|  |
| --- |
|  |
| HTWG Grade App |
| Noten aus dem Studententerminal auslesen und dem Benutzer unterwegs zur Verfügung stellen. |

|  |
| --- |
| MSI - Master Informatik (Embedded Systems)  Wintersemester 2012/13 |
| Manuel Caputo (282208) und Andreas Bug (282207)  21.01.2013 |

# Kurzbeschreibung

Die Thematik der in diesem Dokument beschrieben App ist das auslesen und anzeigen von Abschlüssen und Noten aus dem Notenterminal der HTWG Konstanz.

* Mobil auf Noten zugreifen
* Login mit RZ Benutzername
* Anzeige der Studentendaten, Abschlüssen und Noten
* Tablet und smartphone view
* Suchen, filtern, details anzeigen und teilen von noten

# Einleitung

Im Kurs *Mobile Computing* soll eine App (Applikation) für die Android-Plattform von Google entwickelt werden. Einzige Vorgaben sind, dass in Zweierteams eine App geschrieben werden soll, welche einen mobilen Anteil besitzt und in Android geschrieben ist.

Da wir uns mit einer möglichst realistischen Anwendung beschäftigen wollten, welche zudem einen nützlichen Zweck erfüllt, haben wir uns für eine Noten-App entschieden. Dabei sollen die im Studententerminal vorhandenen Abschlüsse (z.B Bachelor- und Masterabschluss) sowie alle dazugehörenden erbrachten Leistungen ausgelesen und dargestellt werden.

Ziel ist es, dem Benutzer einen bequemen Weg zu bieten, seine Noten und Abschlüsse von einem mobilen Gerät abzurufen. Dabei liegen die Schwierigkeiten des Projekts zum einen in der Verbindung zum Server und dem Auslesen der Daten, zum anderen in der möglichst einfachen Darstellung der Daten.

Da das Studententerminal (QIS-Server) keinen Web-Service oder Datenbankzugang für den Datenzugriff anbietet, musste ein alternativer Weg für den Zugriff gefunden werden. Das Notenterminal verfügt über ein Web-UI, welches über HTTP angesprochen werden kann. Darüber ist es möglich eine Verbindung zu simulieren und die Daten auszulesen. Um an die Abschlüsse und Noten eines Studenten zu gelangen, muss eine HTTPS-Verbindung erstellt werden, welche anschließend genutzt wird um an die gewünschten Informationen zu gelangen. Dabei müssen die HTML-Seiten automatisiert geladen und ausgewertet werden.

Nachdem die extrahierten Daten des Studenten zur Weiterverarbeitung gefiltert wurden, sollen sie in der App dargestellt werden. Zur besseren Handhabung sollen dafür unterschiedliche Ansichten (Layouts) erstellt werden, welche in Kapitel vorgestellt werden.

Da alle Daten direkt vom QIS-Server bezogen werden, muss während die Informationen geladen werden eine Internetverbindung bestehen. Zusätzlich muss der Benutzer natürlich ein Student der HTWG-Konstanz sein und über ein Konto des Rechenzentrums verfügen.

# Erwartete Ergebnisse

Äääähm eine App die unsere noten ausließt!? Wtf?

* Abschlüsse und Noten finden, auslesen und anzeigen
* Dem Benutzer Funktionen zur Verfügung stellen wie Suche und Filter
* Doppelte Noten und falsche Angaben wegen Falscheinträgen auf dem Server
* Langsames auslesen wegen HTTPS Verbindung zum eh schon langsamen Server
* Langsames auslesen wegen komplette HTML Seite durchsuchen

# Werkzeuge und Software

Als Entwicklungsumgebung kam Eclipse zum Einsatz, da dies die von Google offiziell unterstützte IDE ist. Dazu benutzen wir das Android-SDK in der neusten Version für Android 4.2 (API Version 17) zusammen mit Eclipse Juno. Da wir aktuelle Geräte mit Android 4.2.1 besitzen, stellte dies kein Problem dar. Getestet wurde unsere App im Emulator und auf unseren Geräten (das Smartphone Nexus 4 von LG und das Tablet Nexus 7 von Asus).

Für die Verwaltung des Quellcodes und aller anderen Projektdaten wie Dokumentation haben wir die Plattform Github.com benutzt, welche für Open-Source-Projekte kostenlos GIT-Repositories bereitstellt. Als Software für GIT unter Windows kam TortoiseGIT zum Einsatz.

Für die Erstellung des App-Icons wurde das freier Vektorzeichenprogramm Inkscape benutzt. Außerdem verwendeten wir die kostenlose Software CamStudio um Videos der App im Emulator aufzuzeichnen, da es auf Android-Geräten ohne Rootrechte und kostenpflichtige Zusatzsoftware nicht möglich ist, Videos des Bildschirms aufzuzeichnen.

# Versuchsaufbau

Da unsere App keinerlei externe Sensoren wie GPS, Kompass oder ein Gyroskop benutzt und nur eine aktive Internetverbindung benötigt, entfallen in unserem Fall aufwendige Versuchsaufbauten.

Für den Test der Benutzeroberfläche wurde ein Datenadapter erstellt, der einen einfachen Testdatensatz auch ohne Internetverbindung zur Verfügung stellt. Somit war es ohne die Wartezeit für die Datenabfrage möglich, die Benutzeroberfläche zu testen.

Umgekehrt haben wir bei der Entwicklung der Klassen für den Datenzugriffs auf das QIS-Terminal den Android-Code mit der Benutzeroberfläche außen vor gelassen. Die betreffenden Klassen für den Datenzugriff wurden in Eclipse als normaler Java-Code direkt auf dem PC entwickelt und getestet, ohne dabei auf Android(-Code) zurückzugreifen. Da diese Klassen nur Funktionen der Standard-Java-Library benutzen, war der Code anschließend ohne Änderungen auch unter Android lauffähig.

# Architektur

In diesem Kapitel werden die Architektur sowie die Funktionalität der Anwendung beschrieben. Dabei wird auf alle für unsere App wichtigen Implementierungen eingegangen, ohne die allgemeine Architektur einer Android-App näher zu beschreiben.

## Use-Cases

TODO:

## System

Die Android-Architektur gibt dem Entwickler einige Strukturen vor. In der Datei 'AndroidManifest.xml' wird zum Beispiel festgelegt welche *Activity* beim Start aufgerufen werden soll, welche Eigenschaften die Activities haben und welche Rechte die Applikation auf dem Gerät benötigt. Die Angaben der Berechtigungen sind für die Anwendung wichtig, damit sie sich zum Beispiel mit dem Internet verbinden darf. Die Auflistung der erforderlichen Rechte ist auch notwendig um den Benutzer darüber zu informieren, welche internen Funktionen seines Geräts durch die Applikation benutzt werden. Bei der Installation muss der Benutzer diesen Berechtigungen zustimmen, um die Anwendung installieren zu können. Die HTWG Grade App benötigt lediglich die Zugriffsrechte auf das Internet.

Da eine Darstellung aller möglichen Abläufe der App zu umfangreich für dieses Dokument wäre, wird in nur ein überschaubarer Anteil präsentiert.

Abbildung 1: Ausschnitt des Ablaufplans der App

Zum Start der App wird zunächst Abschlüsse-Activity (DegreeListActivity) initialisiert. Wie durch den Lebenszyklus einer Activity bekannt ist, werden dabei einige Methoden durchlaufen. In der onCreate-Methode wird die die Aktualisierung der Daten angefordert. Dabei wird zunächst überprüft ob Benutzerdaten vorliegen. Sollten keine Benutzerdaten gespeichert sein, wird die Einstellungs-Activity (SettingsActivity) gestartet. Sind Benutzerdaten vorhanden, wird eine asynchrone Task gestartet, welche die Daten vom Server lädt. Wenn noch keine Daten geladen sind, wird das sichtbare Fragment durch ein Aktualisierungs-Fragment (RefreshFragment) ersetzt. Nach dem die Daten geladen wurden, wird entschieden ob es sich um die Zwei-Fragment-Ansicht für große Bildschirme, oder die Smartphone-Ansicht handelt. Für Smartphones, werden eine Liste mit den gefundenen Abschlüssen und einige Informationen zu dem RZ-Benutzer angezeigt. Handelt es sich um ein Tablet, welches als "large-screen" erkannt wurde, werden neben einer Liste mit Abschlüssen und den Studentendaten, eine Liste der Kurse des ersten gefundenen Abschlusses angezeigt.

## Implementierung und Funktionalität

Die App enthält mehrere Activity-, Fragment-, und Hilfs-Klassen. Um die Zusammenhänge der einzelnen Klasse besser verstehen zu können, sind in die wichtigsten implementierten Klassen aufgezählt und kurz beschrieben.

|  |  |  |
| --- | --- | --- |
| Klassenname | Klassentyp | Beschreibung |
| DegreeListActivity | Activity-Klasse | Dient als Einstieg in die App. Lädt Ein- oder Zwei-Fragment Ansicht, zeigt das Fragment *DegreeListFragment* als aktuellen View und startet die Aktualisierung der Studentendaten. |
| DegreeListFragment | Fragment-Klasse | Zeigt Liste aller gefundenen Abschlüsse sowie Informationen des konfigurierten RZ-Benutzers. |
| GradesListActivity | Activity-Klasse | Zur Anzeige von gefundenen Noten eines Abschlusses. Wird nur im Smartphone-Layout genutzt um das Fragment der Kursdaten zu laden und Aktionen des Fragments zu verarbeiten. |
| GradesListFragment | Fragment-Klasse | Zeigt Liste aller Kurse, mit eingeschränkten Details, eines ausgewählten Abschlusses. Die Kurse können zur Detailansicht ausgewählt werden. Wenn angezeigt, werden im Menü Optionen zur Filterung und Suche angezeigt. |
| GradeDetailsFragment | Fragment-Klasse | Zur Anzeige von Details eines ausgewählten Kurses. Das Fragment wird unabhängig des Geräts, über den gesamten Bildschirm angezeigt. Zurück auf die Liste, kommt man nur über die *Zurück*-Taste des Geräts. Dem Benutzer wird zusätzlich die Möglichkeit gegeben seine Kursdaten, über eine auf dem Gerät installierte Anwendung, mit anderen zu teilen. |
| RefreshFragment | Fragment-Klasse | Enthält nur einen Aktualisieren-Knopf. Das Fragment wird angezeigt, wenn Benutzerdaten vorliegen, aber keine Daten vom Server geladen wurden. |
| SettingsActivity | Activity-Klasse | Für die Einstellungsmöglichkeiten. Wird automatisch angezeigt, wenn keine Benutzerdaten vorliegen. Kann jeder Zeit aus dem Menü aufgerufen werden. |
| GradeListAdapter | Adapter-Klasse | Erweiterung der SimpleAdapter Klasse von Android, zur Darstellung der Kurse eines Abschlusses in einer Liste. |
| GradeDetailsAdapter | Adapter-Klasse | Erweiterung der SimpleAdapter Klasse von Android, zur Darstellung der Details eines Kurses in Form von einer Liste. |
| Content | Hilfs-Klasse | Datencontainer-Klasse zur temporären Ablage von Abschlüssen, Kursen und Studenteninformationen. Bietet statische Variablen in denen die Daten abgelegt werden, sowie eine Funktion zum Aktualisieren der Daten. |
| QisRequest | Hilfs-Klasse | Erweiterung der AsyncTask Klasse von Android, zum laden der Informationen vom Server. |

Tabelle 1: Wichtige Klassen der App-Implementierung

Besonders hervorzuheben ist die Klasse *QisRequest*. Sie erweitert die Android Klasse *AsyncTask* um einen nebenläufigen Prozess zu nutzen. Da das Laden der Daten vom Server einen länger andauernden Prozess darstellt, muss er durch eine asynchrone Task vom Haupt-UI-Thread getrennt ausgeführt werden. Anderenfalls würde die App sich selbst beenden, da der UI-Thread nicht mehr reagiert. Während die Task die Daten vom QIS-Server lädt und auswertet, wird ein Progress-Dialog angezeigt, welcher nicht geschlossen werden kann. In der asynchronen Task werden die Daten geladen, verarbeitet und abgelegt. Währenddessen werden Informationen, welche die Task zum Status des Ladevorgangs sendet, im Progress-Dialog angezeigt. Sollte es zu einem Fehler beim Laden kommen, wird dieser dem Benutzer angezeigt, so dass er genau weiß warum keine Daten angezeigt werden. Die Task könnte fehlschlagen, wenn zum Beispiel keine Internetverbindung besteht oder falsche Benutzer-Informationen angegeben wurden.

Der Ladevorgang wird in Kapitel genauer erklärt.

### Serververbindung

Da das Studententerminal bzw. die dahinterstehende QIS-Software keine öffentliche Schnittstelle zum Auslesen der Daten bietet, muss auf die HTML-Oberfläche für Webbrowser zurückgegriffen werden. Dabei werden alle nötigen Formulareingaben und Klicks, die normalerweise der Benutzer manuell durchführt, durch unsere Software simuliert. Um die Daten aus dem HTML-Code zu extrahieren sind verschiedene Lösungen denkbar. Die eleganteste Variante ist dabei ein XML-Parser, der XHTML-Dokumente in Form einer Baumstruktur als DOM-Tree zur Verfügung stellt. Aus einem solchen Baum kann man mit Hilfe von X-Path-Abfragen Daten extrahieren.

Leider gibt die QIS-Serversoftware keine validen XHTML-Dokumente zurück, weshalb ein strikter XML-Parser diese Seiten nicht verarbeiten kann. Aus diesem Grund haben wir uns für eine Datenextraktion über reguläre Ausdrücke (Regular Expressions) entschieden. Dabei machen wir uns zu Nutze, dass sich die HTML-Tags, welche die gesuchten Daten einschließen, meist eindeutig sind. Vor dem Namen des Studenten taucht im HTML-Quellcode zum Beispiel stets der String headers="basic\_1"> auf, direkt darauf folgt immer ein </td>.

Konkret werden folgende Schritte beim Auslesen der Daten durchgeführt:

1. Aufbauen einer HTTPS-Verbindung zum Server *qisserver.htwg-konstanz.de*. Dabei ist zu beachten, dass nur HTTP mit SSL funktioniert, einfaches HTTP funktioniert nicht und wäre zudem unsicher.
2. Absenden eines POST-Requests mit Benutzernamen und Passwort für den Login. Die Antwort des Servers enthält einen Session-Cookie der in allen nachfolgenden Anfragen mitgeschickt werden muss.
3. Da der Link zum Notenspiegel eine ID enthält, die sich bei jedem Login verändert, kann der Link nicht direkt aufgerufen werden. Es muss zunächst die Seite der Prüfungsverwaltung abgerufen werden, aus deren HTML-Quellcode der eigentliche Link zum Notenspiegel ausgelesen wird.
4. Im Notenspiegel werden, mit Hilfe eines weiteren regulären Ausdrucks, alle Abschlüsse mit Namen und Link zur Detailansicht ausgelesen.
5. In einer Schleife werden nun alle Detailansichten mit den Notenlisten des jeweiligen Abschlusses aufgerufen und mit Hilfe der regulären Ausdrücke nach Noten durchsucht. Zusätzlich enthält jeder Abschluss noch weitere Daten des Studenten (Namen, Matrikelnummer, Anschrift und Geburtsdatum). Diese Daten müssen nur bei einem Abschluss ausgelesen werden, da sie bei allen Abschlüssen eines Studenten gleich sind.

Da diese Schritte einige Zeit benötigen, (typischerweise 5 bis 20 Sekunden für zwei Abschlüsse) kann die Update-Routine nicht im Kontext der Benutzeroberfläche ausgeführt werden, da diese sonst während des Updates blockiert wäre. (Blockierte bzw. nichtreagierende Benutzeroberflächen werden von Android zwangsbeendet!)

Die Lösung ist das Update in eine asynchrone Task auszulagern. Diese Task läuft getrennt von der Oberfläche und darf daher durch den Netzwerkverkehrt blockiert werden. Um dies zu erreichen, benutzen wir die Klasse *QisRequest*, die von *AsyncTask* erbt somit eine asynchrone Task implementiert. Innerhalb der von uns implementierten Methode *doInBackground()* läuft der eigentliche Updatevorgang und sendet an geeigneten Stellen über die von *AsyncTask* bereitgestellte Methode *publishProgress()* den Fortschritt an einen *ProgressDialog*, welcher wiederum im Kontext der Benutzeroderfläche läuft und den Benutzer über den Fortschritt des Updates informiert.

## Layouts

Seit der Android Version 3.0 ist es möglich, Fragmente in einer App einzusetzen. Damit erschließen sich neue Möglichkeiten bei der angepassten Darstellung einer App in unterschiedlichen Ausgabegräten. Wie in dargestellt, wurde diese Technik in der Anwendung eingesetzt, um auf Tablets eine optimierte Darstellung zu haben.

Abbildung 2: Nutzung der unterschiedlichen Darstellungsmöglichkeiten

Verwendet man ein Gerät mit großem Bildschirm (large-screen), erhält man eine Ansicht mit zwei nebeneinander liegenden Fragmenten. Das linke zeigt eine Liste mit allen gefundenen Abschlüssen, das rechte die Details des ausgewählten Abschlusses. Wählt man einen Abschluss aus, wird das Fragment mit den Noten aktualisiert.

Nutzt man stattdessen ein Smartphone, wird nur eine Liste angezeigt. Wählt man hierbei einen Abschluss aus, wechselt die Activity und ein anderes Fragment wird angezeigt. Somit kann auf die unterschiedlichen Größen des Ausgabegeräts eingegangen werden.

# Erzielte Ergebnisse

In diesem Kapitel wird beschrieben welche Ergebnisse wir durch die Entwicklung erzielen konnten. Dabei wird auf einige Elemente der Benutzeroberfläche sowie die Bedienung der App näher eingegangen.

Was sind unsere Ergebnisse? Wir haben eine App geschrieben und nicht nach irgendwas geforscht oO

## Einstellungen

Beim ersten Starten der App, sind noch keine Benutzerdaten vorhanden und man wird zunächst zu den Einstellungen weitergeleitet. Eingestellt werden können RZ-Benutzername sowie das zugehörige Passwort. Durch Betätigen des "Zurück" Buttons gelangt man nach Eingabe der Benutzerdaten zur Hauptoberfläche.

## Informationen und blockierte UI

Während die Abschlüsse und Noten geladen werden, ist die Benutzeroberfläche blockiert. Dabei werden aktuelle Informationen zum Status des Ladevorgangs in einem Informationsblock wie in angezeigt. Sollte es beim Laden Probleme geben, werden hierbei alle Informationen zur Ursache angezeigt. Beispielsweise eine getrennte Internetverbindung kann einen solchen Fehler verursachen.



Abbildung 3: Informationsblock während des Ladevorgangs

## Abschlüsse-Ansicht

In dieser Ansicht werden alle geladenen Abschlüsse angezeigt. Wurde kein Abschluss gefunden, sind hier nur ein Aktualisierungs-Knopf sowie ein Menü sichtbar. Das Menü ist dabei, wie in zu sehen, recht simpel gehalten.



Abbildung 4: Menü in Abschlüsse-Ansicht  
1: Aktualisieren  
2: Weitere Optionen (In diesem Fall: Einstellungen)

Alle nicht als "large" erkannten Bildschirme zeigen nur eine Ansicht. Handelt es sich zum Beispiel um ein Tablet, welches als "large"-Screen erkannt wird, werden Abschlüsse und Noten in einer gespaltenen Ansicht dargestellt.

Wurde einer oder mehrere Abschlüsse gefunden, werden diese in einer Liste angezeigt. Zusätzlich erhält man Angaben zum eingestellten Benutzer angezeigt. Durch das Auswählen einer der angezeigten Abschlüsse, wird man zu den gefundenen Noten weitergeleitet. Diese Ansicht wird im Weiteren als "Noten-Ansicht" beschrieben.

## Noten-Ansicht

In dieser Ansicht werden alle gefundenen Noten eines Abschlusses angezeigt. Die Noten sind lexikografisch nach Prüfungstext sortiert. Bei kleinen Anzeigegeräten, werden Prüfungstext und Note angezeigt. Große Bildschirme zeigen zudem die ECTS des Kurses. In ist die Notenansicht auf einem kleinen Bildschirm dargestellt.

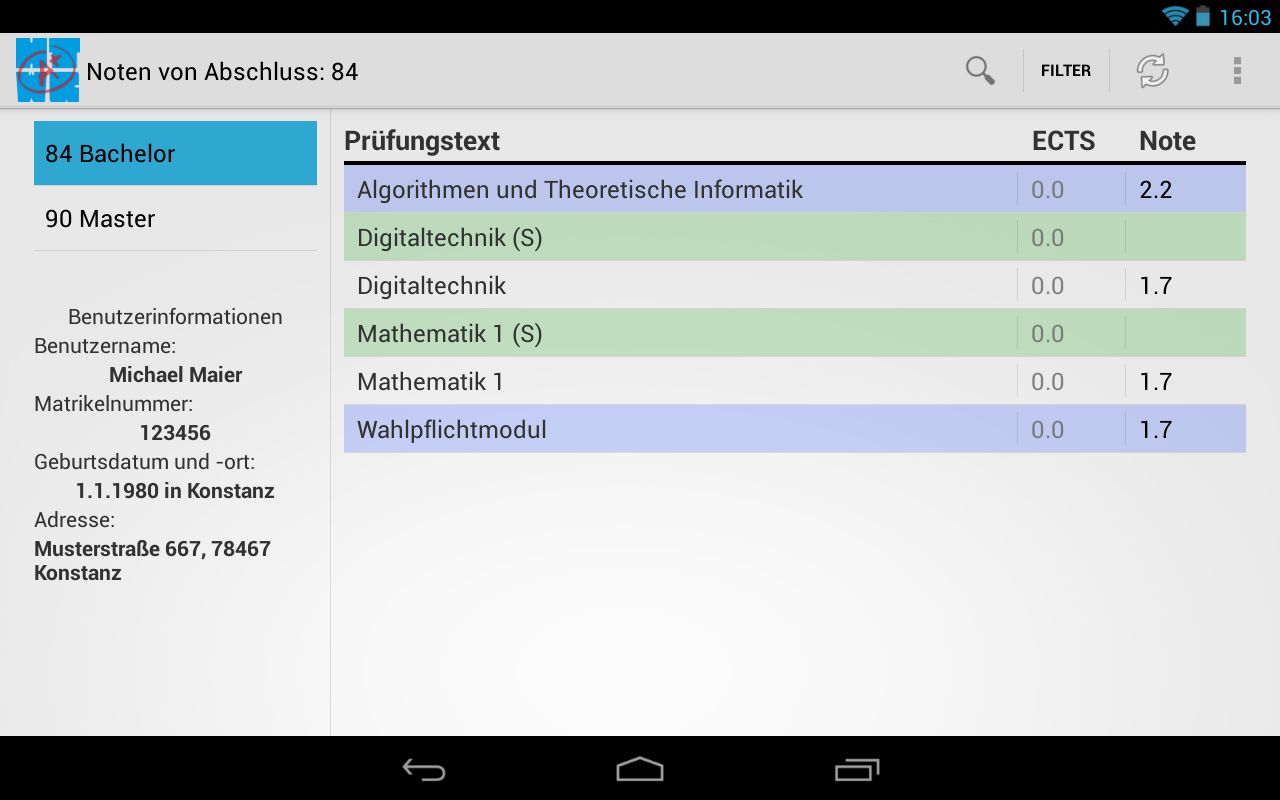
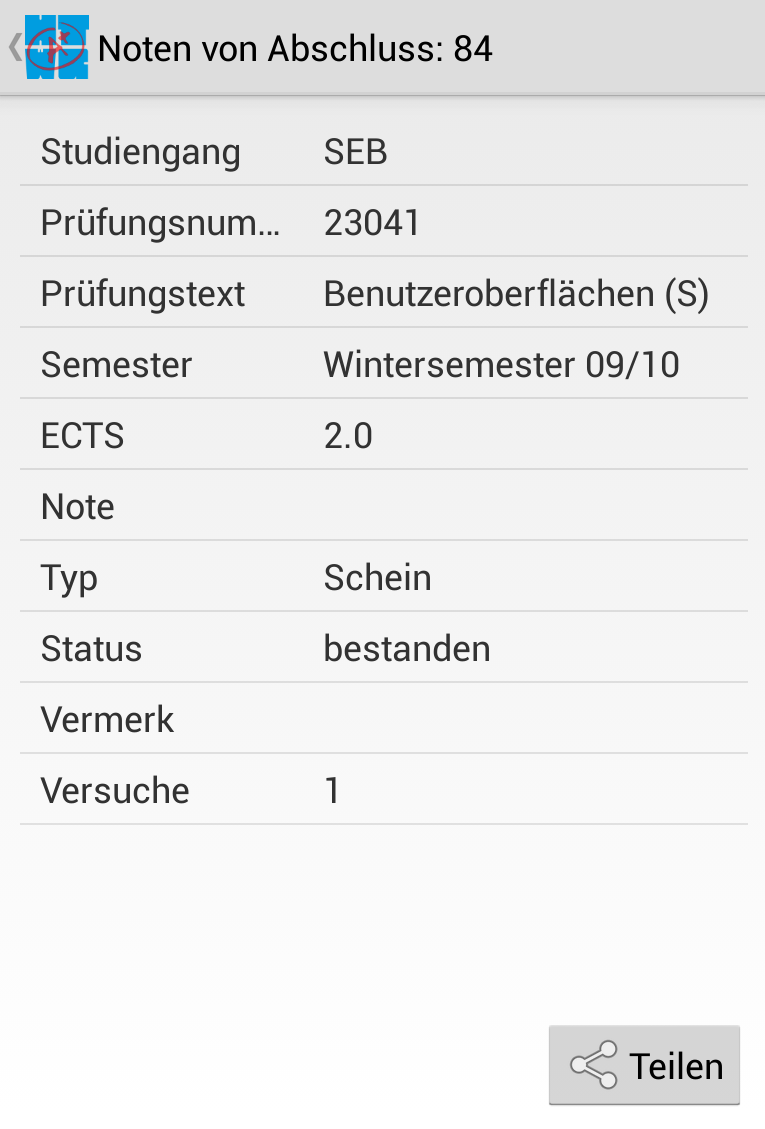


Abbildung 5: Notenansicht. Farbzuordnung der Spalten:  
*Weiß*: benoteter Kurs. *Grün*: Schein. *Blau*: Modul.  
1: Suchfunktion zum suchen nach einem Prüfungstext  
2: Filter zur Filterung nach benoteten Kursen, Scheinen oder Modulen  
3: Aktualisieren  
4: Weitere Optionen (in diesem Fall "Suchverlauf löschen" und "Einstellungen"

Durch das Auswählen einer der Zeilen gelangt man zur Detailansicht, welche im Folgenden beschrieben wird.

TODO: Marker 1-4 anpassen (lassen sich bei mir in Word 2007 nicht löschen/verschieben) -Manu

****

## Noten-Detailansicht

Da in der Notenliste nur begrenzt Platz besteht, werden alle weiteren Details in einer eigenen Übersicht angezeigt. Die zeigt die Details eines Kurses mit dem Prüfungstext "Benutzeroberflächen (S)". Das "(S)" steht hierbei für "Schein".

Für diese Abbildung wurde die "Portrait"-Darstellung gewählt, um zu verdeutlichen, dass es durch mangelnden Platz zu abgeschnittenem Text kommen kann. Durch das Wechseln zu "landscape" kann man den Text komplett lesen.

Abbildung 6: Details eines Kurses vom Typ "Schein"  
1: Teile-Knopf zum Weitergeben des angezeigten Ergebnisses

## Internationalisierung

Die App liegt in zwei Sprachen vor, Deutsch und Englisch. Zur Übersetzung wurden die von Android vorgesehenen Mechanismen verwendet. Es sollte noch angemerkt werden dass nur die App und nicht die Daten selbst übersetzt werden, da das Studententerminal nur deutsche Kursnamen anbietet. Die Sprache kann nicht im Programm geändert werden. Sie wird automatisch von Android beim Programmstart gewählt.

# Bekannte Probleme

In diesem Kapitel wird auf die bekannten Probleme der App eingegangen. Wenn möglich wird dazu gleich auch eine Lösung vorschlagen.

## Geschwindigkeit

Die App benötigt aktuell viel Zeit um die Abschlüsse und Noten aus dem Terminal zu laden. Da keine spezielle API, z.B. in Form eines Web-Services, vorhanden ist, müssen die Daten aus der Webseite selbst ausgelesen werden. Dabei ergeben sich mehrere Gründe für den langsamen Zugriff:

* Es werden immer komplette HTML-Seiten statt nur die gewünschten Daten geladen
* Um an die Daten zu kommen müssen mehrere HTML-Seiten geladen werden
* Der Server an sich ist langsam und braucht für jeden Seitenabruf einige Sekunden

Für dieses Problem gibt es keine Clientseitige Lösung. Der Datenabruf ist zwar gefühlt langsam, aber immer noch um ein vielfaches schneller als ein Mensch der die nötigen Webseiten von Hand abruft. Für einen schnelleren Zugriff müsste die QIS-Software optimiert werden oder auf leistungsfähigere Hardware umziehen. Mit einer speziellen API könnte man das Problem evtl. ebenfalls lösen.

## Reguläre Ausdrücke

Die App benutzt reguläre Ausdrücke um den Quelltext der HTML-Seiten nach den gewünschten Informationen zu durchsuchen. Dabei ergeben sich die folgenden Probleme:

* Wenn die HTML-Struktur der Seite sich ändert, werden nicht mehr alle Noten gefunden
* Wir haben nur mit zwei Accounts getestet. Eventuell gibt es Abweichungen in anderen Studiengängen und Fakultäten, die unsere App nicht berücksichtigt. Dadurch würden nicht alle Noten erkannt werden! Noten aus dem Studium-Generale werden beispielsweise abweichend formatiert.

Leider sind die QIS-Webseiten nicht aus validem XHTML erstellt. Deshalb ist es nicht möglich, die Daten mit Hilfe eines DOM-Trees bzw. mit X-Path zu extrahieren. Sollten zukünftige Versionen des QIS-Servers valides XHTML ausgeben, könnte man dies ändern und so einige der Probleme vermeiden.

## Tablet-Modus

Da die Technik mit Fragmenten zu arbeiten noch vergleichsweise neu ist, haben wir uns zur Entwicklung und zum Testen vor allem auf das Smartphone-Layout konzentriert. Dabei wird immer nur ein Fragment pro Activity angezeigt. Der Tablet-Modus mit zwei Fragmenten (aufgeteilt in linkes und rechtes Fragment) wurde aus diesem Grunde nicht so intensiv getestet, da es sich sonst um den doppelten Aufwand gehandelt hätte.

Ein Fehler welcher uns in diesem Zusammenhang aufgefallen ist, taucht im Zusammenhang mit der Funktion des *Teilen*s auf. Nutzt man den Teilen-Knopf in der Kurs-Detailansicht und geht nach dem Teilen mit Hilfe des *Zurück*-Knopfes zurück zur Kursliste, werden bei weiterer Nutzung des *Zurück*-Knopfes und dem darauffolgenden Auswählen eines anderen Abschlusses, die Kurslisten der beiden Abschlüsse übereinander angezeigt. Das zuerst ausgewählte Fragment erscheint transparent im Hintergrund. Durch den Wechsel in die Android-Umgebung, ohne die App zu beenden, sowie einen wiederaufruf der im Hintergrund aktiven App, wird der Anzeigefehler behoben in dem das Fragment neu gezeichnet wird.

Da unsere Priorität auf dem Smartphone-Layout liegt, mussten wir aus zeitlichen Gründen auf eine genauere Analyse verzichten. Der Fehler ist reproduzierbar und würde in einer weiteren Version korrigiert werden.

## Sonstige Probleme

Viele Noten und Kurse tauchen doppelt im Terminal auf. Dieses Problem wird durch eine nachträgliche Filterung der Daten in der App gelöst. Auch wird teilweise im Master das PSS aus dem Bachelor mit angezeigt. Die Ursache für diese Probleme ist nicht bekannt.

Neben doppelten Datensätzen werden auch fehlerhafte Datensätze im Terminal angezeigt. Da die App nur auslesen und verwerten kann, was auf der HTML-Seite gefunden wird, tauchen falsche Einträge auch in der Notenliste der App auf.

# Verbesserungsmöglichkeiten

Da der zeitliche Rahmen des Projekts auf wenige Wochen beschränkt ist, konnten nicht alle Ideen umgesetzt werden und somit besteht Raum für Verbesserungen, welche im Folgenden erläutert werden.

## Anordnung der Noten

Eine Funktion, um die Einträge der Liste neu anzuordnen, ist aktuell nicht implementiert. Die lexikographische Sortierung nach Prüfungstexten ist nicht veränderbar. Eine logischere Anordnung nach Modulen oder anderen Zusammenhängen ist im Weiteren vorgesehen. Die Noten nach Semestern zu sortieren, wäre eine Möglichkeit, welche auch umsetzbar ist. Eine weitere Idee, die Noten zusätzlich nach Modulen zu gruppieren, ist auch in Zukunft keine Option. Da der QIS-Server keinen Web-Service oder Ähnliches zur Anfrage von Daten zur Verfügung stellt, besteht keine Möglichkeit die zugehörigen Kurse eines Moduls herauszufinden.

## Speichern der gefundenen Daten

Beim Starten der App müssen alle Noten eingelesen werden. Dies bedeutet, es muss eine Internetverbindung vorhanden sein. Während die Einstellungen des Benutzers und Passworts im Speicher des Geräts abgelegt werden, um beim erneuten Start keine Informationen abfragen zu müssen, werden die zuletzt gefundenen Abschlüsse und Noten nicht für den Offlinegebrauch gespeichert. Damit sind beim Start ohne Internetverbindung keine Daten verfügbar. Der Vorteil besteht allerdings in den stets aktuellen Daten, welche bei jedem Start aktualisiert werden.

Nachteile:

* Keine Offlinedaten
* Bei jedem Start alle Daten laden

Vorteile:

* Kein Synchronisationsaufwand zwischen Speicher und neuen Daten

Für die kommende Version ist ein Offlinespeicher in Planung. Dieser könnte zum Beispiel mit SQLite umgesetzt werden. Die Anwendung könnte dann beim Start oder in einstellbaren Intervallen prüfen ob Internetzugang vorhanden ist. Ist dies der Fall werden die Daten aktualisiert. Ist kein Internetzugang vorhanden, werden die Daten aus dem Speicher zusammen mit dem Datum der letzten Aktualisierung angezeigt.

## Auslesen weiterer Daten

Zum aktuellen Stand der Entwicklung, werden in der App lediglich Abschlüsse, Noten und der Benutzername angezeigt. Im Studentenportal der HTWG werden allerdings noch weitere Informationen abgespeichert, welche in der App angezeigt werden könnten. Ein Beispiel hierfür ist eine Liste mit den angemeldeten Prüfungen des eingeloggten Benutzers.

Eine weitere sinnvolle Ergänzung wäre das direkte Abspeichern der Immatrikulationsbescheinigung und des Notenspiegels als PDF-Dokument bzw. das direkte Versenden dieser Dokumente per Email.

## Test-Login Funktion

Ein Knopf welcher in den Einstellungen eingebaut werden könnte, um dem Benutzer die Möglichkeit zu geben, seine eingegebenen Daten zu testen.

## Anzeige für Tablet und Smartphone

Die aktuelle Implementierung unterscheidet zwischen großen Bildschirmen und allen anderen Bildschirmgrößen. Damit erhalten Benutzer mit großem Bildschirm automatisch ein zweispaltiges UI angezeigt. Eine feinere Unterscheidung zwischen den unterschiedlichen Bildschirmgrößen ist wünschenswert. Beispielsweise könnte die Anzeige der Abschlüsse in einem Drop-Down-Menu realisiert werden, sodass die Liste der Abschlüsse im Falle eines kleineren Tablets, nicht zu viel Platz einnimmt.

## Berechnung des Notendurchschnitts

Sofern die Daten des Notenterminals korrekt und zuverlässig wären, also eine Trennung von Grund- und Hauptstudium sowie die Verknüpfungen zwischen Modulen und ihren Teilprüfungen zur Verfügung stehen würden, könnte man eine Funktion zur Berechnung des aktuellen Notendurchschnitts bereitstellen.

# Zusammenfassung

Rückblickend lässt sich sagen, dass wir bei diesem Projekt viel gelernt haben. Wir hatten keinerlei Vorkenntnisse im Bereich der Entwicklung für mobile Anwendungen, konnten aber von unseren Kenntnissen im Bereich der Softwareentwicklung mit Java profitieren. Auch waren bereits in anderen (privaten) Projekten gesammelte Erfahrungen mit dem Datenzugriff über HTTP bei der Umsetzung der App nützlich.

Die gesetzten Ziele aus der ersten Kurzbeschreibung der App wurden alle erreicht. Zusätzlich haben wir noch ein zweites Layout für Tablets einbinden können, welches den Platz auf größeren Displays besser nutzt. Dies stellte sich auch als die fehleranfälligste Funktion heraus, da Änderungen der Benutzeroberfläche stets auf mehreren Geräten getestet werden musste. Zusätzlich existieren noch verschiedene Displaygrößen innerhalb der beiden Kategorien, was zu weiteren Problemen beim Testen und Implementieren führte.

Wie zu erwarten war, ist der Datenzugriff über HTTP mit dem Parsen von für Browser gedachten HTML-Daten zwar möglich, aber nur begrenzt sinnvoll. Eine spezielle API hierfür wären wünschenswert und würde vieles einfacher gestalten. Auch die Daten selbst im Studententerminal sind nicht fehlerfrei. So existieren manche Fächer doppelt (z.B. als Modul und als Note sowie Schein, oder mit einem Schreibfehler im Namen). Die Zuordnung zwischen Scheinen, Kursen und Modulen ist ebenfalls mangelhaft und selbst für uns Menschen nicht beim Betrachten der Webseite nicht ersichtlich. Somit sind viele Möglichkeiten wie zum Beispiel das Berechnen eines korrekten Notendurchschnitts nicht möglich.

TODO:

* Viel gelernt…
* App geschrieben…
* Probleme mit tablet modus…
* Notenterminal sollte verbessert werden…