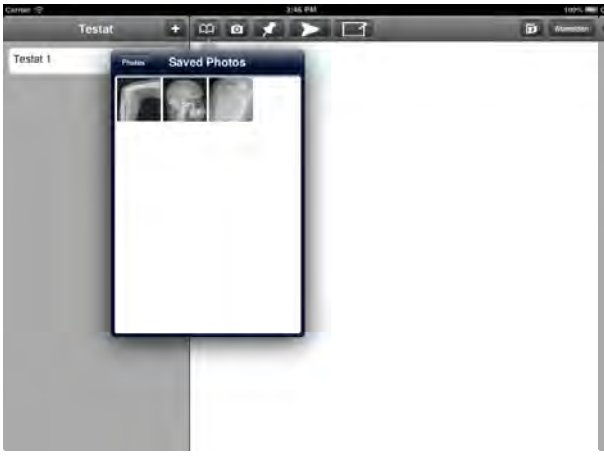
5.1.3 Administratorsicht: Übungen

Nachdem der Anwender das Testat aus der tabellarischen Übersicht ausgewählt hat, werden ihm die dazugehörigen Übungen in der Hauptsicht angezeigt. Zudem hat er die Möglichkeit, über ein Button in der Navigationsleiste der Hauptsicht zurück zu den Testaten zu gelangen. Um eine Übung hinzuzufügen, muss der Benutzer dieselben Schritte, wie zum Erstellen eines neuen Testats, ausführen.

Abbildung 5.1: Die Übersicht (links), nachdem eine Übung hinzugefügt wurde

5.1.4 Administratorsicht: Werkzeuge

Nach der Auswahl einer Übung wird eine Übersicht der zu der Übung gehörenden Pins in der Hauptsicht dargestellt. Wie auch schon bei der Hauptsicht zu den Übungen, hat der Anwender hier die Möglichkeit, mit einem Button zurück zur vorherigen Ansicht zu gehen. Zudem werden die Funktionen *Bild laden* und *Bild aufnehmen* aktiviert. In Abbildung 5.2a wird gezeigt, wie der Benutzer ein Bild aus der Fotobibliothek des iPads lädt.
Nachdem ein Bild als Übungsgrundlage gewählt wurde, werden weitere Funktionen zum Setzen der Pins, Zeichnen von Pfeilen und Markieren von Bildbereichen aktiviert.
Nach Aktivierung des Pfeilwerkzeugs wird dieses weiß hinterlegt dargestellt und der Benutzer kann Pfeile ziehen, indem er eine Wischgeste benutzt. Mithilfe des Markierungswerkzeugs kann er auf dieselbe Weise rechteckige Umrisse über das Bild ziehen, welche zu leicht transparenten, braun eingefärbten Regionen auf dem Bild werden, sobald er seinen Finger vom Bildschirm hebt.
Danach wählt er das Pinwerkzeug aus, welches ihm eine Übersicht aus zwei verschiedenen Pinarten anzeigt. Er zieht den gewünschten Pin aus der Übersicht auf den zu beschriftenden Punkt des Bildes. Hebt er seinen Finger, wird ein neuer Eintrag in der Hauptsicht eingefügt und er wird zur nächsten Hauptsicht weitergeleitet.



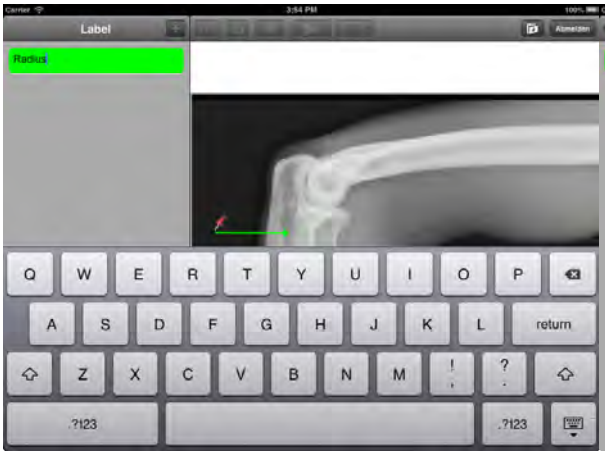
(a) Hinzufügen eines auf dem iPad gespeicherten Bildes

(b) Einsatz des Pfeilwerkzeugs

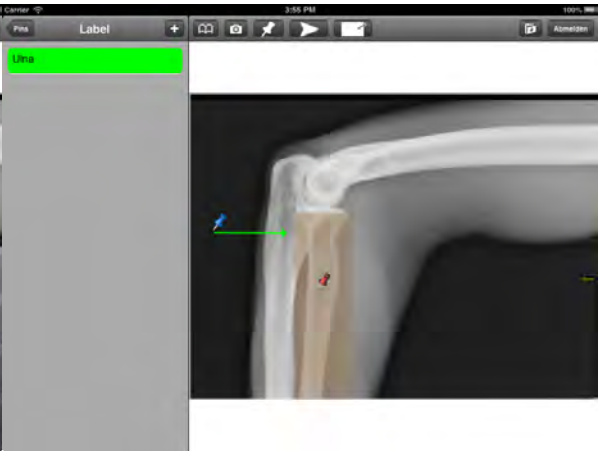
(c) Hinzufügen der Pins.

5.1.5 Administratorsicht: Labels

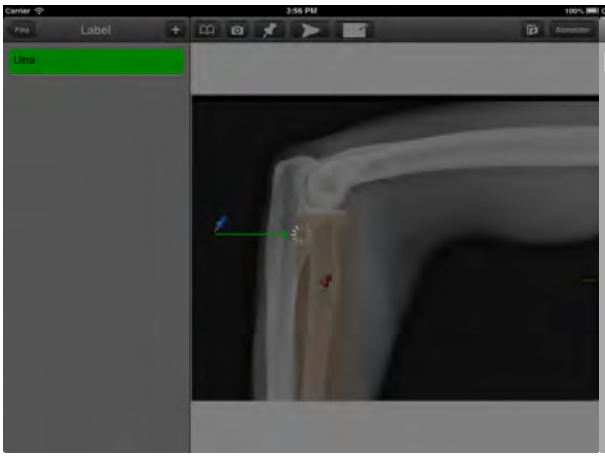
Sobald ein neuer Pin gesetzt oder ein vorhandener angewählt wird, wird dieser rot markiert und die Hauptsicht wechselt zur Übersicht seiner Labels. Die Labels sind alle Bezeichnungen, welche für den Pin als korrekt anerkannt werden. Neue Labels werden, wie gewohnt, mit dem "Plus"-Button hinzugefügt. Die Bezeichnung, welche an oberster Stelle der Hauptsicht steht, wird dem Student später bei der Auswertung als richtige Antwort angezeigt. Wird ein anderer Pin auf der Detailsicht ausgewählt, öffnet sich automatisch die Übersicht der auf diesen Pin bezogenen Labels. Zudem gibt es wieder die Möglichkeit, durch Betätigen eines Button in der Navigationsleiste zurück zur Pin-Übersicht zu wechseln. Will der Administrator die erstellten Daten mit den externen Daten abgleichen, klickt er auf den "Synchronisieren"-Button, worauf die Benutzeroberfläche gesperrt und ein *ActivityIndicator* angezeigt wird, bis die Synchronisation der Daten abgeschlossen ist. Nach Abschluss der Synchronisation wird die Detailsicht zurückgesetzt und in der Hauptsicht die Testate angezeigt. Zudem sind sämtliche Daten, welche erfolgreich synchronisiert wurden, jetzt weiß hinterlegt.



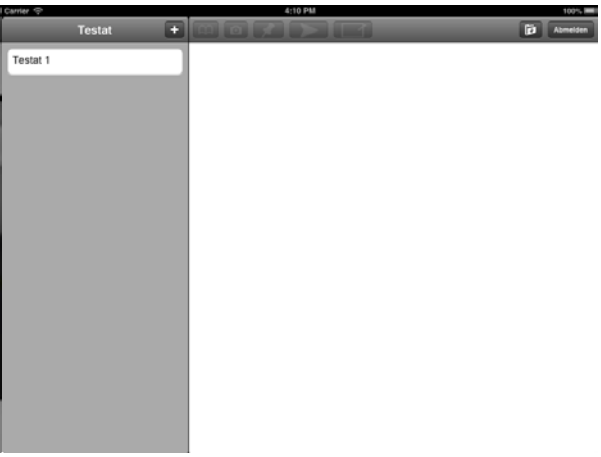
(a) Beschriftung eines Pins



(b) Anzeige der Labels eines ausgewählten Pins



(c) Synchronisierung einer Übung



(d) Die Ansicht nach einer erfolgreichen Synchronisation

5.2 Szenario 2: Student führt eine Übung durch

Der Student verfügt über dieselben Voraussetzungen wie im vorherigen Szenario. Er führt folgende Schritte durch:

1. Die, im vorherigen Szenario von dem Administrator angelegte, Übung wird geladen.
2. Die Pins der Übung werden beschriftet.
3. Der Student lässt sich ein Zwischenergebnis anzeigen.
4. Danach lässt er die Übung auswerten.

5.2.1 Login

Wie auch schon im ersten Anwenderszenario, wird zuerst ein abgedunkelter Login-Bildschirm mit einem *ActivityIndicator* als Ladesymbol angezeigt, während die Daten synchronisiert werden (siehe Abbildung 5.1a). Während diesem Vorgang wird das im vorherigen Szenario erstellte Testat und die

dazugehörige Übung auf das Gerät geladen. Danach meldet sich der Student mit seinen Benutzerdaten am System an. Anhand seiner Daten wird ihm eine Benutzerrolle zugewiesen und da es sich um die Rolle eines Studenten handelt, wird er zur Studentensicht weitergeleitet.

5.2.2 Studentensicht: Testate

Falls sich der Student erfolgreich eingeloggt hat, wird im die Studentensicht angezeigt. Diese ist, wie auch die Administratorsicht, in eine Haupt- und eine Detailsicht aufgeteilt. Die Detailsicht besitzt an ihrer oberen Seite eine Leiste mit den Buttons *Zwischenergebnis*, *Auswerten* und *Abmelden*, wobei die ersten zwei Buttons noch deaktiviert sind. In der Hauptsicht kann aus allen Testaten ausgewählt werden, auf welche der Student Zugriffsrechte besitzt. Nach Auswahl eines Testats werden in der Hauptsicht die dazugehörigen Übungen angezeigt.

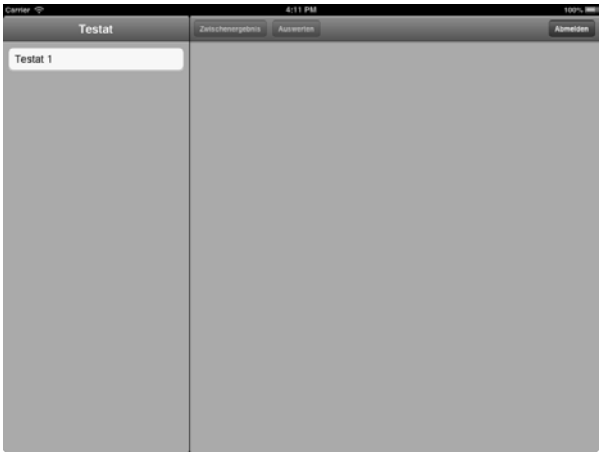


Abbildung 5.2: Auswahl eines Testats.

5.2.3 Studentensicht: Übungen

Hier werden die Übungen angezeigt, die der Student ausführen kann. Zudem hat er die Möglichkeit, sich durch Betätigen eines Buttons wieder zu den Testaten zurück zu navigieren. Die Detailsicht verändert sich nicht.

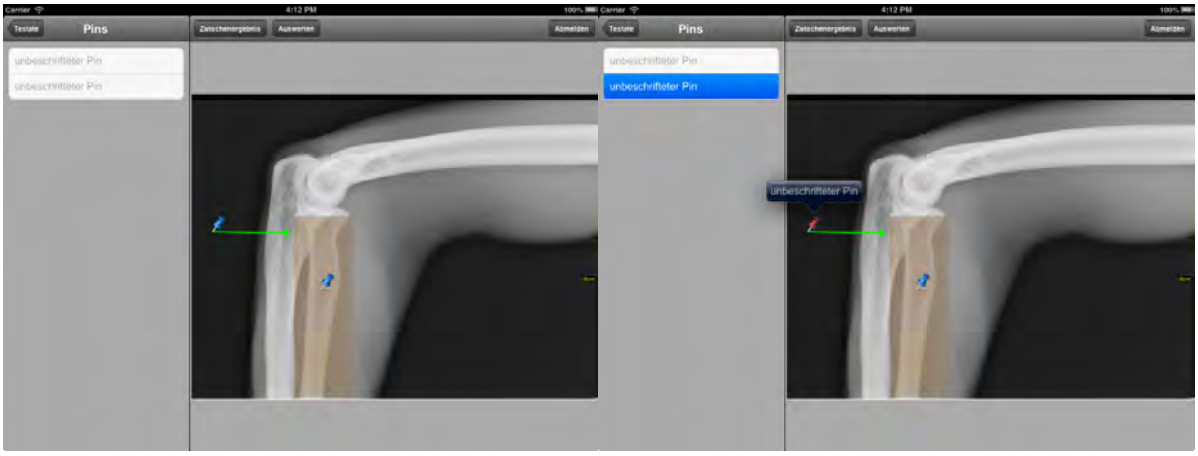
5.2.4 Studentensicht: Pins

Nach Auswahl einer Übung wird das dazugehörige Bild mit den Pfeilen, markierten Bereichen und Pins in der Detailsicht geladen. Zudem werden die Buttons *Zwischenergebnis* und *Auswerten* aktiviert. In der Hauptsicht wird für jeden Pin ein Eintrag generiert. Diese Einträge besitzen den Platzhalter "unbeschrifteter Pin", bis eine Beschriftung eingegeben wurde.
Pins können angewählt werden, indem man diese auf der Detailsicht berührt oder indem man den entsprechenden Eintrag in der Hauptsicht anwählt. In beiden Fällen erscheint ein *Popover* über dem Pin, in dem ein Textfeld mit einem Platzhalter "unbeschrifteter Pin" angezeigt wird. Klickt der Student auf das *Popover*, erscheint eine virtuelle Tastatur, mit der der Pin beschriftet werden kann. Nach Bestätigung der Eingabe wird die Beschriftung auch in der Hauptsicht übernommen.

Abbildung 5.3: Auswahl einer Übung.

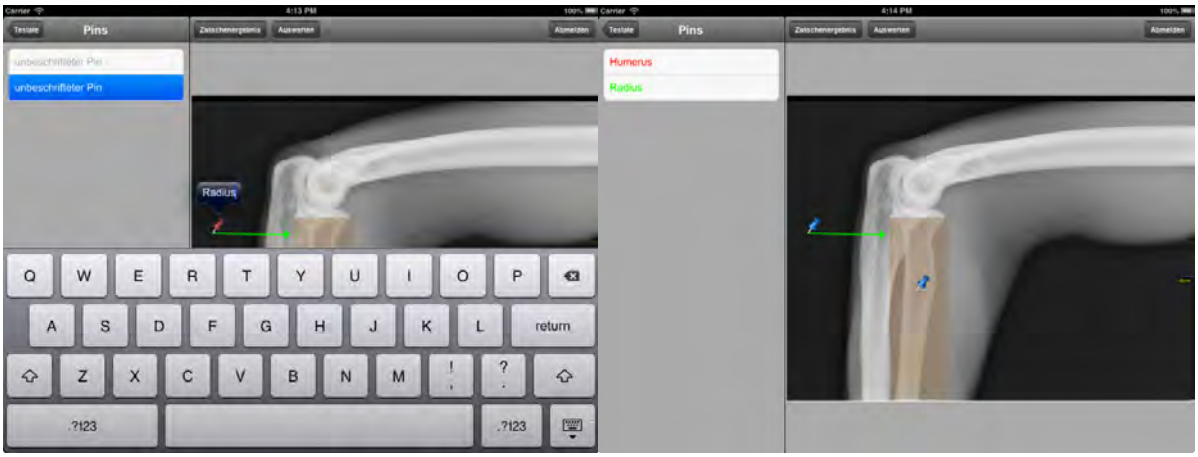
Um sich ein Zwischenergebnis präsentieren zu lassen, drückt der Student auf den entsprechenden Button. Die korrekten Beschriftungen werden jetzt grün in der Hauptsicht dargestellt, während die falschen Beschriftungen rot angezeigt werden. Durch Anwählen eines Pins, werden die Einträge in der Hauptsicht wieder schwarz dargestellt und die Pins können weiter beschriftet werden.
Um die Übung endgültig auszuwerten, betätigt man den *Auswerten*-Button. Darauf öffnet sich eine modale Sicht[2], welche die Studentensicht ausblendet und dem Studenten mitteilt, wie viele Pins richtig beschriftet wurden. Durch Berühren des Bildschirms wird wieder die Studentensicht angezeigt. Hier werden sämtliche Buttons, außer des "Abmelden"-Button, wieder deaktiviert. Richtige Beschriftungen sind in der Hauptsicht grün eingefärbt. Falsche Beschriftungen werden durch die richtigen Antworten ersetzt, unter denen die falschen Antworten des Studenten angezeigt werden. Dabei wird die ersetzte Antwort in grün und die darunter stehende falsche Antwort in rot angezeigt. Wird ein Pin ausgewählt, erscheint ein *Popover* in dem die korrekte Beschriftung des Pins angezeigt wird.
Die Detailsicht kann zurückgesetzt werden, indem man sich, mithilfe eines Buttons in der Hauptsicht, zu den Übungen zurückgeht.

[2]Sicht, die über einer anderen Sicht dargestellt wird. Während sie dargestellt wird sind keine Eingaben in der überdeckten Sicht möglich.



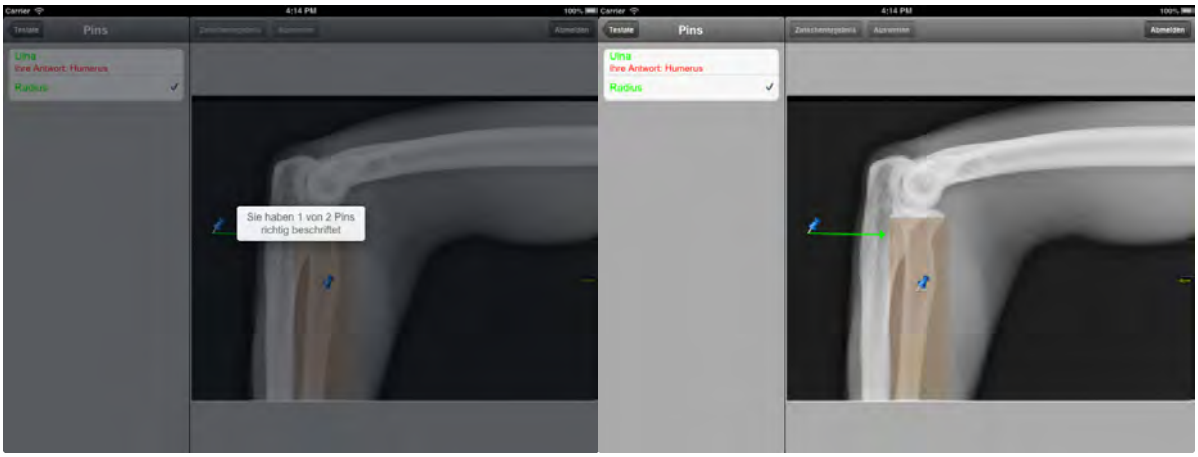
(a) Die Ansicht, nachdem eine Übung geladen wurde

(b) Auswählen eines Pins



(c) Beschriftung eines Pins

(d) Anzeigen eines Zwischenergebnisses



(e) Statistik nach der Auswertung einer Übung

(f) Studentensicht nach der Auswertung einer Übung

5.3 Zusammenfassung

Im Verlauf dieses Kapitels wurde jeweils aus Sicht eines Administrators und eines Studenten ein mögliches Anwenderszenario durchlaufen. In dem Szenario des Administrators legt dieser eine neue Übung an, welche danach im zweiten Szenario von einem Studenten ausgeführt wird.
Zusätzlich zu diesen Szenarien sind weitere Szenarien denkbar. Beispiele weiterer Szenarien werden in Tabelle 5.1 aufgeführt.

Anwender	**Szenario**
Administrator	Eine bestehende Übung wird gelöscht. Danach wird eine weitere Übung bearbeitet, indem ein neues Bild mit der Kamera aufgenommen und für die Übung verwendet wird. Darauf werden die Pins der Übung verändert, indem diese durch Zieh-Gesten auf neue Positionen gesetzt werden. Abschließend wird der Synchronisationsvorgang gestartet.
Administrator	Eine neue Übung wird angelegt. Dazu wird ein Bild aus der Fotobibliothek des iPads geladen und mehrere Pins gesetzt. Beim Synchronisieren der Übung bricht die Internetverbindung ab.
Student	Der Anwender installiert die Anwendung und startet diese zum ersten Mal. Nach dem Start meldet er sich in dem System an. Dabei verfügt er über keine Internetverbindung.
Student	Nachdem er eine Übung ausgewertet hat, navigiert sich der Anwender auf die Ebene der Testate zurück. Dort wählt er ein neues Testat aus. Darauf wählt er eine der ihm jetzt angezeigten Übungen aus, um diese zu starten.

Tabelle 5.1: Mögliche weitere Anwenderszenarien

6 Implementierung

6.1 Grundlagen der iOS-Programmierung

Im folgenden Abschnitt werden Voraussetzungen für die Entwicklung von iOS Anwendungen und Eigenschaften der iOS SDK behandelt. Bei iOS handelt es sich um das Betriebssystem der mobile Apple Geräte iPhone, iPod Touch und iPad.
Um Anwendungen auf einem iOS-Gerät zu entwickeln, ist es zuerst einmal notwendig, sich im iOS Dev Center zu registrieren. Hier erhält man das iOS SDK, welche die Entwicklungsumgebung Xcode enthält.

6.1.1 Xcode

Die "Xcode Tools" werden als Grundlage für die Entwicklung von iOS Anwendungen benötigt. Dabei handelt es sich um eine Entwicklungsumgebung oder auch *Integrated Development Environment* (IDE) und verschiedene Werkzeuge beziehungsweise Anwendungen zur Entwicklung einer iOS Anwendung. Die genauen Bestandteile von Xcode [44] werden im Folgenden vorgestellt.

6.1.1.1 Interface Builder

Der *Interface Builder* liefert die Möglichkeit, grafische Oberflächen für Anwendungen zu designen. Dazu kann der Benutzer aus einer Palette aus Objekten auswählen und diese durch *Drag 'n Drop* auf eine grafische Oberfläche ziehen. Seit Version 4.2 können mithilfe des *Interface Builders* sogenannte *Storyboards* entworfen werden. Hier können zusätzlich zu den Oberflächen auch die Übergänge zwischen den Sichten (sogenannte *segues*) erstellt werden.
Darüber hinaus können mittels *IBAction* und *IBOutlet* die Elemente der Oberflächen mit Methoden eines Objektes der Kontrollschicht verbunden werden [42].

6.1.1.2 Instruments

Instruments ist eine Anwendung, die den Entwickler bei der Analyse verschiedener Problemen unterstützt. Dabei hat er unter anderem die Möglichkeit, sich die CPU-Aktivität bei den verschiedenen Prozessen und Threads anzeigen zu lassen, den Speicherverbrauch und etwaige Speicherlecks zu überwachen, und die Datei- und Netzwerkzugriffe zu analysieren.

6.1.1.3 Simulator

Ein weiteres Feature, das bei dem Paket mitgeliefert wird, ist der Simulator. Mit ihm wird es dem Anwender ermöglicht, seine Apps, ohne dass er ein reales Gerät besitzt, in einer virtuellen Umgebung zu testen. Dabei muss allerdings beachtet werden, dass die tatsächliche Performanz der Anwendung sich mit der des Simulators unterscheiden kann, da dieser die Hardware des Computers und nicht die eines mobilen Gerätes, wie etwa eines iPads, verwendet.
Zudem können manche Funktionen, wie Telefonie oder die Kamera, nicht mit dem Simulator getestet werden.

6.1.2 iOS SDK

Zum Zeitpunkt dieser Arbeit handelte es sich bei der aktuellsten Version des iOS SDK um die Version 6.1 (Stand März 2012). Um das iOS SDK einzusetzen, benötigt man einen intel-basierten Mac mit einem Betriebssystem ab Mac OS X 10.7.4 "Lion" oder höher. Für Softwareentwickler gibt es mehrere Einschränkungen bei der Entwicklung von iOS Anwendungen gegenüber Mac OS Anwendungen. Apps können beispielsweise nur über den sogenannten App-Store installiert beziehungsweise verbreitet werden. Zudem muss man sich für das Apple iOS Development Programm anmelden, um Apps auf Geräten zu installieren, oder sie über den App-Store anzubieten.
Das iOS SDK ist in verschiedene Schichten unterteilt, wobei in jeder Ebene verschiedene Frameworks Zugriff auf Hardware- beziehungsweise Softwarekomponenten der iOS-Geräte (z.B. Kamera, Adressbuch, usw.) ermöglichen. Eine Übersicht über die verschiedenen Schichten ist in Abbildung 6.1 zu sehen. Die Abstraktionsschichten werden im Folgenden, angefangen bei der untersten Schicht, beschrieben.

6.1.2.1 Core OS

In dieser Schicht werden der virtuelle Systemspeicher, Threads, das File-System, Netzwerkfunktionen und die interne Prozesskommunikation verwaltet. Zudem werden verschiedenen Frameworks angeboten, welche dem Entwickler die Berechnungen zur Signalverarbeitung und Bildbearbeitung (*Accelerate*), den Zugriff auf externes Zubehör (*External Accessory* und *Core Bluetooth*), Sicherheit für die Daten des Anwenders (*Security* und *Generic Security Services*), und Zugriff auf System Aufrufe (*System*) bieten [10].

6.1.2.2 Core Services

Hier wird eine Vielzahl an Basisdiensten für Anwendungen und höhere Schichten angeboten. Sie enthält beispielsweise Dienste zur Verwaltung der *iCloud* [6] und die Verwaltung der Lebenszyklen von

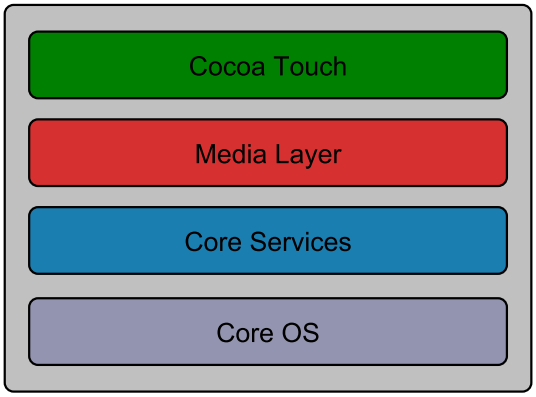


Abbildung 6.1: Die iOS Schichten

Objekten mithilfe von *ARC* (*Automatic Reference Counting*). Darüber hinaus werden Frameworks für die Verwaltung der Kontaktdaten auf dem Gerät (*Address Book*), die Implementierung von Netzwerkfunktionalität (*CFNetwork*), für die Verwaltung und Persistierung von Objektgraphen (*Core Data*), und viele weitere, angeboten. Als wichtiges Framework wäre hier außerdem noch das *Foundation* Framework zu nennen. Dieses liefert Dienste des *Core Foundation* als Objective-C-Schnittstelle und mit ihm wird die grundlegende Handhabung von Datentypen, wie Listen, Zeichenketten und das Datum beziehungsweise die Zeit, verwaltet. Zudem bietet es beispielsweise wesentliche Klassen zur Ereignisbehandlung, grafischen Gestaltung und Netzwerkkommunikation und verwaltet die Handhabung von Threads [11].

6.1.2.3 Media

Diese Schicht stellt Dienste für Grafik-, Audio- und Videofunktionalitäten bereit. Für die Grafikfunktionalitäten werden unter anderem die Frameworks *Core Graphics*, *Core Animation* und *Core Image* angeboten.
Dabei werden vom *Core Graphics* Framework (oder auch *Quartz*) eine Rendering Engine für zweidimensionale Grafiken bereitgestellt, während das *Core Animation* Framework Unterstützung bei der Animierung von Sichten und anderen Inhalten bietet und das *Core Image* Framework Funktionen zur Bearbeitung von Video- und Bildmaterial verwaltet.
Für den Einsatz von Audio Technologien kommen Frameworks wie das *Media Player* Framework und das *AV Foundation* Framework zum Einsatz. Durch das *Media Player* Framework wird der Zugriff auf die iTunes Bibliothek des Benutzers und das Abspielen von Musikdateien und Playlists erleichtert. Das *AV Foundation* Framework bietet eine Sammlung an Objective-C Interfaces zur Verwaltung der Wiedergabe und Aufnahme von Audio.
Um mit Videodateien in einer Anwendung zu arbeiten, oder zum Streamen von Videodateien, werden ebenfalls die Frameworks *Media Player* und *AV Foundation*, zusätzlich aber auch Frameworks wie das *Core Media* Framework, verwendet. Zum direkten Abspielen der Videos wird dabei das *Media Player* Framework verwendet und durch *AV Foundation* das Aufnehmen und Abspielen von Videos unterstützt. *Core Media* beschreibt die Low-Level-Datentypen, die von den Frameworks aus höhe-

ren Schichten verwendet werden und bietet Low-Level-Schnittstellen zur Manipulation von Medien [12].

6.1.2.4 Cocoa Touch

Aus Sicht der Anwendungsentwickler ist *Cocoa Touch* sicher die wichtigste Schicht, da sie, neben anderen Funktionen, die Basis für die Anwendungsentwicklung mit grafischen Oberflächen für Mac OS X und iOS bildet.
Darüber hinaus wird zur einfachen Umsetzung von grafischen Oberflächen beispielsweise das sogenannte *Storyboard* angewendet. Hier hat man die Möglichkeit, Sichten, aus vorgegebenen Komponenten, zusammenzustellen und die Navigation zwischen den Sichten zu verwalten. Zudem kann hier das sogenannte *Auto Layout* verwendet werden, um beispielsweise einen Button immer 20 Bildpunkte von dem linken Seitenrand seiner übergeordneten Sicht zu platzieren.
Das Framework, welches die Funktionen zur Erstellung von grafischen Oberflächen bietet, wird als *UIKit* bezeichnet und enthält zudem weitere Funktionen, wie etwa:

- *Unterstützung für Touch-Events* Es werden verschiedene Klassen zur Erkennung von Gesten angeboten. So können beispielsweise einfach Gesten wie Ziehen, Drücken, Wischen usw. erkannt werden und darauf reagiert werden.
- *Darstellung von Web- und Textinhalten* Es werden Sichten angeboten, welche den Entwickler bei der Darstellung bestimmter Inhalte unterstützen.
- *Zugriff auf Gerätehardware* Zugriff auf integrierte Hardware, wie beispielsweise die Kamera oder der Bewegungssensor.
- *Unterstützung für Druckvorgänge* Naheliegende Drucker können erkannt werden und Inhalte via WLAN-Verbindung gedruckt werden.

In der *Cocoa Touch*-Schicht ist außerdem das *Foundation* Framework enhalten. Dieses liefert die Basisklasse *NSObject*, auf welcher alle anderen Klassen dieser Schicht aufgebaut sind.
Zudem sind verschiedene andere Frameworks enthalten, wie das *AddressBookUI* Framework, welches das Anlegen und Bearbeiten von Kontaktdaten, die auf dem iPad gespeichert sind, ermöglicht, und das *GameKit* Framework, welches die Peer-to-Peer Verbindung zwischen mehreren Geräten via Bluetooth ermöglicht, um beispielsweise einen Voice-Chat zu unterstützen. Außerdem wird durch das *MapKit* Framework das Einbinden von Karten und Satellitenbilder in eine Anwendung ermöglicht und mithilfe des *MessageKit* das Senden und Empfangen von E-Mails [9].

6.1.3 Automatic Reference Counting

Das *Automatic Reference Counting* (*ARC*) wurde 2011 von Apple eingeführt, um Softwareentwicklern die Speicherverwaltung zu erleichtern.
Zuvor musste ein Entwickler Speicher für Objekte selbst reservieren (*allokieren*) und wieder freigeben (*deallokieren*). Machte er dabei einen Fehler, verbrauchte die Anwendung mehr Speicher als notwendig und konnte gegebenenfalls deshalb automatisch vom System beendet werden.
In Objective-C werden Objekte, anhand der Anzahl ihrer Referenzen, im Speicher gehalten. Ohne *ARC* musste der Referenzzähler eines Objektes manuell durch das Aufrufen des Befehls *retain* erhöht, beziehungsweise durch das Aufrufen von *release* vermindert werden. Erreicht die Anzahl der

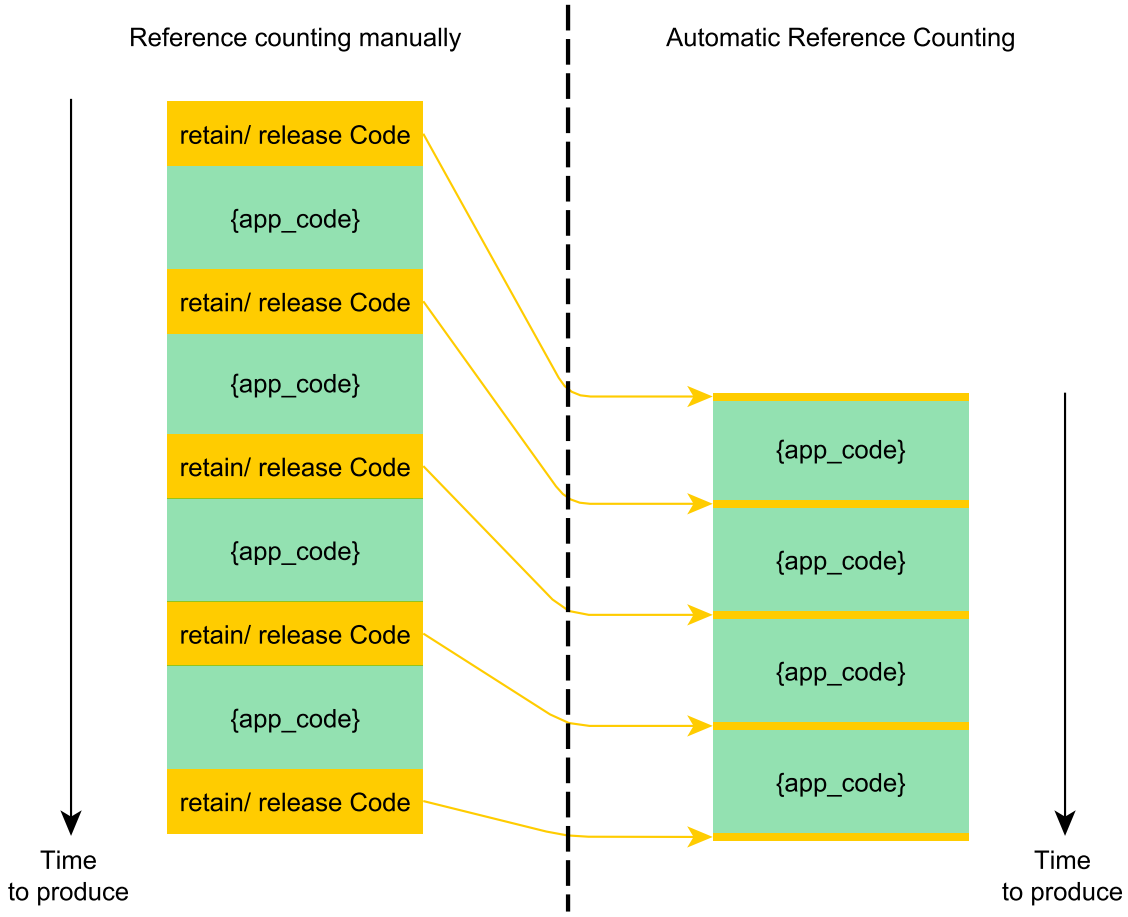


Abbildung 6.2: Code mit und ohne ARC [16]

Referenzen null, so wird das Objekt aus dem Speicher entfernt.
ARC übernimmt das Einsetzen dieser Befehle für den Entwickler. Das bedeutet, dass es sich bei *ARC* nicht um eine *Garbage Collection* handelt, wie dies beispielsweise bei Java der Fall ist. Bei einer *Garbage Collection* handelt es sich um einen Hintergrundprozess, der während der Laufzeit des Programmes Referenzen auf Objekte erhöht und vermindert und diese entfernt, sobald keine Referenzen mehr vorhanden sind, während *ARC* die Befehle zum Erhöhen und Vermindern des Referenzzählers während der Kompilierung des Programmcodes einfügt [16]. In Abbildung 6.2 wird der Unterschied zwischen Programmcode der mit *ARC* und Programmcode der ohne *ARC* geschrieben wurde dargestellt.

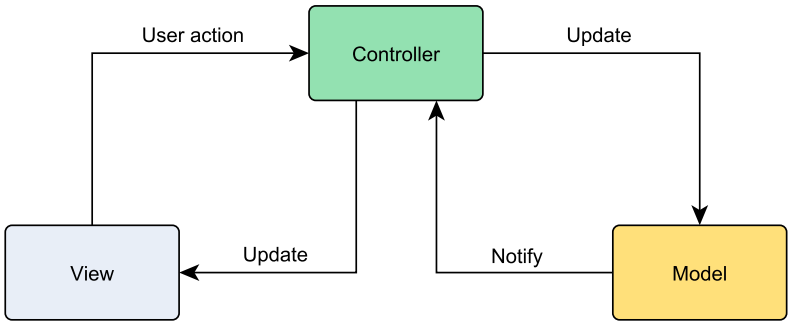


Abbildung 6.3: MVC Architektur [4]

6.1.4 Design Patterns

Die Frameworks, welche bei der Entwicklung von iOS Anwendungen zum Einsatz kommen, benutzen verschiedene Design Patterns [7]. Viele der Cocoa Technologien und Architekturen basieren auf dem *Model-View-Controller* (*MVC*) Design Pattern und erfordern, dass ihre benutzerdefinierten Objekte eine der *MVC* Rollen spielen. Darüber hinaus werden noch andere Design Patterns eingesetzt, welche nachfolgend erklärt werden:

6.1.4.1 Model-View-Controller

Das *Model-View-Controller* Pattern weist jedem Objekt in einer Anwendung eine bestimmte Rolle zu. Dabei existieren die drei Rollen *Model* (Datenmodell), *View* (Anwendungsoberfläche) und *Controller* (Geschäftslogik). Objekte der gleichen Rolle bilden eine Schicht, deshalb wird dieses Pattern oft auch als Drei-Schichten-Architektur bezeichnet. Die Rollen legen außerdem fest, wie Objekte miteinander kommunizieren.
Objekte in der Rolle eines *Models* beinhalten die Semantik einer Anwendung, das heißt hier definierten Klassen beziehungsweise Methoden liefern die Daten für die Ansichten der Anwendung.
Die Objekte der *View*-Schicht werden von der *UIView* Klasse abgeleitet und können direkt programmiert als auch mit dem Storyboard erstellt werden.
Die *Controller*-Schicht verbindet die *Model*- und *View*-Schicht. Die Programmierlogik und Kommunikation zwischen den Schichten wird hauptsächlich durch den Einsatz von *Delegation*, dem *Target-Action-Mechanismus* und *Notifications* implementiert.
Abbildung 6.3 stellt die verschiedenen Schichten und wie diese miteinander kommunizieren dar.

6.1.4.2 Delegation

Das *Delegation* Pattern wird zur Änderung von Methoden einer Klasse verwendet, ohne dabei eine Unterklasse erzeugen zu müssen. Um dies zu realisieren, besitzt ein Objekt eine Referenz auf ein anderes Objekt, um bei Bedarf Methoden dieses Objektes aufzurufen. Solche delegierten Methoden werden vor allem in der *View*-Schicht benutzt, um zwischen *View*- und *Controller*-Schicht zu kommunizieren. Ein Beispiel dafür wäre, wenn der Anwender in einer *UITableView* eine Zelle ausgewählt hat. In diesem Fall wird die *didSelectRowAtIndexPath* Methode des referenzierten Objektes in der *Controller*-Schicht aufgerufen und dort die gewünschte Logik zur Benutzereingabe ausgeführt.

6.1.4.3 Target-Action

Das *Target-Action* Pattern leitet Benutzereignisse (*Events*) in Form von Nachrichten (*Actions*) an Objekte (*Targets*) weiter. Die Methode zur Bearbeitung der Action muss dabei im *Target*-Objekt implementiert sein, da es ansonsten zum Absturz der Anwendung kommt.
Dieses Pattern wird vor allem von Elementen der Anwendungsoberfläche, die von der Klasse *UIControl* abgeleitet werden, verwendet. So wird bei Betätigung eines *UIButtons* beispielsweise eine *Action* erstellt und an einen *Controller*-Objekt weitergeleitet.
In Abbildung 6.4 wird dargestellt, wie ein solcher Target-Action-Mechanismus mithilfe des Storyboards eingerichtet wird.

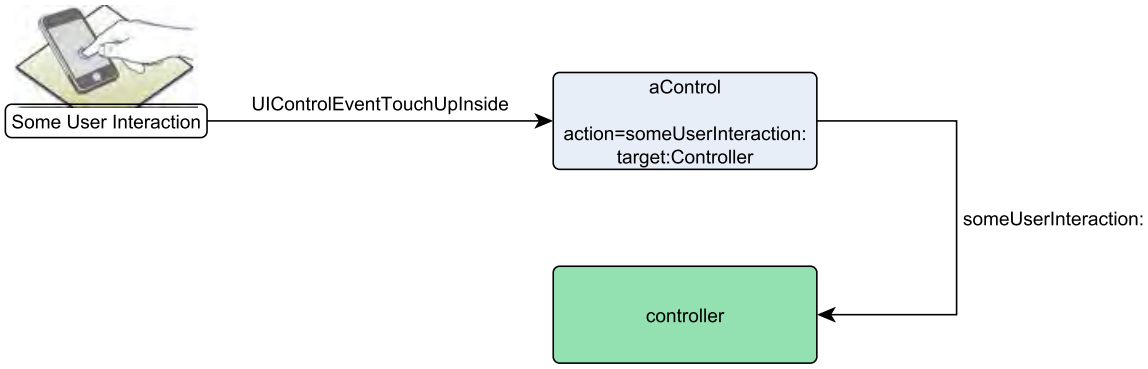


Abbildung 6.4: Target-Action-Mechanismus [3]

6.1.4.4 Observer-Pattern

Zusätzlich zu den bisher behandeltet Kommunikationsmechanismen existiert das sogenannte *Observer*-Pattern (Beobachter-Muster). Dieses wird dazu verwendet, mehrere Objekte über die Statusänderungen eines einzelnen Objektes zu benachrichtigen. Dazu wird das Objekt einer Observer-Liste hinzugefügt, indem es sich bei dem dafür vorgesehenen Objekt der Klasse *NSNotificationCenter* anmeldet. Beobachtete Objekte erzeugen nun sogenannte *NSNotifactions*, die zur Benachrichtigung anderer Objekte dienen, und verteilen diese mithilfe eines sogenannten *Broadcasts* über das *NSNotifactionCenter* an alle interessierten Beobachter [15]. Die beobachteten Objekte können dann, mithilfe der Informationen, welche sie der *NSNotification* entnehmen, angemessen darauf reagieren.
Objekte können sich auch über das sogenannte *Key-Value-Observing* bestimmte Änderungen von Eigenschaften anderer Objekte mitteilen lassen. Hierbei registriert sich der Beobachter bei dem zu beobachtenden Objekt und erhält die *NSNotification* direkt und nicht über das *NSNotificationCenter*[14].

6.2 Besonderheiten der iPad-Programmierung

Bei der Entwicklung von Software für das iPad wird, wie bei anderen mobilen iOS Geräten, auf das iOS SDK zurückgegriffen. Dadurch, dass das iPad sich durch seine Auflösung, Leistung und Größe,

von den anderen mobilen Geräten von Apple unterscheidet, gibt es besondere Funktionen, die nur für dessen Programmierung vorgesehen sind.

6.2.1 UIPopoverController

Der *UIPopoverController* wird dazu benutzt, um temporär Inhalte innerhalb eines *Popovers* zu präsentieren. Dabei wird nur ein Teil der dahinterliegenden Sicht verdeckt. Das *Popover* wird automatisch geschlossen, sobald man den Bildschirm außerhalb des *Popovers* berührt. Andere Arten zum Schließen des *Popovers* müssen explizit implementiert werden [20].

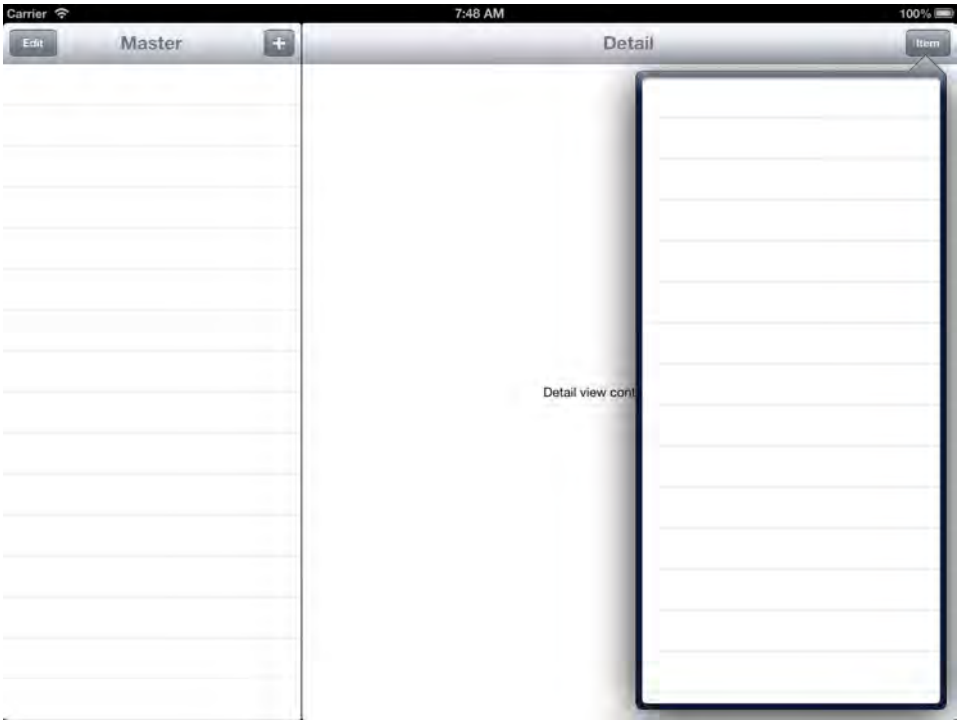


Abbildung 6.5: iPad Popover

6.2.2 UISplitViewController

Ein *UISplitViewController* dient zur Verwaltung von zwei *View Controllern*, welche für die Darstellung von Sichten dienen. Zudem verwaltet er das Verhalten der *View Controller* bei Änderung der Bildschirmausrichtung.
Die von den Controllern verwalteten Sichten werden in eine Hauptsicht (*Master View*) und eine Detailsicht (*Detail View*) aufgeteilt. Die Hauptsicht dient zur Darstellung einer Liste von Objekten, während die Detailsicht zur Anzeige genauerer Informationen, des in der Hauptsicht ausgewählten Objektes, dient.
Beim Halten des Gerätes im Querformat stellt der *UISplitViewController* die beiden *View Controller* nebeneinander dar. Hält man das iPad im Hochformat, wird die Hauptsicht ausgeblendet. Dafür wird in der Leiste der Detailsicht ein Button hinzugefügt, durch das die Hauptsicht mithilfe eines *Popo-*

vers angezeigt werden kann. Man kann dieses Verhalten jedoch beliebig verändern, indem man das gewünschte Verhalten in den delegierten Methoden des *UISplitViewControllers* implementiert [19].

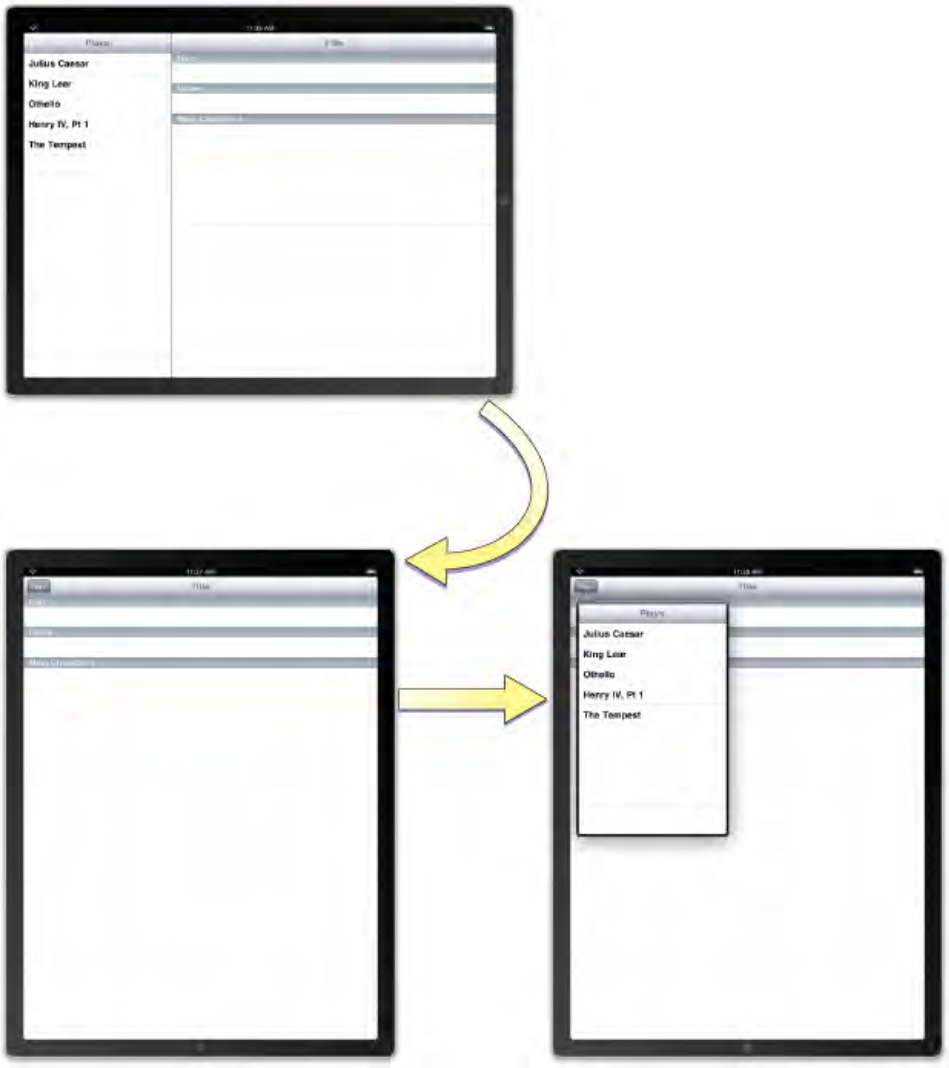


Abbildung 6.6: *UISplitViewController* im Quer- und Hochformat [19]

6.2.3 Modale Sichten

Modale Sichten werden normalerweise im Vollbild-Modus dargestellt. Da beim iPad aber eine größere Auflösung beziehungsweise ein größerer Bildschirm zur Verfügung stehen, gibt es hier drei verschiedene Stile, um modale Sichten zu präsentieren. Folgende Stile werden angeboten:

- *UIModalPresentationFullScreen* Die modale Sicht füllt den kompletten Bildschirm aus (siehe Abbildung 6.7a).
- *UIModalPresentationPageSheet* Im Hochformat füllt die modale Sicht den kompletten Bildschirm aus. Im Querformat wird ebenfalls die komplette Höhe von der Sicht ausgefüllt, während die Breite der Sicht nur der des Bildschirms im Hochformat entspricht. Die sichtbaren Teile der

Master-Sicht werden verdunkelt dargestellt, da der Benutzer nicht mit ihnen interagieren kann (siehe Abbildung 6.7b).

- *UIModalPresentationFormSheet* Die Sicht bedeckt nur teilweise die Breite und Höhe des Bildschirms und wird zentriert dargestellt. Die sichtbaren Abschnitte der dahinter liegenden Sicht sind abgeblendet und können nicht verwendet werden. Wird eine Tastatur eingeblendet, verschiebt sich die Sicht nach oben (siehe Abbildung 6.7c).
- *UIModalPresentationCurrentContext* Die modale Sicht füllt denselben Bereich des Bildschirms aus wie ihre Master-Sicht.

(a)

(b)

(c)

Abbildung 6.7: Darstellung von modalen Sichten auf dem iPad [20]

6.3 Architektur im Überblick

Im diesem Abschnitt erfolgt ein vereinfachter Überblick über die Softwarearchitektur der App und den dazugehörigen externen Komponenten. Bei den externen Komponenten handelt es sich um einen bestehenden Web Service des Instituts für Datenbanken und Informationssysteme und einen Web Service, welcher explizit für diese Arbeit implementiert wurde.
Der Web Service des Instituts wird dabei benutzt, um mithilfe der Benutzerdaten zu verifizieren, dass es sich bei dem Anwender um einen Studenten oder eine Lehrkraft der Universität handelt. Dies geschieht, indem der *SoapAPIClient* eine entsprechende SOAP-Nachricht an den Web Service gesendet wird, welcher wiederum die Information, durch eine LDAP[1] Anfrage an den Verzeichnisdienst des Kommunikations- und Informationszentrum (kiz) der Universität Ulm, erhält.
Für die Umsetzung des anderen Web Services wurde ein Apache Server aufgesetzt auf dem ein in PHP erstellter RESTful Web Service ausgeführt wird, der wiederum auf eine MySQL Datenbank zugreift. Mit ihm kann, wie es bei REST üblich ist, mithilfe von HTTP-Requests kommuniziert werden. Die Ankopplung an den Web Service geschieht dabei über den *RestAPIClient*. Er liefert nach erfolgreicher Authentifizierung die Zugriffsrechte und die Rolle des Benutzers.
Auf der Seite der App sind direkte Zugriffe auf die Datenbank nicht möglich. Indirekt werden aber Datenbankoperationen mithilfe des Core Data Frameworks 6.4.2, welches durch die Klasse *CoreDataController* implementiert wurde, ausgeführt. Zur Persistierung der Daten dient dabei eine SQLite Datenbank.
Um die lokalen Daten mit denen des Servers zu synchronisieren, wurde die Klasse *SyncEngine* implementiert. Die SyncEngine-Instanz gleicht dabei die Daten, welche sie mit dem *CoreDataController* aus der Datenbank ausliest, mit denen der *MySQL* Datenbank des RESTful Web Services ab.
Der *LoginViewController* nimmt die Eingabe der Benutzerdaten entgegen und wertet diese mithilfe der Client-Objekte aus.
Die Sichten der verschiedenen Rollen, deren Controller mithilfe des *PASplitViewController* verwaltet werden, sind hier aus Gründen der vereinfachten Darstellung nicht vorhanden. In der Rolle als Administrator können neue Inhalte erstellt werden und mithilfe des *CoreDataControllers* in eine SQLite Datenbank gespeichert werden. Zudem werden Bilddateien direkt in das Applikationsverzeichnis (*Application Directory*) der Anwendung gespeichert.

[1] Ein Protokoll zur Abfrage und Modifikation von Informationen eines Verzeichnisdienstes

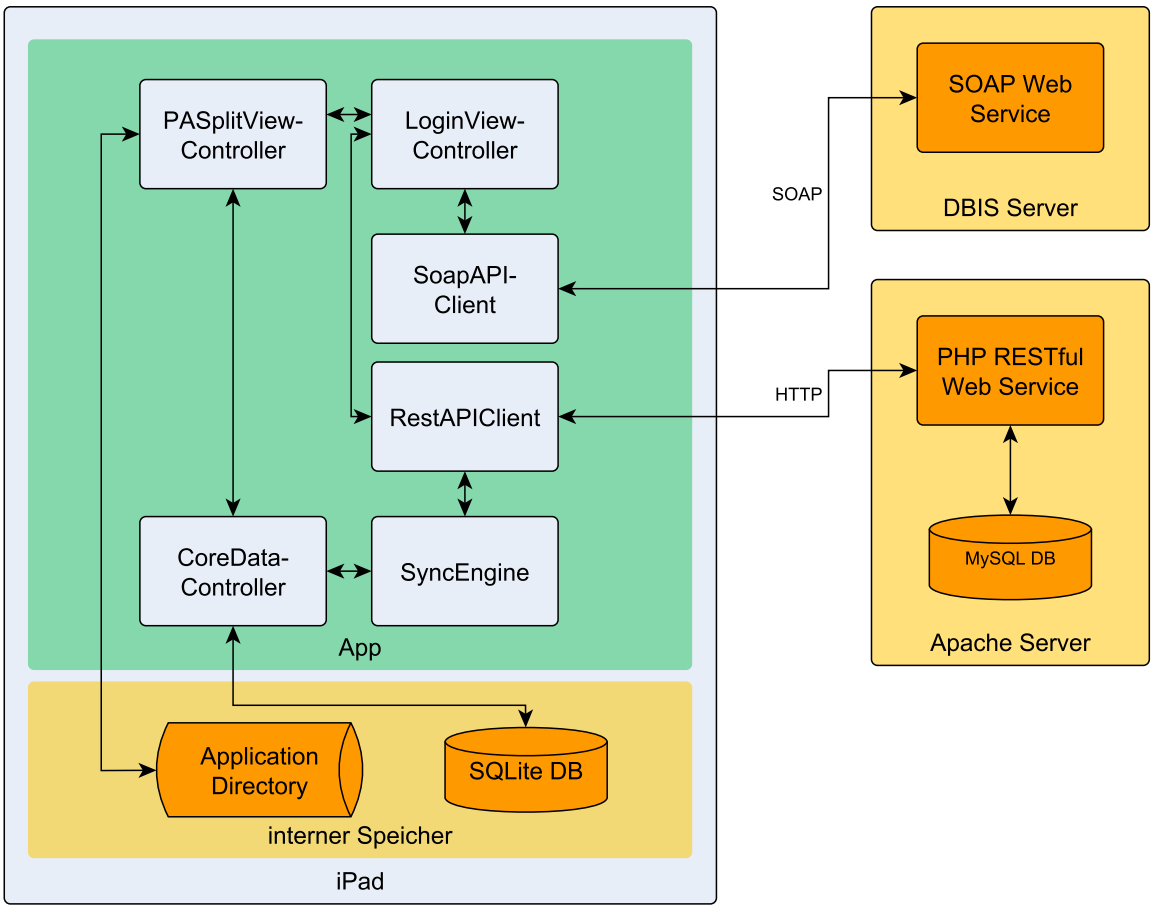


Abbildung 6.8: Die Softwarearchitektur der App

6.4 Verwendete APIs, Frameworks und Libraries

Hier werden Frameworks und Libraries vorgestellt, die in der Implementierung verwendet wurden.

6.4.1 AFNetworking

AFNetworking ist eine Open-Source-Bibliothek für iOS und Mac OSX, die verschiedene Netzwerkoperationen (wie beispielsweise HTTP- XML- oder JSON-Requests) bietet und unterstützt. Sie beruht auf *NSURLConnection*, *NSOperation* und anderen Klassen des Foundation Frameworks, die standardmäßig Verwendung finden, um über Netzwerk zu kommunizieren (beispielsweise zur Erstellung von HTTP-Requests).
AFNetworking benutzt sehr häufig sogenannte *block*-Codes. Diese wurden in Mac OSX 10.6 beziehungsweise iOS 4.0 neu eingeführt. Im Allgemeinen sind *blocks* nur ein Stück Code das zu einem späteren Zeitpunkt ausgeführt werden kann. Es handelt sich dabei um First-Class-Funktionen [38], welche im Prinzip wie Objective-C Objekte behandelt werden und deshalb auch von Methoden zurückgegeben, als Parameter gesetzt oder Variablen zugewiesen werden können. Der Vorteil ist, dass man keine Callbacks an *Delegates* ausführen muss oder das Events mit dem *NSNotificationCenter* erzeugt, was meist einen weit verstreuten und unübersichtlichen Code zur Folge hat. Der Aufbau eines *block*-Codes wird in Abbildung 6.9 dargestellt.
Ein weiterer Vorteil von AFNetworking ist seine Modularität. So sind in AFNetworking eine Reihe von spezialisierten Klassen enthalten, die für wiederkehrende Aufgaben, wie das Abrufen und Verarbeiten von JSON-Daten von einem Webdienst, genutzt werden können. Darüber hinaus ist es leicht, AFNetworking zu erweitern, um es individuell an die Anforderungen der zu entwickelnden App anzupassen.

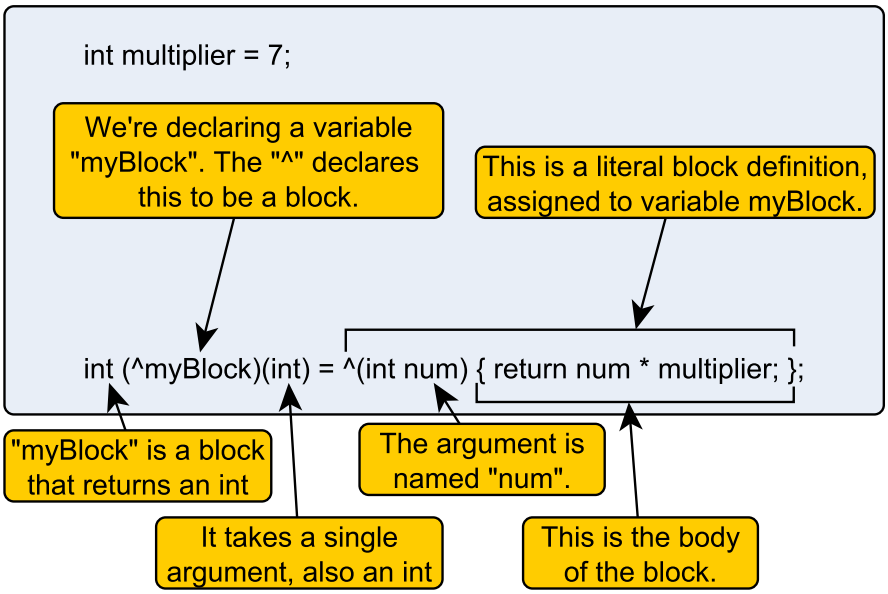


Abbildung 6.9: Aufbau eines block-Codes [2]

6.4.2 Core Data

Core Data ist ein Framework, dass die Verwaltung und Persistierung eines komplexen Datenmodells durch den Einsatz von Objektgraphen erleichtert. Standardmäßig wird im Hintergrund für die Persistierung eine *SQLite* Datenbank benutzt. Es können jedoch auch andere Datenbanken bzw. Persistierungsmöglichkeiten verwendet werden.
Das Framework bietet folgende unterstützende Funktionen:

- *Automatische Verwaltung von Beziehungen* Beziehungen zwischen den Daten werden automatisch verwaltet (beispielsweise kaskadierendes Löschen bei entsprechenden Beziehungen).
- *Logging von Änderungen und Undo Unterstützung* Core Data zeichnet Veränderungen an Daten auf und ermöglicht so, diese rückgängig zu machen.
- *Migration des Datenschema* Es bietet Unterstützung bei der Migration zu einem neuen Datenschema, falls das Datenmodell für zukünftige Entwicklungen angepasst werden muss.
- *Integration in die Controller-Schicht* Datenelemente des Datenmodells werden in Klassen umgewandelt, deren Objekte direkt in der Kontrollschicht verwendet werden können.
- *Filterung von Objekten* Die Klasse *NSPredicate* kann verwendet werden, um bestimmte Objekte anhand vorgegebener Kriterien zu laden, ohne dabei komplizierte SQL Abfragen anwenden zu müssen.
- *Speicheroptimierung* Das Framework lädt nur die Daten in den Speicher, welche gerade gebraucht werden (lazy loading).

6.4.3 QuartzCore

Das *Quartz Core* Framework bietet Funktionen, um die Ebenen (*CALayer*) zu verändern und Animationen (*CAAnimation*) zu erzeugen. Objekte der Klasse *CALayer* sind die Ebenen, aus denen eine Sicht (*UIView*) aufgebaut ist. Sie enthalten Attribute (wie beispielsweise die Position, Dimension, Hintergrundfarbe usw.), die animiert werden können. Das letztendliche Zeichnen wird durch eine *UIView* ausgeführt, in welcher die Ebenen gespeichert werden. Das bedeutet, dass es sich bei *CALayer* um Datencontainer handelt, welche die Werte der Parameter einer Sicht repräsentieren, diese aber nicht darstellen.
Die Klasse befindet sich also, im Sinne des MVC Patterns (siehe Abschnitt 6.1.4.1), in der Model-Schicht.
Es werden zudem verschiedene, von *CALayer* abgeleitete Klassen, angeboten, um den Entwickler bei der Anwendung von Ebenen in den verschiedenen Sichten zu unterstützen. So gibt es unter anderem *CATextLayer* zur Unterstützung bei Sichten mit Textinhalten (*UITextViews*, *UITextFields* usw.) und *CAScrollLayer* für scrollbare Sichten (*UIScrollViews*).
Animationen werden dazu verwendet, die Eigenschaften einer Ebene über einen gewissen Zeitraum hinweg zu verändern, und so optische Effekte zur Verbesserung der Anwendungsoberfläche zu erzeugen.

6.4.4 Core Graphics

Core Graphics ist eine API (Application Programming Interface), welche auf *C* basiert und den Anwender bei der Verwaltung des grafischen Kontextes und dem Zeichnen von zweidimensionalen Objekten unterstützt. Dabei erfolgt das Zeichnen nicht auf Grundlage von Pixeln, sondern anhand geometrischer Punkte. Deshalb kann Code, der auf Frameworks dieser API basiert, auf jedem iOS Gerät, unabhängig von dessen Auflösung, eingesetzt werden.
Es werden beispielsweise Funktionen, wie das Einsetzen von Gradienten, das Zeichnen von geometrischen Strukturen und Transformationen angeboten. Um geometrische Strukturen zu zeichnen, werden zuerst sogenannte Pfade (*CGPaths*) festgelegt und danach gerendert.

6.5 Web Service

Damit jeder Benutzer mit denselben Daten arbeiten kann, müssen diese mit extern gespeicherten Daten synchronisiert werden. Die externe Speicherung erfolgt, indem die Daten an einen Web Service geschickt werden, welcher als Schnittstelle zu einer externen MySQL-Datenbank, beziehungsweise eines externen Speichers für Bilddateien fungiert. Außerdem liefert der Web Service nach erfolgreicher Authentifizierung des Benutzers, seine Benutzerrolle und die verschiedenen Bereiche, zu denen er Zugriff hat.
Zur Übertragung von Daten zwischen der App und dem Web Service, wurde das JSON Datenformat gewählt. Das Format zeichnet sich dadurch aus, dass es aus einer für den Menschen und Maschinen einfach lesbarer Form besteht und in den gängigen Programmiersprachen bereits Parser dafür existieren. JSON unterstützt die Datentypen "boolescher Wert", "Zeichenkette", "Zahl", "Array", "Objekt" und "Nullwert". Ein Beispiel für die Übertragung von Daten im JSON Format wird in Listing 6.1 gezeigt.
Das Ziel war es, ein möglichst einfachen und leichtgewichtigen Web Service zu entwerfen. Deshalb wurde für die Umsetzung eine REST (*Representational State Transfer*) Architektur, welche in der Skriptsprache PHP implementiert wurde, verwendet.
Der Vorteil dieser Architektur ist, dass für die Kommunikation zwischen dem Server und dem Client direkt das HTTP-Protokoll [41] verwendet wird, anstatt einem kompliziertem Mechanismus, wie es etwa bei *CORBA* [40] der Fall wäre, oder ein darauf aufbauendes Protokoll, wie beispielsweise *SOAP*.
Um die grundlegenden Datenbankoperationen "Erstellen", "Lesen", "Aktualisieren" und "Löschen" ("Create", "Read", "Update", "Delete", kurz *CRUD*) umzusetzen, wird auf die HTTP-Request-Methoden GET, POST, PUT und DELETE zurückgegriffen. In Tabelle 6.1 werden die verschiedenen Methoden des Web Service dargestellt.
Der Web Service wurde in PHP implementiert, da in diese Sprache eine breite Datenbankunterstützung [35] und Internetprotokolleinbindung, sowie eine große Bandbreite an Funktionsbibliotheken angeboten wird. Zudem gibt es in PHP viele Frameworks, welche die Implementierung eines RESTful Web Service unterstützen. Verwendet wurde hier das SLIM Framework [32]. SLIM ist ein sogenanntes "Micro-Framework", d.h. es bietet nicht den Funktionsumfang anderer Frameworks, wie beispielsweise CodeIgniter [23], Zend [45] usw., ist aber gerade deshalb auch nicht so komplex und schnell zu erlernen.

```
{
 "title":"3.Testat: Brust- und Bauchsitus, Retrositus, kleines Becken",
 "objectId":"08E2BB08-6BCC-4818-8FD6-8843114E7FB1",
 "date_modified":"19-02-2013 13:39:25466",
 "contentId":"0"
}
```

Listing 6.1: Daten eines Testats, die im JSON Format dargestellt werden

Methode	URL	Beschreibung
GET	http://medapp.dbis.info/PrepApp-Webservice/classes/[classname]{?ID}	Liefert eine HTTP-Response mit den Objekten der geforderten Klasse "classname" (z.B. Testate). Optional können noch die *UUID*, der bereits vorhandenen Objekte der Klasse als Parameter übergeben werden, um nur Objekte, welche lokal nicht vorhanden sind, zu laden.
POST	http://medapp.dbis.info/PrepApp-Webservice/classes/[classname]	Erstellt ein neues Objekt der Klasse "classname" in der Datenbank des Webservers und liefert als Response das erstellte Objekt zurück.
PUT	http://medapp.dbis.info/PrepApp-Webservice/classes/[classname]	Aktualisiert ein vorhandenes Objekt der Klasse "classname" in der Datenbank des Webservers und liefert als Response das veränderte Objekt zurück.
DELETE	http://medapp.dbis.info/PrepApp-Webservice/classes/[classname]	Löscht ein vorhandenes Objekt der Klasse "classname" in der Datenbank des Webservers.
GET	http://medapp.dbis.info/PrepApp-Webservice/role/[user]	Liefert die Benutzerrolle des Benutzers "user" und auf welche Inhalte zugegriffen werden darf. Wenn der Benutzer nicht vorhanden ist, wird der Zugriff verweigert.

Tabelle 6.1: Methoden des RESTful Web Service

6.6 Datenmodell

Die Persistenz sämtlicher Daten, ausgenommen der Bilddateien, wird in mehreren Datenbanken sichergestellt. Das Datenmodell der App teilt sich dabei in eine lokale Datenbank, in der die Daten auf dem Gerät persistent gehalten werden, und einer externen Datenbank, anhand derer lokale Daten mit den Daten anderer Benutzer synchronisiert werden können, auf.
Für die lokale Datenhaltung wird als Backend eine SQLite-Datenbank verwendet, welche durch das Core Data Framework (siehe Abschnitt 6.4.2) verwaltet wird.
Zum Speichern der Bilddateien wurden zwei mögliche Konzepte entwickelt. Das erste Konzept sah keine lokale Speicherung der Bilddateien vor. Dies hat den Vorteil, dass diese nicht synchronisiert werden müssen. Zudem kann die App, ab einer gewissen Anzahl an Übungen, viel Speicherplatz des Gerätes in Anspruch nehmen. Der Nachteil ist allerdings, dass die Anwendung unverhältnismäßig lange Ladezeiten hat, da bei jedem Start sämtliche Bilder von dem Server übertragen werden müssen.
Bei der zweiten Alternative werden die Bilddateien lokal gespeichert. Der Vorteil ist, dass die Ladezeiten somit minimiert werden, da kein ständiges Nachladen der Bilder bei Programmstart nötig ist. Es muss allerdings zusätzlich Logik zur Synchronisierung der Bilddateien implementiert werden.
Da die Größe der Bilddateien auf 2MB beschränkt ist (siehe dazu Abschnitt 6.8.4.1), und aus den Anforderungen hervorging, dass nur eine begrenzte Anzahl an Übungen existiert, fiel die Auswahl auf die zweite Methode der lokalen Speicherung. Was zudem für diese Methode spricht, ist das bei einer mobilen Anwendung nicht immer von einer konstanten Internetverbindung ausgegangen werden kann, und so wäre es teilweise gar nicht möglich, auf die Testate zuzugreifen.
Wie anfangs erwähnt, werden Bilddateien nicht direkt in der Datenbank gespeichert. In den Core Data Release Notes wird das direkte Speichern (beispielsweise als BLOB[2] oder Text) in der Datenbank nur bei sehr kleinen Bilddateien empfohlen (siehe [5]). Da in der App als oberste Grenze der Dateigröße eines Bildes 2MB festgelegt wurden, empfahl es sich deshalb, die Bilder im Dateisystem des iPads beziehungsweise auf dem Web Server zu speichern und nur als Referenzen, in Form von URIs, in der Datenbank zu halten.

Als Backend der externen Datenhaltung dient eine MySQL-Datenbank, welche durch den RESTful Web Service verwaltet wird. Das externe Datenmodell entspricht dem lokalen Datenmodell mit der Erweiterung, dass hier auch die Benutzer mit den entsprechenden Zugriffsrechten gespeichert werden.

Das Datenmodell wurde anhand der in Kapitel 3 formulierten Anforderungen konzipiert und ist in Abbildung 6.10 zu sehen.
Jedes Datenelement besitzt einen Zeitstempel, der angibt, wann dieses zuletzt verändert wurde, einen Synchronisierungsstatus, und eine *UUID* (*Universally Unique Identifier*)[3], welche zur Synchronisierung mit der externen Datenbank benötigt werden (siehe Abschnitt 6.7).
Die Objekte der Klasse *PrepfileSection* (Preparation File Section) stellen Testate dar. Sie besitzen eine sogenannte *contentId*, durch welche der Zugriff auf das Testat geregelt wird und ihre Bezeichnung.
PrepFile (Preparation File) stellt eine Übung dar. Zur Umsetzung der Synchronisierungsstrategie besitzt jede Übung einen Schalter, welcher anzeigt, ob die dazugehörige Bilddatei geändert wurde. Darüber hinaus besitzt es eine Referenz auf die aktuell verwendete Bilddatei und den Namen der Übung.

[2] In Datenbanken können große Datenmengen als sogenannte *Binary Large Objects* (BLOBs) abgelegt werden. Dabei handelt es sich um nicht weiter strukturierte Objekte beziehungsweise Felddaten
[3] Eine global eindeutige Zeichenfolge zur Identifizierung von Objekten

Ein *HighlightedLayer* beschreibt einen hervorgehobenen Bildbereich. Hier wird nur der Rahmen des zu hervorhebenden Bereichs gespeichert.
Die Pfeile werden als *Drawables* gespeichert. Sie enthalten den Bezierpfad, der sie beschreibt (siehe Abschnitt 6.8.4.2). Da man theoretisch auch andere Grafiken als Pins verwenden könnte, wurde das dazugehörige Datenelement als *Draggable* bezeichnet. Hier wird der Typ des Pins (klein, groß) und seine Position gespeichert.
Die gültigen Beschriftungen zu den Pins werden als "Label" modelliert und enthalten diese als Attribut.

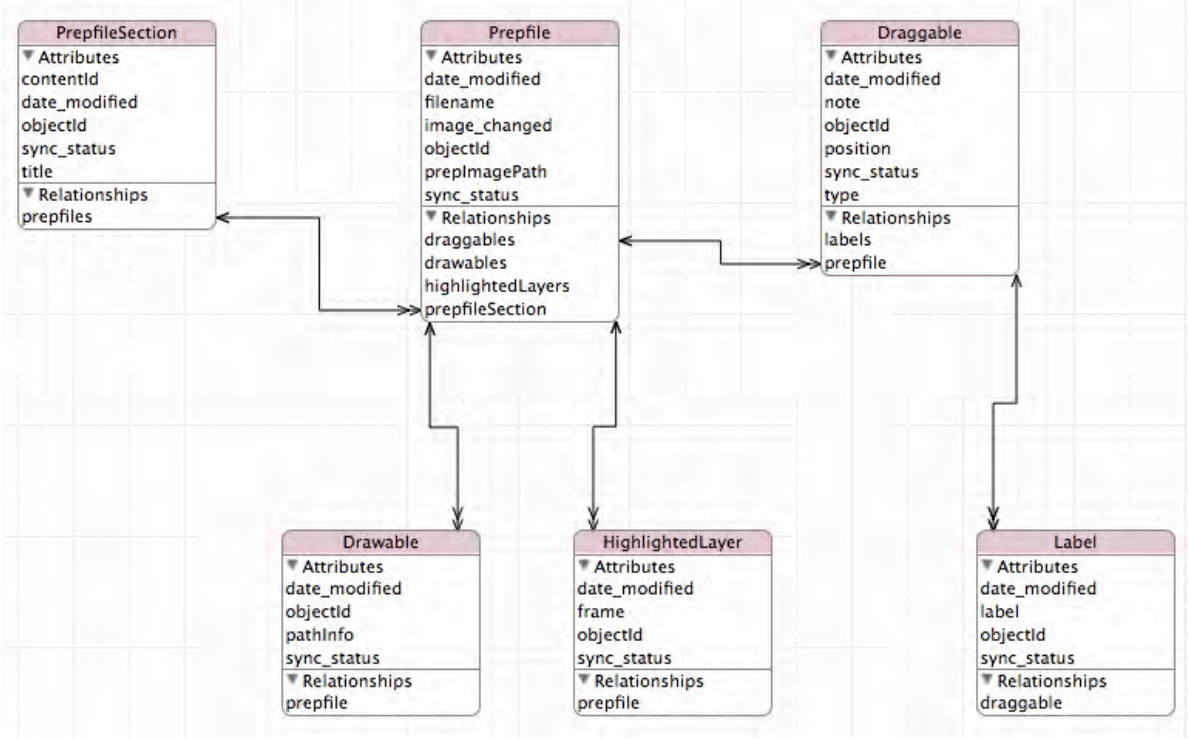


Abbildung 6.10: Das Datenmodell dargestellt als Objektgraph von Core Data

6.7 Synchronisierung der Daten

Damit die Daten zwischen der externen Datenbank und den lokalen Datenbanken synchron gehalten werden können, musste eine Möglichkeit zur Synchronisierung implementiert werden. Da es sich bei der Implementierung um einen Prototyp handelt und aus den Anforderungen bekannt war, dass es eine begrenzte Anzahl an Testate beziehungsweise Übungen geben würde, wurde die sogenannte "snapshot"-Strategie [34] zur Replikation der Daten bei der Synchronisierung verwendet. Es wird dabei zwischen zwei verschiedenen Synchronisierungen unterschieden.

Synchronisierung bei Programmstart

Bei dieser Synchronisierung wird von der externen Datenbank ein "Schnappschuss" erzeugt, das heißt alle Daten der externen Datenbank werden durch den Web Service an den anfragenden iPad über-

tragen. Auf dem iPad werden noch nicht vorhandene Daten erstellt und vorhandene Daten erneuert. Eine Ausnahme bilden hier die Bilddateien. Da es sehr zeit- und ressourcenintensiv wäre, bei jedem Programmstart sämtliche Bilddateien zu übertragen, erfolgt dies nur bei Übungen, bei denen sich das Bild geändert hat. Um festzustellen, wann dies der Fall ist, werden die Bilder anhand von *UUIDs* benannt. Entspricht der Dateiname also einer anderen *UUID* als der lokal vorhandenen, wird ein zusätzlicher HTTP-Request an den Web Service gesendet, um die neue Bilddatei herunterzuladen. Die alte Datei wird dabei gelöscht. Im Datenmodell ist für jedes Datenelement das letzte Änderungsdatum als Attribut vorhanden, um möglicherweise zukünftig andere Daten-Replikations-Strategien für den Web Service zu implementieren (siehe dazu auch Kapitel 8).

Synchronisierung in der Administratorsicht

In der Administratorsicht kann der Benutzer, durch einen Button in der oberen Leiste der Detailsicht, einen Synchronisierungsvorgang starten. Zudem erscheinen die Bezeichnungen der Datenelemente in den Hauptsichten dieser Sicht, je nach Ausprägung ihres Synchronisierungsstatus, farblich hinterlegt. Datenelemente haben die möglichen Status *ObjectSynced*, *ObjectCreated*, *ObjectChanged* und *ObjectDeleted*.
Werden in der Administratorsicht neue Datenelemente angelegt (beispielsweise Testate, Übungen usw.), erhalten diese den Status *ObjectCreated* und erscheinen grün hinterlegt in der jeweils zur Datenebene gehörenden Hauptsicht.
Werden vorhandene Datenelemente verändert, wird der Status zu *ObjectChanged* verändert und die dazugehörende Bezeichnung in der Hauptsicht ebenfalls grün hinterlegt.
Objekte, die bereits synchronisiert wurden, sind weiß hinterlegt und haben den Status *ObjectSynced*.
Will man ein Datenelement löschen, wird dies in der Tabelle rot hinterlegt und der Status *ObjectDeleted* gesetzt (siehe Abbildung 5.1a).
Bei erfolgreicher Beendung eines Synchronisierungsvorganges erhalten sämtliche Datenelemente den *ObjectSynced*-Status. Falls mehrere Benutzer lokal dasselbe Datenelement bearbeiten und schließlich synchronisieren wollen, wird nach dem "Last Write Wins"-Prinzip verfahren. Es wird also ausschließlich die Version des Datenelementes in die externe Datenbank geschrieben, welche zuletzt gesendet wurde. Diese Strategie erschien sinnvoll, da Datenelemente in der Regel nur durch einen Benutzer, beziehungsweise von einem Gerät aus, bearbeitet werden. Daher kann man beispielsweise davon ausgehen, dass eine Übung nicht simultan von mehreren Geräten aus verändert wird.

6.8 Die grafische Benutzeroberfläche

In diesem Abschnitt werden die verschiedenen Sichten und die Werkzeuge, welche zur Bearbeitung von Übungen eingesetzt werden können, vorgestellt. In der folgenden Übersicht wird das Storyboard mit den verschiedenen *View Controllern* und deren Übergängen dargestellt. Der Aufruf des modalen *View Controllers*, welcher den Anmeldungsvorgang verwaltet (2) und die Übergange von dem *Split View Controller* (1) zu den jeweiligen Benutzersichten ((3) Studentensicht, (4) Administratorsicht) wurden programmatisch umgesetzt, da ein *Split View Controller* immer der *Root View Controller* sein sollte (siehe [19]) und verschiedene Detailsichten für *Split View Controller* hier nicht unterstützt werden. (5) ist der *Navigation Controller*, der die Hauptsichten der Administratorsicht verwaltet, während (6) die Hauptsichten der Studentensicht und deren Übergange verwaltet. (7) ist die modale Sicht, welche dem Studenten nach der Auswertung einer Übung angezeigt wird.

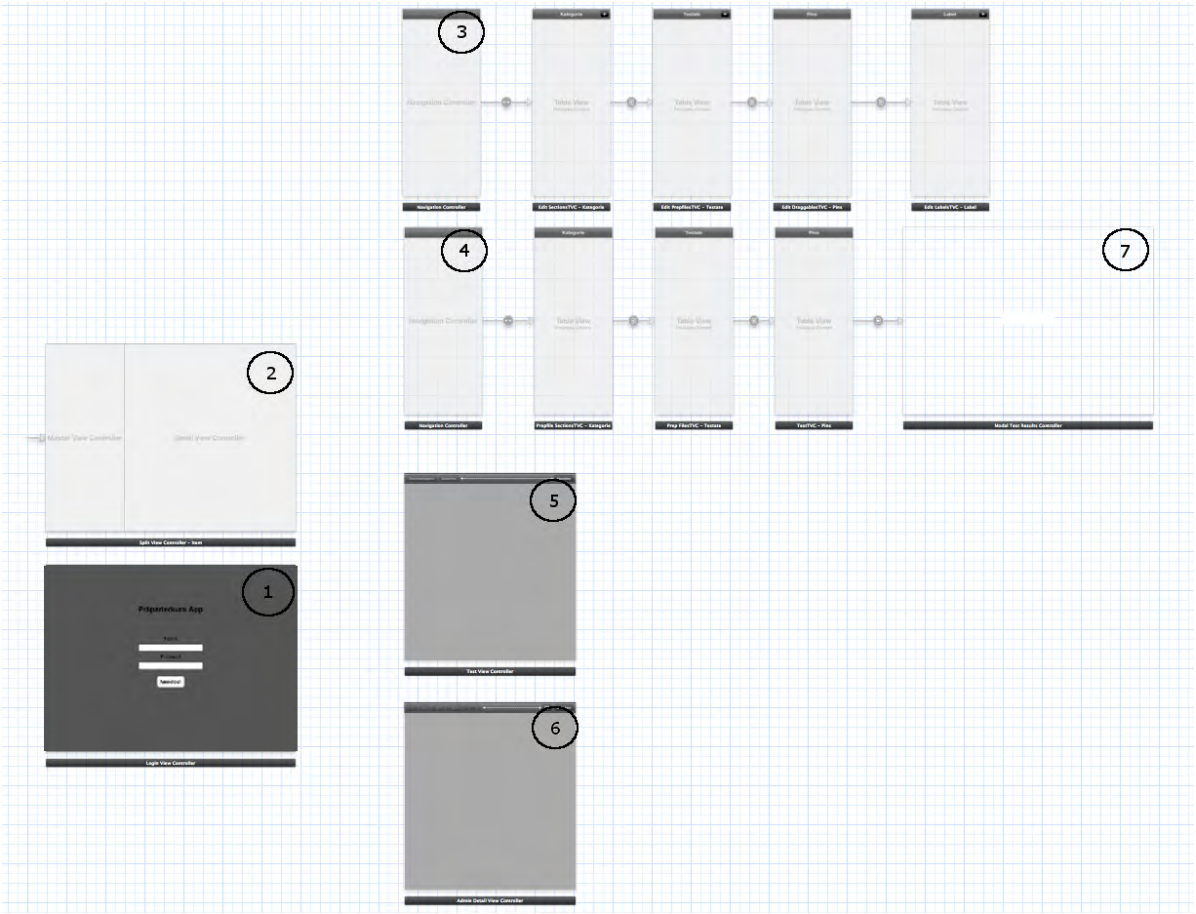


Abbildung 6.11: Das Storyboard der App

6.8.1 PASplitViewController

Der *PASplitViewController* (siehe Abbildung 6.11 (1)) erbt von der Klasse *UISplitViewController* (siehe Abschnitt 6.2.2) des UIKit-Frameworks und erhält als zusätzliche Funktion das Anzeigen eines modalen *View Controllers*, sobald er dem Benutzer angezeigt werden soll. Wie in Abbildung 6.11 zu erkennen ist, handelt sich bei ihm um den *Root View Controller*, das heißt er ist der erste *View Controller*, der bei Programmstart angezeigt wird. Dies wurde so implementiert, da laut dem "View Controller Catalog for iOS" [18] ein *Split View Controller* immer als *Root View Controller* genutzt werden sollte. Da der Benutzer sich aber zuerst einmal anmelden muss, sollte eigentlich der *LoginViewController* als *Root View Controller* benutzt werden. Dies wurde gelöst, indem beim Laden des *PASplitViewController* sofort der *LoginViewController* als modale Sicht darüber angezeigt wird.

6.8.2 LoginViewController

Nachdem die Anfangssynchronisation abgeschlossen wurde, erhält der Benutzer Zugriff auf diese Sicht (siehe Abbildung 6.11 (2)).
Der Controller verwaltet zwei Textfelder, die zur Eingabe des Benutzernamens und des dazu-

gehörigen Passworts dienen. Darüber hinaus verwaltet er einen Button, der, nach Eingabe der Benutzerdaten, den Authentifizierungsprozess startet (siehe Abbildung 6.12). Dafür werden die Benutzerdaten an einen sogenannten *SoapAPIClient* übergeben, welcher diese manuell in eine *SOAP*-Nachricht einbindet und danach an den Web Service des Instituts für Datenbanken und Informationssysteme sendet. Die *SOAP*-Nachricht, welche durch den Web Service zurückgesendet wird, enthält Informationen darüber, ob es sich bei den eingegebenen Benutzerdaten um einen Angestellten beziehungsweise Studenten der Universität Ulm handelt. Um diese Informationen aus der *SOAP*-Nachricht zu erhalten, wurde ein XML Parser implementiert. Die dazu angewendete Methode des Parser wird in Listing 6.2 vorgestellt.
Wurde der Benutzer nicht erfolgreich authentifiziert, bricht der Vorgang ab und es wird eine Fehlermeldung angezeigt. Bei erfolgreicher Authentifizierung wird ein HTTP-Request an den in dieser Arbeit implementierten Web-Service 6.5 gesendet, welcher nur den Benutzernamen als Parameter enthält. Der Web Service schickt die dem Benutzernamen zugewiesenen Zugriffsrechte und die Rolle des Benutzers zurück, worauf ihm dementsprechend eine Hauptsicht angezeigt wird (siehe nachfolgende Abschnitte). Ein Sequenzdiagramm des Vorganges wird in Abbildung 6.13 dargestellt.

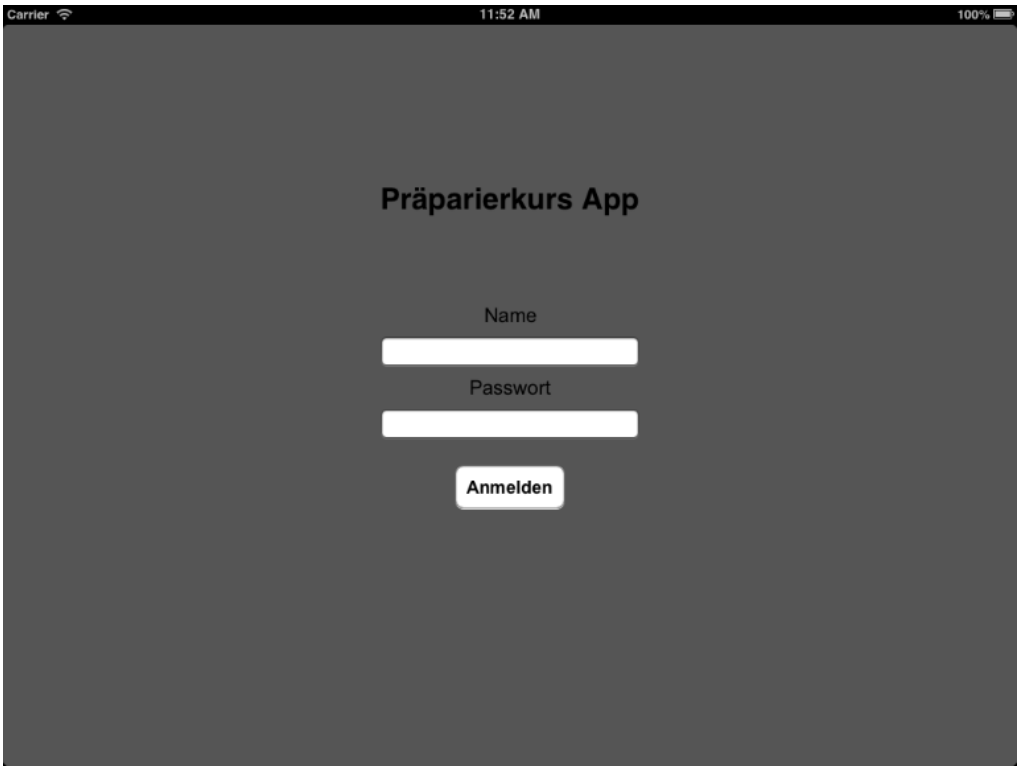


Abbildung 6.12: Die Sicht des *LoginViewController*

```
1            .
2            .
3            .
4
5  if([elementName isEqualToString:@"soapenv:Envelope"]) {
6    // We reached the end of the XML document
7    return;
8    }
9
10   if(elementName isEqualToString:@"authenticated"]) {
11   // Check if the value of the authenticated tag is "ja"
12   _authenticationSuccess = [currentElementValue isEqualToString:@"ja"];
13   }
14
15           .
16           .
17           .
18 }
```

Listing 6.2: XML Parser überprüft die Antwort des SOAP Web Service

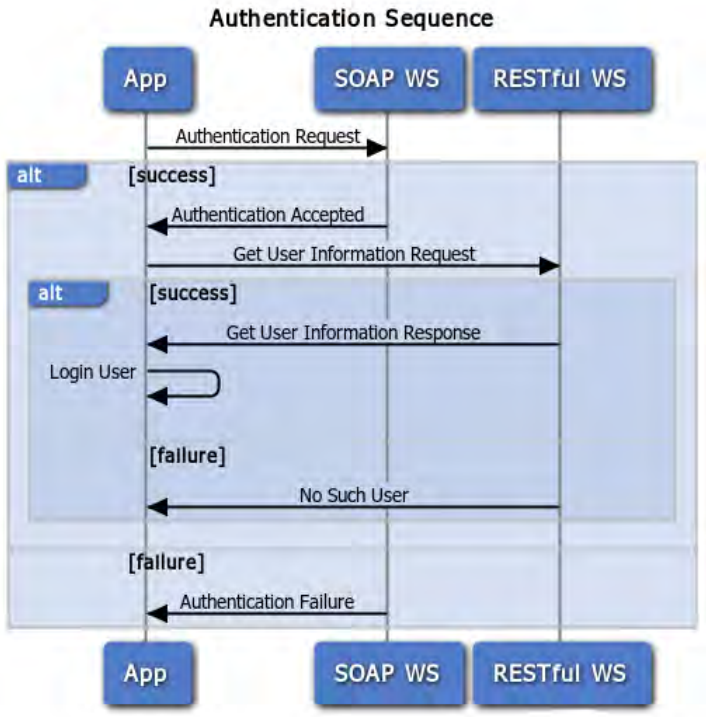


Abbildung 6.13: UML-Sequenzdiagramm zur Authentifizierung eines Benutzers[4]

[3]Erstellt mit WebSequenceDiagrams (http://www.websequencediagrams.com/).

6.8.3 Studentensicht

Hauptsicht

Nach erfolgreicher Anmeldung in der Rolle als Student wird der Hauptsichtbereich des *PASplitView-Controller* von einem *UINavigationsController* verwaltet (siehe Abbildung 6.11 (4)). Dieser verwaltet verschiedene *Table View Controller* zur Ansicht der Datenebenen (Testat→Übungen→Beschriftungen) in Tabellenform. Jede Tabellensicht besitzt dabei, die von dem *UINavigationsController* bereitgestellte Navigationsleiste. Mit ihr hat der Benutzer die Möglichkeit, sich zur nächsten Datenebene zurück zu navigieren, in dem er auf den mit dem Namen der Datenebene beschrifteten Button links in der Leiste drückt. In der Mitte der Leiste befindet sich die Bezeichnung der aktuell angezeigten Datenebene. Besonders hervorzuheben ist hier der *Table View Controller TestTVC*, welcher mehrere Funktionen besitzt. In der Studentensicht wird zwischen den drei Zuständen *Test* (während Eingaben erfolgen), *PreResult* (Zwischenauswertung) und *Evaluate* (Endauswertung) unterschieden. Bei jedem Zustandsübergang wird die Tabellensicht neu geladen (siehe Zustandsautomat in Abbildung 6.14). Das Verhalten des *TestTVCs* ist je Zustand folgendermaßen:

Test Sobald der Benutzer Pins beschriftet, werden diese Beschriftungen als Datenquelle für die anzuzeigende Tabellensicht verwendet. Das heißt, dass für jeden bereits beschrifteten Pin die Beschriftung in der dem Pin zugeordneten Tabellenzelle angezeigt wird. Bei noch unbeschrifteten Pins, wird der Platzhalter "unbeschrifteter Pin" angezeigt. Durch Selektieren eines Pins in der Detailsicht wird auch die dazugehörige Tabellenzelle selektiert. Bei jeder Benutzereingabe wird die Tabellensicht in diesem Zustand neu geladen (siehe Abbildung 6.15).

PreResult Sobald der Benutzer seine Eingaben beendet hat, kann er sich ein Zwischenergebnis anzeigen lassen, indem er auf den mit "Zwischenergebnis" beschrifteten Button in der Detailsicht drückt und so einen Zustandsübergang zu "PreResult" auslöst. Dabei ändert sich die Farbe der korrekten Beschriftung zu grün, falsche Beschriftungen werden dagegen rot angezeigt (siehe Abbildung 6.16). Nach der Zwischenauswertung der Pins können diese weiterhin beschriftet werden. Darüber hinaus ist jederzeit eine weitere Zwischenauswertung möglich. Wenn der Benutzer eine neue Eingabe tätigt, wird die Tabellensicht im "Test"-Zustand neu geladen.

Evaluate Will der Benutzer die Übung endgültig auswerten lassen, betätigt er den mit "Auswerten" beschrifteten Button in der Detailsicht. Es erscheint eine leicht transparente modale Sicht, in deren Zentrum sich eine Textsicht befindet. In dieser kann der Benutzer ablesen, wie viele Pins, im Verhältnis zu der Gesamtanzahl der Pins, korrekt beschriftet wurden (siehe Abbildung 6.18). Durch Berührung des Bildschirms wird die modale Sicht geschlossen und es wird die Übung mit den korrekten Beschriftungen angezeigt. Beschriftungen, welche der Benutzer falsch oder gar nicht angegeben hat, werden dabei in rot, richtige Beschriftungen dagegen in grün, dargestellt (siehe Abbildung 6.17). Sobald sich der Benutzer im "Evaluate"-Zustand befindet, kann der Zustand nicht mehr gewechselt werden, bis eine neue Übung geladen wird.

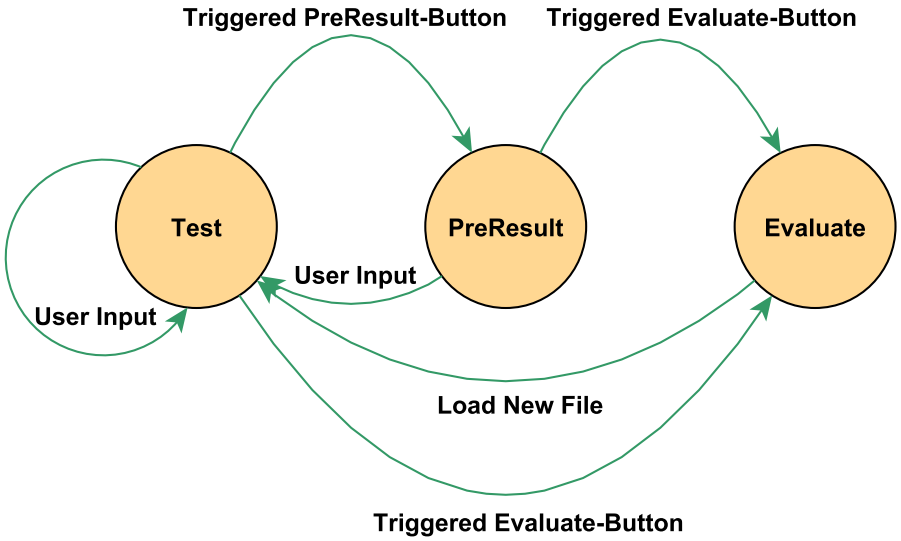


Abbildung 6.14: Zustandsautomat des *TestTVC*.

Detailsicht

Die Detailsicht des Split View Controllers wird in der Studentensicht durch den *TestViewController* verwaltet (siehe Abbildung 6.11 (5)).
Am oberen Rand der Sicht befindet sich eine Leiste. In der Leiste befinden sich die zwei Buttons, welche von der Hauptsicht zur Anzeige eines Zwischenergebnisses beziehungsweise zur Auswertung benötigt werden. Außerdem befindet sich noch ein weiterer Button in der Leiste, mit welchem sich der Benutzer abmelden kann, um wieder zu der Sicht des "LoginViewController" zu gelangen.
Der restliche Bereich besteht aus drei übereinander gelegten Sichten.
Bei der obersten Sicht handelt es sich um die sogenannte *PrepAppView*. Diese wird von der Klasse *UIView* des UIKit Frameworks abgeleitet. Hier wird die Methode "*drawRect:*" überschrieben, um die Pfeile, welche durch die Klasse *PathDrawInfo* realisiert werden, zu zeichnen. Eine weitere Funktion der Sicht ist, dass hier auch bestimmte Bereiche hervorgehoben werden, welche in *HighlightedLayer* gespeichert sind. Darüber hinaus werden die Pins in Form von "*Subviews*" (etwa "Untersichten") direkt auf ihr platziert.
Die darunterliegende Sicht ist die *PrepAppImageView*, welche von der *UIImageView* des UIKit-Frameworks erbt. Sie dient lediglich zur Darstellung der mit der Übung verknüpften Bilddatei.
Die unterste Sicht bildet eine *UIScrollView*, welche ebenfalls durch das UIKit-Framework bereitgestellt wird. Durch sie wird dem Benutzer eine Zoomfunktion in den Übungen ermöglicht. Dafür wird die übliche Zoomgeste von Apple benutzt. Darüber hinaus hat der Anwender die Möglichkeit, bestimmte Bildbereiche mit zweimaligen Antippen hereinzuzoomen und, indem er die Sicht mit zwei Fingern zweimalig antippt, wieder herauszuzoomen.
Beim Laden einer Übung werden die Pins, welche als Objekte der Klasse *DraggableViews* ebenfalls von *UIView* erben, als Untersichten der *PrepAppView* hinzugefügt. Um eine direkte Beschriftung, wie in Kapitel 4, Abschnitt 4.2.2.1 beschrieben wird, zu ermöglichen, bekommen die Pins einen *UITapGestureRecognizer* zugewiesen. Sobald der Benutzer durch Berühren eines Pins eine solche Geste auslöst, wird ein *Popover* über dem Pin angezeigt. Dieses *Popover* enthält eine Textfeld, welches

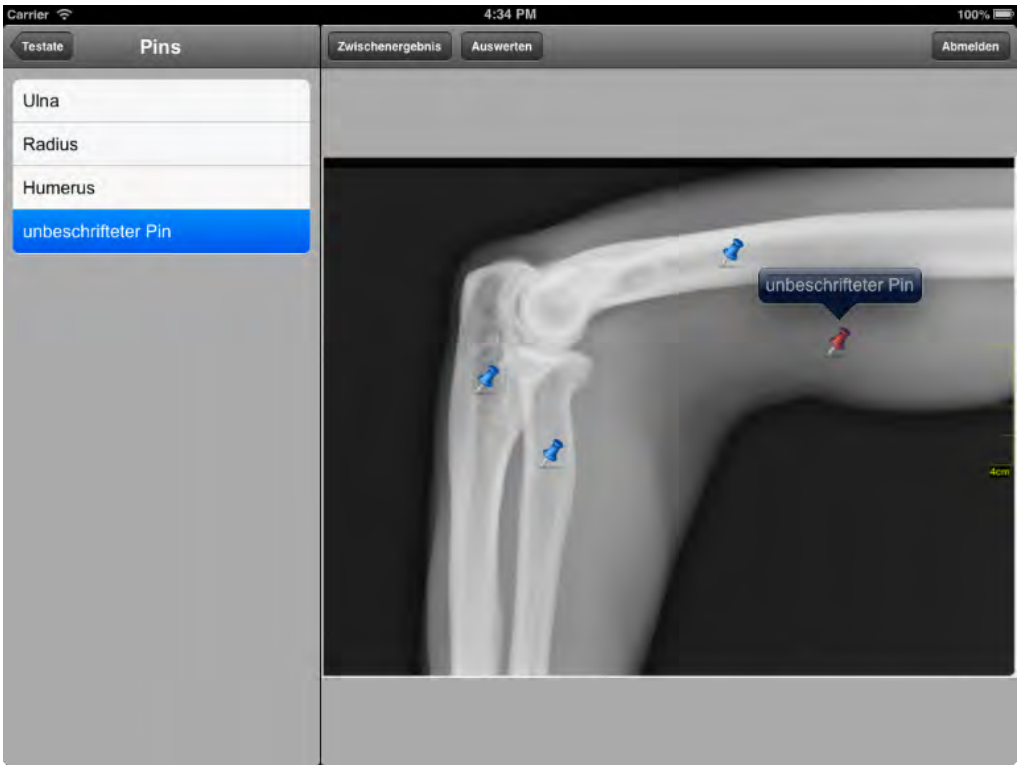


Abbildung 6.15: *TestTVC* im Zustand *Test*

Abbildung 6.16: *TestTVC* im Zustand *PreResult*

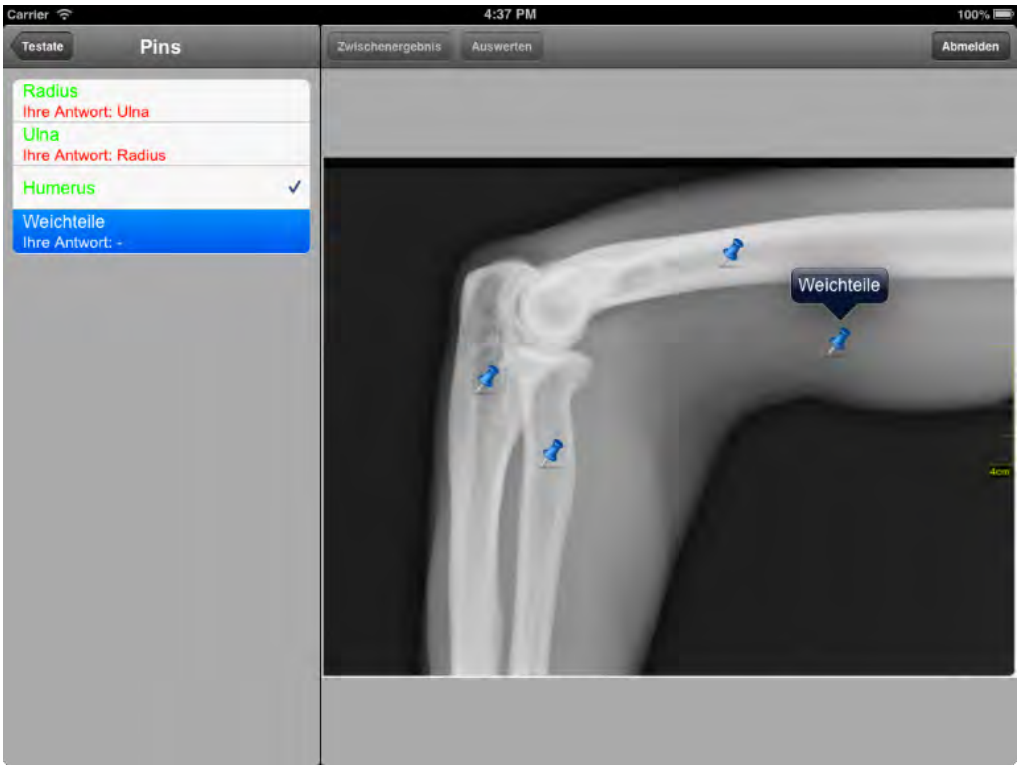


Abbildung 6.17: *TestTVC* im Zustand *Evaluate*

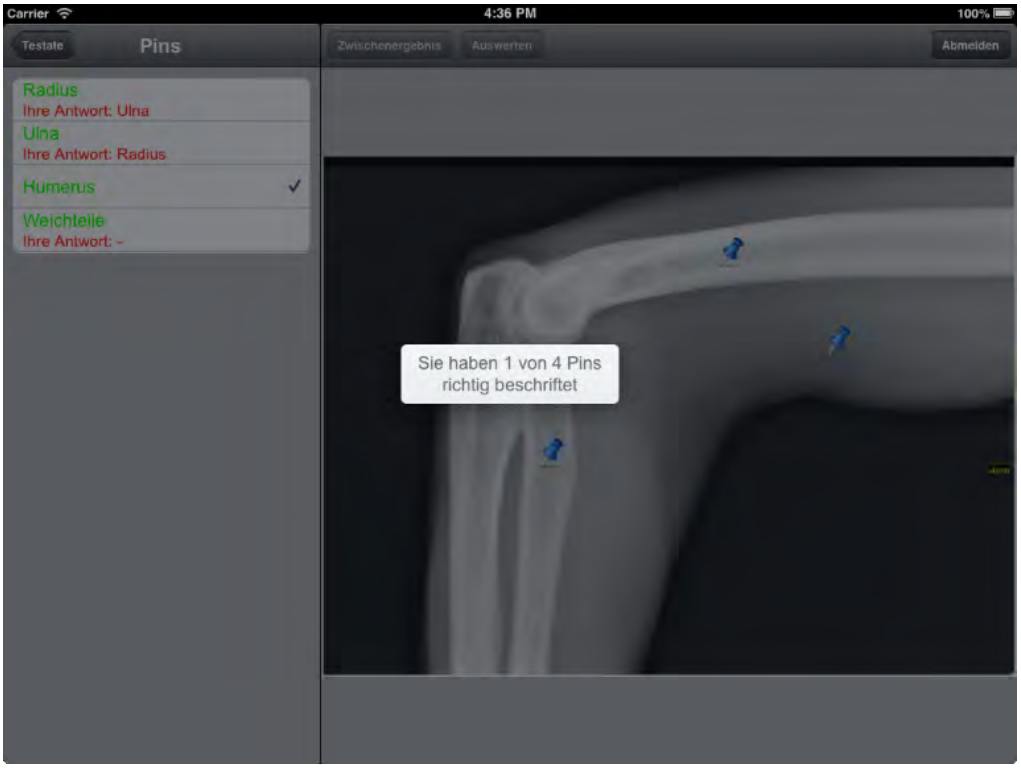


Abbildung 6.18: Das Übungsergebnis in einem modalen Fenster

zuerst den Platzhalter "Pin beschriften" anzeigt und so dem Benutzer suggerieren soll, dass er das Textfeld zur Beschriftung des Pins berühren muss. Nach Berührung des Textfeldes verschwindet der Platzhalter und es wird eine virtuelle Tastatur angezeigt. Nachdem der Benutzer seine Antwort eingetippt hat und mit der Eingabetaste beziehungsweise "Tastatur-verbergen"-Taste bestätigt, wird dem Pin diese Beschriftung zugewiesen und zukünftig auch in der Tabelle und bei weiteren Berührungen des Pins angezeigt (siehe Abbildung 6.15). Wurde die Übung ausgewertet, werden nur noch die richtigen Begriffe beim Berühren eines Pins angezeigt (siehe Abbildung 6.17).

6.8.4 Administatorsicht

Hauptsicht

Wie auch bei der Studentensicht, werden nach Anmeldung des Benutzers in der Rolle des Administrators, die Hauptsichten von einem *UINavigationController* verwaltet (siehe Abbildung 6.11 (3)). Mit ihm kann sich der Anwender durch die verschiedenen Tabellensichten der *UITableViewController* navigieren, welche von der Klasse *EditableTVC* abgeleitet werden.
Im Unterschied zu der Studentensicht, wird es dem Benutzer in jeder Tabellensicht ermöglicht, ein eigenes Objekt anzulegen. Um neue Objekte anzulegen, gibt es in der Navigationsleiste ein "Plus"-Button. Nach Drücken des "Plus"-Buttons wird die oberste Zelle mit einem Platzhalter versehen (beispielsweise "Testat eingeben" in der Datenebene der *PrepfileSections*) und es erscheint die virtuelle Tastatur, um die gewünschte Bezeichnung einzugeben.
Indem eine Wischgeste auf einer der Zellen ausgeführt wird, kann sich der Anwender einen "Löschen"-Button anzeigen lassen. Durch Betätigen des Buttons werden Zelleneinträge als gelöscht markiert, so dass sie bei der nächsten Synchronisierung gelöscht werden. Dieses Verhalten wird bereits durch die Klasse *UITableViewController* angeboten.
Darüber hinaus ist es möglich, bestehende Einträge umzubenennen. Um diese Funktion umzusetzen, wird zuerst ein *UILongPressGestureRecognizer* für den *EditableTVC* registriert. Dieser erzeugt ein Event, sobald der Benutzer die Tabellensicht länger als eine Sekunde lang gedrückt hält. Anhand der berührten Stelle, wird der sogenannte *indexPath*[5] der berührten Zelle ermittelt. Zudem werden anstatt der standardmäßig für die Zellen einer Tabellensicht benutzten Objekte der Klasse *UITableViewCell* eigene Objekte der Klasse *EditableCell* verwendet. Diese werden von der Klasse *UITableViewCell* abgeleitet und besitzen als zusätzliche Funktion, dass bei ihnen *PlaceholderTextViews* zur Anzeige der Zelleninhalte verwendet werden, anstatt der normalerweise verwendeten *UILabels*.
Die Klasse *PlaceholderTextView* ist ebenfalls eine eigene Klasse, welche von *UITextView* abgeleitet wird und zusätzlich zu dieser Platzhalter erlaubt. Der Vorteil dabei von *UITextView* abzuleiten besteht darin, dass Objekte dieser Klasse gegenüber *UILabels* das Delegieren von Methoden an den Controller ermöglichen.
Um die Größe einer Zelle korrekt anzupassen, während der Anwender diese beschriftet, werden Methoden der *PlaceholderTextView* an das Objekt der Klasse *EditableTVC* delegiert. In Listing 6.3 wird die Methode, die zur Berechnung der Höhe einer Zelle dient, dargestellt. Diese wird immer aufgerufen, sobald die vom Objekt *EditableTVC* verwaltete, *UITableView* neu gezeichnet wird.
Wie in Abschnitt 6.7 beschrieben wird, werden als gelöscht markierte Objekte rot, veränderte beziehungsweise neu erstellte Objekte grün, und bereits synchronisierte weiß hinterlegt. Dies wird in Abbildung 5.1a gezeigt.

[5] Ein *indexPath* dient zur Identifizierung einer Zelle. Er enthält die Sektion und die Reihe der Zelle.

```
1  - (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NS
       IndexPath *)indexPath
2  {
3      // this method is called for each cell and returns its height
4      if (_tVCContent != Pins) {
5
6          // update height while user is typing into a textview
7          if (_typing && indexPath.section == self.currentSelection.section &&
               indexPath.row == _currentSelection.row) {
8              CGFloat rowHeight = self.textView.contentSize.height + 6;
9              return (DEFAULTHEIGHT>rowHeight ? DEFAULTHEIGHT:rowHeight);
10         }
11
12         // else return the default height (44) or the calculated height if the
               text doesn't fit
13         else {
14             CGFloat rowHeight = 0.0f;
15             NSString *text = [self getTextFromManagedObjectAtIndexPath:
                   indexPath];
16
17             // calculate the size of the text
18             CGSize size = [text sizeWithFont:[UIFont fontWithName:@"Arial" size
                   :18.0f] constrainedToSize:CGSizeMake(tableView.frame.size.width
                    - 38, CGFLOAT_MAX) lineBreakMode:NSLineBreakByWordWrapping];
19
20             // height of the text + height of a line with used font for padding
21             rowHeight = size.height + [[UIFont fontWithName:@"Arial" size:18.0f
                   ] lineHeight];
22
23             return (DEFAULTHEIGHT>rowHeight ? DEFAULTHEIGHT:rowHeight);
24         }
25
26     }
27     else {
28         return [super tableView:tableView heightForRowAtIndexPath:indexPath];;
29     }
30 }
```

Listing 6.3: Dynamische Berechnung der Höhe einer Tabellenzelle

Detailsicht

In der Administratorsicht wird die Detailsicht durch den *AdminDetailViewController* verwaltet (siehe Abbildung 6.11 (6)).
Wie auch bei der Studentensicht, befindet sich am oberen Rand der Sicht eine Leiste mit Buttons und der restliche Bereich besteht aus den übereinanderliegenden Sichten *PrepAppView*, *PrepAppImageView* und *UIScrollView*.
Neben einem Button zum Abmelden vom System, existieren vier weitere Buttons, deren Funktionen in den nachfolgenden Abschnitten genauer erklärt werden.

6.8.4.1 Bild auswählen/ Bild aufnehmen

Wurde eine Übung angelegt, muss der Benutzer zuerst ein Bild als Hintergrund festlegen, um auf weitere Funktionen zugreifen zu können.
Hier hat der Benutzer entweder die Möglichkeit, ein Bild aus den auf dem iPad gespeicherten Bildern auszuwählen, oder ein eigenes Bild mithilfe der Kamera aufzunehmen. Um diese Funktion umzusetzen, wurde der *UIImagePickerController* des *UIKit* Frameworks verwendet. Die unterstützten Dateiformate für Bilddateien ergeben sich daraus, dass die Klasse *UIImagePickerController UIImages*, welche von *UIImageViews* benutzt werden, verwendet. Es werden dementsprechend alle Dateiformate, welche durch die Klasse *UIImage* repräsentiert werden können, unterstützt. Dabei handelt es sich um die in Tabelle 6.2 dargestellten Formate. Die Speicherung der Bilddateien erfolgt allerdings ausschließlich im *Portable Network Graphic* Format.

Format	Dateiendungen
Tagged Image File Format (TIFF)	.tiff, .tif
Joint Photographic Experts Group (JPEG)	.jpg, .jpeg
Graphic Interchange Format (GIF)	.gif
Portable Network Graphic (PNG)	.png
Windows Bitmap Format (DIB)	.bmp, .BMPf
Windows Icon Format	.ico
Windows Cursor	.cur
XWindow bitmap	.xbm

Tabelle 6.2: Unterstütze Dateiformate für Bilddateien [17]

Durch Drücken des "Bild auswählen"- beziehungsweise "Bild aufnehmen"-Buttons wird der *UIImagePickerController* in einem *Popover* instantiiert. Je nach Wahl wird im *UIImagePickerController* eine Datenquelle und die anzuzeigende Bildformate beziehungsweise, ob ein Bild oder Video aufgenommen werden soll, gesetzt.
Werden Bilder der Foto Bibliothek des iPads als Datenquelle gewählt, wird im *Popover* eine Tabellensicht zur Übersicht über die vorhandenen Fotosammlungen gezeigt. Diese Fotosammlungen können beispielsweise mithilfe der in iTunes [43] angebotenen Synchronisierung auf das iPad übertragen werden. Nach Auswahl eines Albums werden die dazugehörigen Bilder als verkleinerten Vorschau dargestellt. Falls in dem *Popover* nicht genug Platz für die Darstellung sämtlicher Vorschaubilder vorhanden ist, kann mit Wischgesten gescrollt werden.

Wurde die Kamera als Datenquelle ausgewählt, bietet der *Popover* die Sicht der Kamera als Vorschau und ein Button, um Bilder aufzunehmen. Wurde ein Bild aufgenommen, hat der Benutzer die Wahl, dieses zu verwerfen und erneut zur Kamerasicht zu gelangen, oder es für die Übung zu verwenden. Außerdem kann mit der bekannten Zoom-Geste herein- und herausgezoomt werden. Darüber hinaus wird nach Auswahl einer Bilddatei die Dateigröße überprüft. Dateien, welche größer als 2MB sind, werden komprimiert, indem ihre Auflösung angepasst wird, was dazu dienen soll, langen Synchronisierungszeiten vorzubeugen. Dazu wird die Auflösung der Bilddatei auf 640x480 Bildpunkte skaliert.

6.8.4.2 Werkzeuge

Um bestimmte Aspekte des Hintergrundbildes zu betonen, werden zwei Werkzeuge angeboten. Drückt der Benutzer einen der Werkzeug-Buttons, wird das Bild des Buttons durch ein neues ersetzt, bei dem die schwarzen und weißen Bildbereiche invertiert wurden. Auf diese Weise soll dem Benutzer vermitteln werden, dass das Werkzeug gerade angewählt ist (siehe Abbildung 6.19).
In beiden Werkzeugen werden Bezierkurven eingesetzt. Bezierkurven werden zum einfachen Zeichnen von Linien verwendet und werden durch die Klasse *UIBezierPath* des *UIKit*-Frameworks realisiert.
Das Prinzip der Werkzeug-Objekte ist dabei, dass sobald eines der Werkzeuge ausgewählt wurde, sämtliche Touch-Events an diese weitergeleitet werden, welche auf der *PrepAppView* registriert werden. Dazu werden Methoden verwendet, welche durch die *PrepAppView* Klasse von der *UIResponder* Klasse geerbt werden. Diese Methoden sind *touchesBegan:withEvent:*, *touchesMoved:withEvent:*, *touchesEnded:withEvent:* und *touchesCancelled:withEvent:*. Mit ihnen lassen sich unter anderem Start- und Endpunkt einer Berührung bestimmen, welche als Parameter für die Berechnungen der Werkzeuge dienen.

(a) **(b)**

Abbildung 6.19: Grafiken für die angewählten Werkzeuge

Pfeile

Mit dem Pfeilwerkzeug lassen sich, mithilfe von Wischgesten, Pfeile auf die *PrepAppView* zeichnen. Dazu wird anhand des Start- und Endpunktes der Berührung ein Pfad durch den Einsatz von Bezierkurven erstellt. An den Endpunkt der Berührung wir außerdem ein Dreieck gezeichnet, dass als Pfeilspitze dient. Damit das Dreieck die gleiche Neigung wie die gezeichnete Linie hat, wird eine Transformation mithilfe der *CGAffineTransform* Klasse vorgenommen.
Es wird außerdem ein weiterer *UIBezierPath* in Form eines Rechteckes um den bestehenden gezeichnet. Dieser legt den Bereich fest, in dem der Pfeil bei Berührung angewählt werden kann (siehe dazu Abbildung 6.20). Durch längeres Gedrückthalten eines Pfeiles, hat man die Möglichkeit, diesen zu löschen.
In Listing 6.4 wird die Methode gezeigt, welche nach Beendigung der Wischgeste ausgeführt wird.

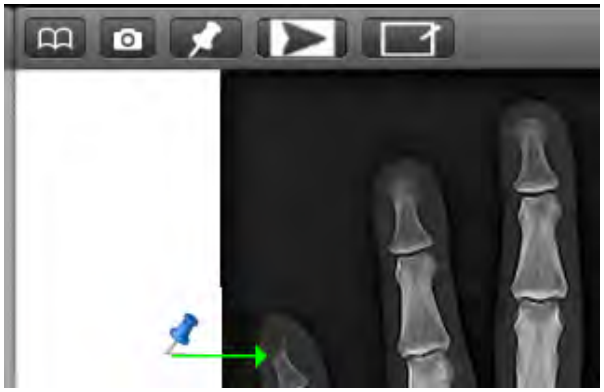


Abbildung 6.20: Beispielhafte Benutzung des Pfeil-Werkzeugs

Listing 6.4: Zeichnen eines Pfeils, sobald die Wischgeste abgeschlossen wurde

```
// create BezierPath from startPoint to endPoint and let the delegate draw it
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    UIView *touchedView = [_delegate viewForUse];
        // make a line from the start point to the current point
        // check if start point exists
        if (_trackingTouch) {
            CGPoint endPoint = [[touches anyObject] locationInView:touchedView
                ];

            CGFloat xDist = (endPoint.x - _startPoint.x);
            CGFloat yDist = (endPoint.y - _startPoint.y);
            CGFloat distance = sqrt((xDist * xDist) + (yDist * yDist));

            // only draw at a minimum length of 12 pixels
            if (distance >= 12.0f) {
                UIBezierPath *path = [UIBezierPath bezierPath];
                [path setLineWidth:2.0f];

                // arrow body
                [path moveToPoint:_startPoint];
                [path addLineToPoint:endPoint];
                [path closePath];

                // arrow head
                // make a path for the arrow head
                UIBezierPath *arrowHeadPath = [UIBezierPath bezierPath];
                [arrowHeadPath moveToPoint:CGPointMake(endPoint.x - 6, endPoint
                    .y - 6)];
                [arrowHeadPath addLineToPoint:endPoint];
                [arrowHeadPath addLineToPoint:CGPointMake(endPoint.x - 6,
                    endPoint.y + 6)];

                // get the right angle for the arrow head
                CGFloat angle = atan2f(yDist , xDist);

                // transform arrowhead to the right angle
                CGAffineTransform translateTransform =
                    CGAffineTransformMakeTranslation(endPoint.x, endPoint.y);
```

```
36                  CGAffineTransform rotationTransform =
                        CGAffineTransformMakeRotation(angle);
37                  CGAffineTransform customRotation = CGAffineTransformConcat(
                        CGAffineTransformConcat(CGAffineTransformInvert(
                        translateTransform), rotationTransform), translateTransform
                        );
38
39                  [arrowHeadPath applyTransform:customRotation];
40                  [arrowHeadPath closePath];
41
42                  // append arrow head to the body
43                  [path appendPath:arrowHeadPath];
44
45                  // make an rect to define a touch area for hit testing
46                  CGRect touchArea = CGRectMake(_startPoint.x, _startPoint.y -
                        10, distance + 8, 20.0f);
47                  UIBezierPath *touchPath = [UIBezierPath bezierPathWithRect:
                        touchArea];
48
49                  // transform touch area to the right angle
50                  translateTransform = CGAffineTransformMakeTranslation(
                        _startPoint.x, _startPoint.y);
51                  rotationTransform = CGAffineTransformMakeRotation(angle);
52                  customRotation = CGAffineTransformConcat(
                        CGAffineTransformConcat(CGAffineTransformInvert(
                        translateTransform), rotationTransform), translateTransform
                        );
53
54                  [touchPath applyTransform:customRotation];
55                  [touchPath closePath];
56
57
58                  PathDrawInfo *info = [PathDrawInfo pathDrawingInfoWithPath:path
                         andTouchablePath:touchPath];
59                  [_delegate addDrawable:info];
60              }
61              _trackingTouch = nil;
62          }
63      }
64  }
```

Listing 6.4: Zeichnen eines Pfeils, sobald die Wischgeste abgeschlossen wurde

Bereiche hervorheben

Mit Werkzeug zur Hervorhebung von Bildbereichen können bestimmte Bildbereich mit einem leicht transparenten braunen Farbton markiert werden. Zur Zeichnung des temporären Rahmen, welcher den ausgewählten Bereich während der Berührung darstellt, wurde die Klasse *UIBezierPath* verwendet. Die endgültige Färbung des Bildbereiches geschieht jedoch durch das Einfärben eines *CALayer* der *PrepAppView*. *CALayer* sind Klassen, die simple Rechtecke auf dem Bildschirm repräsentieren, welche visuelle Inhalte darstellen. So ein *CALayer* wird auch von einer *UIView* benutzt, um darauf zu zeichnen. Aus diesem Grund hat es sich angeboten, direkt auf diese zuzugreifen und sie zu verändert, anstatt neue *UIView*s als Untersichten der *PrepAppView* hinzuzufügen, und diese dann entsprechend zu bearbeiten.

6.8.4.3 Pins

Da es sich bei dem Setzen eines Pins um einen komplizierten Vorgang handelt, bei dem mehrere Klassen beteiligt sind, werden in Tabelle 6.3 zuerst die Klassen und ihre Eigenschaften vorgestellt und danach wie diese miteinander interagieren.

Klasse	Basisklasse	Beschreibung
DragAndDrop-TableViewController	UITableViewController	Die Klasse verfügt über zwei *Delegates*. Bei diesen handelt es sich um ein Objekt von dem ein bewegliches Objekt (Grafik, Tabellenzelle, usw.) aufgenommen werden kann und ein weiteres, auf dem das Objekt abgelegt wird. Dabei werden Events, welche von Gesten des Anwenders ausgelöst werden, an die *Delegates* weitergeleitet.
Draggables-MenuController	DragAndDrop-TableViewController	Der Controller verwaltet die Sicht, welche der Anwender angezeigt bekommt, sobald er den Pin-Button betätigt.
DraggableView	UIView	Objekte dieser Klasse stellen die Pins dar. Um dies zu realisieren, verfügen sie über die Begriffslisten und Benutzereingaben, die Position und die Grafik des darzustellenden Pins.

Tabelle 6.3: Die Klassen, welche beim Setzen eines Pins beteiligt sind.

Durch das Berühren des Pin-Buttons wird ein *Popover* geöffnet, dessen Inhalt von der Klasse *DraggablesMenuController* verwaltet wird. Der *DraggablesMenuController* erstellt eine *UITableView* mit zwei Zellen, in denen die beiden Pin-Arten dargestellt werden. Zudem weist er jeder Zelle einen *UIPanGestureRecognizer* zu, damit Pan-Gesten (in etwa Zieh-Gesten) des Anwenders registriert werden können. Zusätzlich zu dieser Funktion, werden durch diese Klasse auch Aufgaben, anhand der Events, welche von dem *UIPanGestureRecognizer* erzeugt werden, delegiert. Dies geschieht, da von der Oberklasse *DragAndDropTableViewController* geerbt wird (siehe Tabelle 6.3). Als *Delegates* dienen dabei sie selbst und die Klasse *AdminDetailViewController*. Welche Methoden des *Delegates* aufgerufen werden, entscheidet sich anhand des Zustands der Geste. Es wird dabei zwischen drei verschiedenen Zuständen unterschieden:

- *UIGestureRecognizerStateBegan* Der Zeitpunkt an dem der Benutzer die Geste ausgelöst hat.
- *UIGestureRecognizerStateChanged* Die Zeitpunkte während der Benutzer die Geste ausführt
- *UIGestureRecognizerStateEnded* Der Zeitpunkt an dem der Benutzer die Geste beendet hat.

In den ersten beiden Zuständen werden eigene Methoden des *DraggablesMenuController* aufgerufen. Sobald der Anwender mit einer Pan-Geste beginnt (*UIGestureRecognizerStateBegan*), wird in der angewählten Zelle der *UITableView* das Bild des Pins entfernt und eine neue *UIImageView* mit dem Bild erstellt (siehe folgendes Listing).

```
-(void)dragAndDropTableViewController:(DragAndDropTableViewController *)ddtvc
    draggingGestureWillBegin:(UIGestureRecognizer *)gesture forCell:(UI
    TableViewCell *)cell{

    // copy image from the cell
    UIImage *img = [[UIImage alloc] init];
    img = cell.imageView.image;

    // create an UIImageView with the image at the location where the gesture
        began
    self.dragAndDropView = [[UIImageView alloc] initWithImage:img];
    [self.dragAndDropView setBackgroundColor:[UIColor clearColor]];
    [self.dragAndDropView setCenter:[gesture locationInView:self.prepAppView]];

    // add UIImageView to the PrepAppView
    [self.prepAppView addSubview:self.dragAndDropView];

    // close Popover
    [self.container dismissPopoverAnimated:YES];
}
```

Diese Sicht wird der *PrepAppView* als Untersicht hinzugefügt und beim Ausführen der Pan-Geste (*UIGestureRecognizerStateChanged*) mit dem Finger des Benutzers mit bewegt, dabei aber nicht außerhalb der *PrepAppView* gezogen werden kann (siehe folgendes Listing).

```
-(void)dragAndDropTableViewController:(DragAndDropTableViewController *)ddtvc
    draggingGestureDidMove:(UIPanGestureRecognizer *)gesture
{
  // get the current location of the gesture in the PrepAppView
    CGPoint translation = [gesture translationInView:self.prepAppView];

    // get current center point of the UIImageView
    CGPoint currentCenter = self.dragAndDropView.center;

    \\ check if UIImageView is inside the bounds of the PrepAppView
    CGFloat maxX = self.prepAppView.bounds.size.width;
    if (currentCenter.x + translation.x < 0 )
    {
        translation.x =  (0 - currentCenter.x);
    }
    if (currentCenter.x + translation.x >= maxX )
    {
        translation.x = (maxX - translation.x);
    }

    CGFloat maxY = self.prepAppView.bounds.size.height;
    if (currentCenter.y  + translation.y < 0 )
    {
        translation.y = (0 - currentCenter.y);
    }
    if (currentCenter.y + translation.y >= maxY )
    {
        translation.y = (maxY - translation.y);
    }

    // move the UIImageView
    currentCenter.x += translation.x;
```

```
32      currentCenter.y += translation.y;
33      self.dragAndDropView.center = currentCenter;
34
35      // changing the translation value resets the velocity of the pan
36      [gesture setTranslation:CGPointZero inView:self.prepAppView];
37  }
```

Endet die Pan-Geste (*UIGestureRecognizerStateEnded*), indem der Benutzer seinen Finger hebt, wird sowohl in dem *DraggablesMenuController* als auch in dem *AdminDetailViewController* eine Methode aufgerufen.
In der Methode des *DraggablesMenuController* wird die *UIImageView* entfernt. Der Aufruf in dem *AdminDetailViewController* erzeugt dagegen an dieser Stelle eine *DraggableView* (siehe Tabelle 6.3) als Untersicht der *PrepAppView*. Zudem wird dem *DraggableView*-Objekt wiederum ein *UIPanGestureRecognizer* zugewiesen. Die entsprechenden Daten zur Instantiierung des *DraggableView*-Objektes werden mithilfe von *Core Data* in der Datenbank gespeichert.
Um auch Pins hinter dem *Popover* zu platzieren, wird dieser entfernt, sobald innerhalb des *Popovers* eine Pan-Geste registriert wird (siehe folgendes Listing).

```
1   -(void)dragAndDropTableViewController:(DragAndDropTableViewController *)ddtvc
        droppedGesture:(UIGestureRecognizer *)gesture{
2
3       CGPoint position = [gesture locationInView:self.prepAppView];
4       UIView *viewHit = [self.prepAppView hitTest:position withEvent:nil];
5       NSLog(@"view in which we dropped the pin: %@", [viewHit description]);
6
7       if([ddtvc isKindOfClass:[DraggablesMenuController class]]){
8           DraggablesMenuController *dmc = (DraggablesMenuController *) ddtvc;
9           PinType selectedPin = dmc.selectedPin;
10
11          if([self.prepAppView  isEqual:viewHit]){
12              DraggableView *dView = [DraggableView createViewWithType:
                    selectedPin position:position label:nil note:nil];
13              [self addDraggable:dView];
14              [self draggableWasSelected:dView.id];
15              [self handleDismissedPopoverController:dmc.container];
16              self.currentDraggable = dView;
17          }
18      }
19  }
```

Des Weiteren öffnet sich an der linken Bildschirmseite eine Tabelle, in der die gültigen Bezeichnungen für den Pin eingegeben werden können. Da zumindest eine Bezeichnung existieren muss, wird die virtuelle Tastatur präsentiert und alle sonstigen Eingabemöglichkeiten blockiert. Erst nachdem der Benutzer zumindest eine beliebige Ziffer zur Beschriftung des Pins eingegeben hat, kann er durch die Eingabe- oder "Tastatur verbergen"-Taste neue Benutzereingaben mithilfe der Benutzeroberfläche tätigen. Der Zeitpunkt, an dem ein Pin abgelegt wurde, ist in Abbildung 6.21 erfasst.

6.9 Zusammenfassung

In diesem Kapitel erfolgte ein Überblick über die Implementierung des Prototypen. Dazu wurden zuerst die grundlegenden Kenntnisse vermittelt, die für die Entwicklung von Software für ein iOS-Gerät notwendig sind. Danach wurden die wichtigsten Unterschiede bei der Entwicklung auf einem

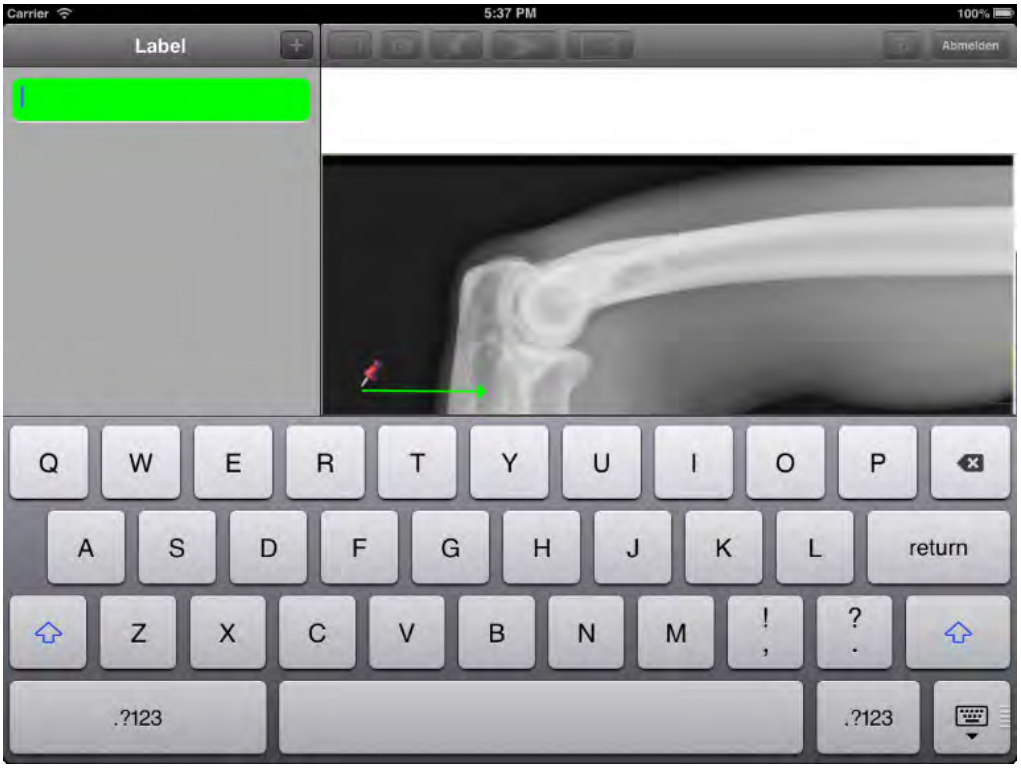


Abbildung 6.21: Setzen eines Pins

iPad gegenüber anderen iOS-Geräten aufgezeigt. Darüber hinaus wurde die Softwarearchitektur des Prototypen vorgestellt und die in der Entwicklung eingesetzten Frameworks und Bibliotheken vorgestellt. Abschließend werden die einzelnen Komponenten des Systems genauer betrachtet und im Detail vorgestellt.
Die bereits in dem iOS-SDK enthaltenen Klassen haben die Entwicklung des Prototyps in einigen Bereichen unterstützt. Allerdings war es auch des Öfteren nötig, diese anzupassen, um die doch eher speziellen Funktionen, wie die Pins oder direkt manipulierbare Tabellensichten, zu realisieren. Zudem erleichterte die Hinzunahme der *AFNetworking* Bibliothek die Kommunikation mit den Web Services, was für einen reibungslosen Ablauf bei der Synchronisation der Daten essentiell war.

7 Diskussion

In diesem Kapitel erfolgt eine Diskussion darüber, ob die in Kapitel 3 an das System gestellten Anforderungen erreicht wurden. Bei Anforderungen, die nicht umgesetzt werden konnten, wird beschrieben aus welchem Grund dies der Fall war. Zur Übersicht wurden die verschiedenen Anforderungspunkte in der folgenden Tabelle 7.1 zusammengefasst.

Tabelle 7.1: Umsetzung der Anforderungen

Allgemeine Anforderungen		
Anforderung	**Beschreibung**	**Implementiert**
Mobilität	Für die Synchronisierung der Daten wird eine Internetverbindung benötigt, was die Mobilität einschränken könnte, da diese nicht überall verfügbar ist. Bereits synchronisierte Testate und Übungen können jederzeit mit dem iPad bearbeitet werden. Zudem ist es als Administrator auch jederzeit möglich, Testate und Übungen zu erstellen, zu bearbeiten oder zu löschen.	ja
Sämtliche Orientierungen des Gerätes werden unterstützt	Es wurde nur die Orientierung im Querformat umgesetzt.	nein
Übungen (Bildmaterial)	Mithilfe von Kamerafotos oder Bilddateien, die in den in Tabelle 6.2 genannten Formaten vorliegen, können Übungen für die Testate erstellt werden. Die Übungen haben eine ähnliche, aber einheitlichere optische Repräsentation wie im bisherigen System.	ja

Fortgesetzt auf der nächsten Seite

Tabelle 7.1 – Fortgesetzt von der vorherigen Seite

Anforderung	Beschreibung	Implementiert
Übungen (3D-Modelle)	Die Architektur der App müsste grundlegend verändert werden, um frei rotierbare 3D-Modelle darstellen, beziehungsweise erstellen zu können. Da dieses Thema an sich den Umfang einer eigener Arbeit hätte, wurde es hier nicht implementiert.	nein
Login	Durch Angabe von Benutzerdaten wird die Rolle des Anwenders identifiziert und entsprechende Funktionen bereitgestellt und Inhalte angezeigt.	ja
Synchronisierung der Daten	Bei jedem Programmstart werden die Daten synchronisiert. Zudem können Administratoren jederzeit die von ihnen erstellten Daten synchronisieren lassen und damit den Studenten bereitstellen.	ja
Administrator Anforderungen		
Anlegen einer Dateistruktur	Für jede Hierarchieebene der Dateistruktur wurde eine Tabellensicht implementiert. In den Tabellen können Einträge hinzugefügt, gelöscht und bearbeitet werden. Durch die Navigation zwischen den verschiedenen Tabellensichten, wird ein Drill-Down auf die nächste Hierarchieebene realisiert (beispielsweise wird nach Auswahl eines Testats die Tabellensicht der dazugehörigen Übungen geöffnet).	ja
Bildmaterial einfügen	Bildmaterial kann durch die Kamera des iPads aufgenommen werden und direkt aus der Fotobibliothek des iPads geladen werden. Die Auswahl von Bildern aus einer Webressource wurde dagegen nicht implementiert.	teilweise
Bildmaterial bearbeiten	Es wurden Werkzeuge implementiert, mit denen es möglich ist, Pfeile in das Bild zu zeichnen und Bildbereiche hervorzuheben, indem sie farblich hinterlegt werden.	ja
Beschriftungen einfügen	Die anatomischen Strukturen, welche auf den Bildern abgebildet sind, werden mit Pins markiert. Jedem Pin wird dabei eine Beschriftung zugeordnet.	ja
Student Anforderungen		
Auswahl von Übungen	Dem Studenten wird die gleiche Drill-Down Navigation zwischen den Tabellensichten geboten wie einem Administrator. So können gezielt Übungen ausgewählt werden.	ja

Fortgesetzt auf der nächsten Seite

Tabelle 7.1 – Fortgesetzt von der vorherigen Seite

Anforderung	Beschreibung	Implementiert
Zwischenergebnis anzeigen	Durch Drücken eines Buttons kann man sich das aktuelle Zwischenergebnis anzeigen. Danach kann die Übung weiter bearbeitet werden.	ja
Übung auswerten	Will der Student ein endgültige Auswertung der Übung, drückt er auf den dafür vorhergesehenen Button. Darauf bekommt er angezeigt, welche anatomischen Strukturen richtig und welche falsch benannt wurden. Bei den falsch benannten Strukturen wird zudem die korrekte Antwort angezeigt.	ja
	Weitere Anforderungen	
Mehrsprachigkeit	Mitteilungen an den Benutzer und die Beschriftung der Buttons sind nur in deutscher Sprache verfügbar.	nein
Begriffslisten	Zu jedem Pin können beliebig viele gültige Bezeichnungen eingetragen werden. Die Bezeichnung, welche als erstes eingetragen wurde, wird dem Studenten bei falschen Antworten als Korrektur angezeigt.	ja

Wie der Tabelle zu entnehmen ist, konnten drei Anforderungen nicht implementiert und eine nur teilweise umgesetzt werden.
Ursprünglich war die Anwendung der App im Hoch- und Querformat vorgesehen. Da aber die Bilddateien der Übungen der Größe der Sicht, welche diese darstellt, angepasst werden, um verschiedene Bildgrößen zuzulassen, veränderten sich die Bildgrößen je Orientierung erheblich. Da die Pins anhand eines festen Koordinatensystems der Sicht geladen werden und nicht an Bildpixeln verankert werden, führte dies dazu, dass die Positionen der Pins nicht mehr mit den zu beschriftenden Bildbereichen übereinstimmten. Zudem wurde dadurch das Hinzufügen neuer Beschriftungen in der Administratorsicht erschwert, da die Tabellensicht, um neue Beschriftungen einzufügen, nur mit einem *Popover* eingeblendet werden konnte, wie dies bei einer *UISplitView* üblich ist (siehe dazu Abschnitt 6.2.2).
Die nächste Anforderung, welche nicht umgesetzt wurde, ist die bisherigen Testate und deren Übungen als frei rotierbare 3D-Modelle darzustellen. Da der Umfang, um diese Funktion umzusetzen, für diese Arbeit zu groß gewesen wäre, wurde auf die Implementierung von 3D-Modellen verzichtet. Zudem wäre es dann sinnvoll, eine Möglichkeit anzubieten, selbst 3D-Modelle in der Administratorsicht entwerfen zu können, was die Komplexität der App um ein Vielfaches erhöht hätte.
Die Implementierung einer Auswahl von verschiedenen Sprachen ist in der App ebenfalls noch nicht realisiert worden, da diese Anforderung eine geringe Priorität hatte und zeitlich nicht mehr umgesetzt werden konnte.
Die ursprünglichen Möglichkeiten, um Bildmaterial in das System einzupflegen, wurden, während der Entwicklung um die Möglichkeit, Bilder auch mithilfe der Kamera des iPads aufzunehmen, erweitert. Dafür wurde aber die ursprüngliche Anforderung, Bildmaterial auch über Webressourcen

einpflegen zu können, verworfen. Diese Anforderung hätte die Mobilität der App eingeschränkt, da man wieder auf eine Internetverbindung angewiesen wäre und so nicht jederzeit Übungen erstellen könnte, da kein Bildmaterial vorhanden ist.

8 Zusammenfassung

Die Aufgabe der Diplomarbeit bestand darin, ein bestehendes System zur Vorbereitung auf den Präparierkurs des Instituts für Anatomie und Zellbiologie zu verbessern und daraus eine mobile Anwendung für das Apple iPad zu entwickeln. Dazu wurde das bestehende System analysiert und mehrere Gespräche mit den beteiligten Lehrkräften geführt.
Anhand dieser Anforderungen wurden Architekturkonzepte und mögliche Benutzeroberflächen für die Anwendung entworfen. Aus diesen wurden schließlich die für am besten erachtende Entwürfe ausgewählt und in einer Implementierungsphase umgesetzt. Im folgenden Abschnitt erfolgt ein Ausblick auf die möglichen Weiterentwicklungen und Verbesserungen, welche man ergänzend in die App implementieren könnte.

8.1 Ausblick

Während der Entwicklung der App stellten sich mehrere Möglichkeiten zur Verbesserung beziehungsweise Erweiterung verschiedener Aspekte des Systems heraus. Mögliche Verbesserungen und Erweiterungen, welche zukünftig bei dem System vorgenommen werden könnten, sind:

- *Mehrsprachigkeit*
  In der Loginsicht könnte ein Button implementiert werden, mit dem man eine neue Sicht beziehungsweise eine Sicht innerhalb eines *Popovers* anzeigen könnte, in der man die Möglichkeit hat, verschiedene Sprachen auszuwählen. Die Auswahl einer Sprache wirkt sich dabei auf Dialog und die Beschriftungen der Buttons aus. Zudem müssen zu den Pins Begriffslisten in der entsprechenden Sprache neu eingepflegt werden.
- *3D-Modelle*
  Die Testate und Übungen des bisherigen Systems könnten als frei rotierbare 3D-Modelle in das App eingepflegt werden. So könnten Körperregionen besser dargestellt werden, da eine Betrachtung von allen Seiten möglich wäre und deshalb keine anatomischen Strukturen verdeckt

werden könnten. Wird eine Übung durch einen Studenten bearbeitet, könnte dieser, durch Klicken auf den entsprechenden Eintrag in der linken Tabellensicht, das 3D-Modell automatisch auf den zu beschriftenden Teil des Modells rotieren und zentrieren. Die Funktionalität, um eigene 3D-Modelle anzulegen, würde die Benutzerfreundlichkeit für den durchschnittlichen Anwender stark reduzieren und daher wird davon abgeraten, dies zu implementieren. Stattdessen könnten neue 3D-Modelle durch den Entwickler direkt in die App eingepflegt und durch Updates hinzugefügt werden.

- *Bildmaterial von einer Webressource laden*
  Bildmaterial könnte mithilfe der App auf einen externen Server geladen werden und wäre so für jeden Administrator verfügbar, um neue Übungen zu erstellen. Ein weiterer Vorteil dieser Erweiterung wäre, dass, das mit der Kamera des iPads aufgenommene Bildmaterial, mit anderen Administratoren geteilt werden könnte.
- *Anwendung im Hochformat*
  Das Bearbeiten der Übungen in der Studentensicht wäre durchaus auch im Hochformat denkbar. Da Pins direkt bei Berührung eines *Popovers* beschriftet werden können, ist die Anzeige der Tabellensicht auf der linken Seite nicht dauerhaft notwendig. Der Anwender kann sich bei Bedarf die Tabellensicht durch einen Button, der ebenfalls ein *Popover* öffnet, einblenden lassen. So sieht er, welche Pins noch unbeschriftet sind und kann diese, indem er den dazugehörigen Eintrag in der Tabelle anwählt, auswählen. In der Administratorsicht wäre das Hochformat eher störend, da man hier für jeden Pin die Tabellensicht an der linken Bildschirmseite benötigt, um mehrere gültige Bezeichnungen für einen Pin festzulegen.
- *Push-Synchronisation*
  Bisher werden die Daten nur bei Programmstart beziehungsweise manuell durch Drücken eines Buttons synchronisiert. Man spricht in diesem Fall von einer clienseitigen Synchronisation der Daten. Als Alternative könnte ein sogenannter *Push*-Mechanismus implementiert werden, der die Synchronisation der Daten serverseitig startet, sobald eine Veränderung an den Daten stattfindet. Dazu müssten sämtliche Geräte während der Laufzeit der App über eine konstante Verbindung zum Web Service verfügen.
- *Delta-Synchronisation*
  Die Datensätze im System verfügen bereits über Zeitstempel, welche zum jetzigen Zeitpunkt noch nicht eingesetzt werden. Mit ihnen könnte man eine sogenannte Delta-Synchronisation realisieren. Bei einer Delta-Synchronisation werden nur noch die geänderten Daten eines Datensatzes übertragen, anstatt wie bisher ein "Schnappschuss" der kompletten Datenbank. Dies wurde die Synchronisation beschleunigen, indem es die zu übertragende Datenmenge zwischen Server und Client verringert.
- *Verbesserung der Benutzeroberfläche*
  Da zum Zeitpunkt dieser Arbeit die App noch nicht zur Lehre eingesetzt wurde, gab es noch keine Rückmeldungen zu seiner Benutzerfreundlichkeit. Durch Feedback von den Studenten könnte die Benutzeroberfläche weiter verbessert und angepasst werden.
- *Realisierung eines Hilfedialog*
  Durch einen Button könnte ein Hilfedialog aufgerufen werden, der dem Benutzer, je nach Rolle, die Funktionen der App übersichtlich darstellt und erklärt.

8.2 Fazit

In dieser Arbeit wurde für den Präparierkurs des Instituts für Anatomie und Zellbiologie ein umfangreicher Prototyp entwickelt, der zwei verschiedene Ansichten mit jeweils verschiedenen Funktionen bietet. Mit ihnen kann der Anwender entweder Übungen ausführen (Studentensicht) oder neue Übungen beziehungsweise Testate anlegen und bearbeiten (Administratorsicht).
Indem Administratoren ermöglicht wird, selbst beliebige Inhalte zu erstellen, können die Lehrinhalte jederzeit erweitert werden. Dabei ist das App nicht nur auf die Nutzung im medizinischen Kontext beschränkt, sondern kann zur Vermittlung beliebiger Lehrinhalte verwendet werden.
Um leicht neue Inhalte in das System einzupflegen, wurde zudem die Unterstützung der iPad-Kamera implementiert. Somit kann flexibel Bildmaterial aufgenommen und eingepflegt werden, um neue Übungen zu erstellen.
Die Benutzerfreundlichkeit für die Studenten wurde entgegen dem alten System erheblich verbessert und es wäre denkbar, dass es in Zukunft möglich wäre, prüfungsrelevante Inhalte mit die App abzuprüfen.
Darüber hinaus kann sich ein Student viel flexibler auf eine Veranstaltung vorbereiten, da ihm die App ermöglicht, dies mobil auf seinem iPad zu tun. Dadurch kann er auch sämtliche Vorteile eines solchen mobilen Geräts nutzen, wie beispielsweise einen leichten Transport durch sein geringes Gewicht oder seine intuitive Benutzersteuerung.

Abbildungsverzeichnis

1.1 Übersicht über die Arbeit . . . 3

2.1 Die Moodle-Lernplattform der Universität Ulm [37] . . . 6
2.2 Die Ansicht nach Öffnen der Präpmappe [27] . . . 6
2.3 Übung mit Pfeilen und Linien [27] . . . 7
2.4 Asterisk zur Markierung ergänzender Informationen [27] . . . 8
2.5 Beschriftung einer Übung mit Ziffern . . . 8
2.6 Gestrichelte Linien deuten auf nicht direkt sichtbare Strukturen . . . 9
2.7 Verbindung zwischen unbeschrifteten und beschrifteten Übungen . . . 9
2.8 Problem der Abgrenzung zwischen M- und E-Learning [36] . . . 11
2.9 Die Benutzeroberfläche der Skeleton System Pro III App [1] . . . 12
2.10 "360 View" der Science360 App [25] . . . 13
2.11 Übersicht von Snapguide [33] . . . 13

3.1 Bisherige Auswahl von Übungen in der "Präpmappe" . . . 19
3.2 Übung ohne (a) und mit (b) Beschriftung . . . 21

4.1 Horizontale vs. vertikale Prototypen. [26] . . . 26
4.2 Die Informationsarchitektur der App . . . 27
4.3 In Betracht gezogene Anmeldungsbildschirme . . . 28
4.4 Das Basisdesign der Anwendung . . . 29
4.5 In Betracht gezogene Konzepte zur Übersicht über die Datenhierarchie . . . 30
4.6 In Betracht gezogene Pin-Grafiken . . . 31
4.7 Ansicht bei Auswahl eines Pins in der Studentensicht . . . 31
4.8 Bearbeitung von Einträgen in der Hauptsicht . . . 32
4.9 Vorschaubilder bei der Auswahl eines Bildes . . . 33
4.10 Beispielhafte Anwendung der Werkzeuge . . . 34

5.1 Die Übersicht (links), nachdem eine Übung hinzugefügt wurde . . . 37
5.2 Auswahl eines Testats. . . . 40
5.3 Auswahl einer Übung. . . . 41

6.1 Die iOS Schichten . . . 47
6.2 Code mit und ohne ARC [16] . . . 49
6.3 MVC Architektur [4] . . . 50
6.4 Target-Action-Mechanismus [3] . . . 51
6.5 iPad Popover . . . 52
6.6 *UISplitViewController* im Quer- und Hochformat [19] . . . 53
6.7 Darstellung von modalen Sichten auf dem iPad [20] . . . 55

6.8 Die Softwarearchitektur der App 57
6.9 Aufbau eines block-Codes [2] 58
6.10 Das Datenmodell dargestellt als Objektgraph von Core Data 63
6.11 Das Storyboard der App 65
6.12 Die Sicht des *LoginViewController* 66
6.13 UML-Sequenzdiagramm zur Authentifizierung eines Benutzers 68
6.14 Zustandsautomat des *TestTVC*. 69
6.15 *TestTVC* im Zustand *Test* 70
6.16 *TestTVC* im Zustand *PreResult* 70
6.17 *TestTVC* im Zustand *Evaluate* 71
6.18 Das Übungsergebnis in einem modalen Fenster 71
6.19 Grafiken für die angewählten Werkzeuge 75
6.20 Beispielhafte Benutzung des Pfeil-Werkzeugs 76
6.21 Setzen eines Pins 81

Tabellenverzeichnis

2.1 Erfüllte M-Learning Kriterien der Apps 14

3.1 Schwachstellen des bisherigen Systems 23

5.1 Mögliche weitere Anwenderszenarien 43

6.1 Methoden des RESTful Web Service 61
6.2 Unterstütze Dateiformate für Bilddateien [17] 74
6.3 Die Klassen, welche beim Setzen eines Pins beteiligt sind. 78

7.1 Umsetzung der Anforderungen 83

Literaturverzeichnis

[1] 3D4MEDICAL: *Skeletal System Pro III*. http://applications.3d4medical.com/skeletal_pro. (Zuletzt besucht am 23.März 2013).

[2] APPLE: *Blocks Programming Topics: Introduction*. http://developer.apple.com/library/ios/#documentation/cocoa/Conceptual/Blocks/Articles/00_Introduction.html. (Zuletzt besucht am 23.März 2013).

[3] APPLE: *Cocoa Application Competencies for iOS: Target-Action*. https://developer.apple.com/library/mac/#documentation/General/Conceptual/Devpedia-CocoaApp/TargetAction.html. (Zuletzt besucht am 23.März 2013).

[4] APPLE: *Cocoa Core Competencies: Model-View-Controller*. https://developer.apple.com/library/mac/#documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html. (Zuletzt besucht am 23.März 2013).

[5] APPLE: *Core Data Release Notes for OS X v10.7 and iOS 5.0:*. http://developer.apple.com/library/mac/releasenotes/DataManagement/RN-CoreData/index.html. (Zuletzt besucht am 23.März 2013).

[6] APPLE: *iCloud*. http://www.apple.com/de/icloud/. (Zuletzt besucht am 23.März 2013).

[7] APPLE: *iOS App Programming Guide: About iOS App Programming*. http://developer.apple.com/library/ios/#documentation/iphone/conceptual/iphoneosprogrammingguide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40007072-CH1-SW1. (Zuletzt besucht am 23.März 2013).

[8] APPLE: *iOS Human Interface Guidelines: Platform Characteristics*. http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/Characteristics/Characteristics.html.

[9] APPLE: *iOS Technology Overview: Cocoa Touch Layer*. http://developer.apple.com/library/ios/#documentation/miscellaneous/conceptual/iphoneostechoverview/iPhoneOSTechnologies/iPhoneOSTechnologies.html. (Zuletzt besucht am 23.März 2013).

[10] APPLE: *iOS Technology Overview: Core OS Layer*. http://developer.apple.com/library/ios/#documentation/miscellaneous/conceptual/iphoneostechoverview/coreoslayer/coreoslayer.html. (Zuletzt besucht am 23.März 2013).

[11] APPLE: *iOS Technology Overview: Core Services Layer*. http://developer.apple.com/library/ios/#documentation/miscellaneous/conceptual/iphoneostechoverview/CoreServicesLayer/CoreServicesLayer.html. (Zuletzt besucht am 23.März 2013).

[12] APPLE: *iOS Technology Overview: Media Layer*. http://developer.apple.com/library/ios/#documentation/miscellaneous/conceptual/iphoneostechoverview/MediaLayer/MediaLayer.html. (Zuletzt besucht am 23.März 2013).

[13] Apple: *iPad - technical specifications.* http://www.apple.com/ipad/specs/.

[14] Apple: *Key-Value Observing Programming Guide: Introduction to Key-Value Observing Programming Guide.* http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/KeyValueObserving/KeyValueObserving.html. (Zuletzt besucht am 23.März 2013).

[15] Apple: *Notification Programming Topics: Notifications.* http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/Notifications/Articles/Notifications.html#//apple_ref/doc/uid/20000215-BCICIHGE. (Zuletzt besucht am 23.März 2013).

[16] Apple: *Transitioning to ARC Release Notes.* (Zuletzt besucht am 23.März 2013).

[17] Apple: *UIImage Class Reference.* http://developer.apple.com/library/ios/#documentation/uikit/reference/UIImage_Class/Reference/Reference.html. (Zuletzt besucht am 23.März 2013).

[18] Apple: *View Controller Catalog for iOS: About View Controllers.* http://developer.apple.com/library/ios/#documentation/WindowsViews/Conceptual/ViewControllerCatalog/Introduction.html. (Zuletzt besucht am 23.März 2013).

[19] Apple: *View Controller Catalog for iOS: Split View Controllers.* http://developer.apple.com/library/ios/#documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/SplitViewControllers.html. (Zuletzt besucht am 23.März 2013).

[20] Apple: *View Controller Programming Guide for iOS: Presenting View Controllers from Other View Controllers.* http://developer.apple.com/library/ios/#featuredarticles/ViewControllerPGforiPhoneOS/ModalViewControllers/ModalViewControllers.html. (Zuletzt besucht am 23.März 2013).

[21] Back, Andrea, Oliver Bendel und Daniel Stoller-Schai: *E-Learning im Unternehmen: Grundlagen-Strategien-Methoden-Technologien.* 2001.

[22] DAINTITH, J.: *A Dictionary of Computing.* http://www.encyclopedia.com/doc/1O11-softwareprototyping.html, 2004. (Zuletzt besucht am 23.März 2013).

[23] EllisLab: *CodeIgniter.* http://ellislab.com/codeigniter. (Zuletzt besucht am 23.März 2013).

[24] Georgiev, Georgieva, Smrikarov: *Proceedings of the 5th international conference on Computer Systems and Technologies.* 2004. (Stand März 2013).

[25] iTunes: *Science360 for iPad.* https://itunes.apple.com/app/science360-for-ipad/id439928181?mt=8. (Zuletzt besucht am 23.März 2013).

[26] Nielsen, J.: *Usability Engineering.* Morgan Kaufmann, September 1994.

[27] Präpmappe: *Veranstaltung: Makroskopische Anatomie (Präparierkurs).* https://www.lernplattform.medizin.uni-ulm.de/moodle/course/view.php?id=18&topic=0#section-2. (Zuletzt besucht am 23.März 2013).

[28] Pryss, Rüdiger, David Langer, Manfred Reichert und Alena Hallerbach: *Mobile Task Management for Medical Ward Rounds - The MEDo Approach.* In: *1st Int'l Workshop on Adaptive Case Management (ACM'12), BPM'12 Workshops*, LNBIP. Springer, September 2012.

[29] Pryss, Rüdiger, Julian Tiedeken, Ulrich Kreher und Manfred Reichert: *Towards Flexible Process Support on Mobile Devices.* In: *Proc. CAiSE'10 Forum - Information Systems Evolution*, Nummer 72 in *LNBIP*, Seiten 150–165. Springer, 2010.

[30] Reichert, Manfred und Barbara Weber: *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies.* Springer, Berlin-Heidelberg, 2012.

[31] Schulmeister, Rolf: *eLearning.* Oldenbourg Verlag, 2006.

[32] SLIM: *Slim Framework.* `http://www.slimframework.com/`. (Zuletzt besucht am 23.März 2013).

[33] Snapguide: *Snapguide.* `http://snapguide.com/`. (Zuletzt besucht am 23.März 2013).

[34] Techopedia: *What is Snapshot Replication?* `http://www.techopedia.com/definition/24441/snapshot-replication`. (Zuletzt besucht am 23.März 2013).

[35] The PHP Group: *PHP: Datenbankerweiterungen - Manual.* `http://www.php.net/manual/de/refs.database.php`. (Zuletzt besucht am 23.März 2013).

[36] Traxler, J.: *Defining mobile learning.* In: *Proceedings, IADIS international conference on mobile learning, Malta*, 2005.

[37] Universität Ulm: *Lernplattform Medizin Universität Ulm.* `https://www.lernplattform.medizin.uni-ulm.de/moodle/`.

[38] Wikipedia: *First-class function.* `http://en.wikipedia.org/wiki/First-class_function`. (Zuletzt besucht am 23.März 2013).

[39] Wikipedia: *Liste von Softwareentwicklungsprozessen.* `http://de.wikipedia.org/w/index.php?title=Liste_von_Softwareentwicklungsprozessen&oldid=110378243`, November 2012. (Zuletzt besucht am 23.März 2013).

[40] Wikipedia: *Common Object Request Broker Architecture.* `http://en.wikipedia.org/w/index.php?title=Common_Object_Request_Broker_Architecture&oldid=545665737`, März 2013. (Zuletzt besucht am 23.März 2013).

[41] Wikipedia: *Hypertext Transfer Protocol.* `http://en.wikipedia.org/w/index.php?title=Hypertext_Transfer_Protocol&oldid=546859884`, März 2013. (Zuletzt besucht am 23.März 2013).

[42] Wikipedia: *Interface Builder.* `http://en.wikipedia.org/w/index.php?title=Interface_Builder&oldid=540531086`, März 2013. (Zuletzt besucht am 23.März 2013).

[43] Wikipedia: *iTunes.* `http://de.wikipedia.org/w/index.php?title=ITunes&oldid=115461875`, März 2013. (Zuletzt besucht am 23.März 2013).

[44] Wikipedia: *Xcode.* `http://en.wikipedia.org/w/index.php?title=Xcode&oldid=543266627`, März 2013. (Zuletzt besucht am 23.März 2013).

[45] ZEND: *Zend Framework.* `http://framework.zend.com/`. (Zuletzt besucht am 23.März 2013).