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

|  |
| --- |
| Manuel Caputo (282210) und Andreas Bug (282207)  03.01.2013 |

Inhalt

[1. Einleitung 4](#_Toc345185860)

[2. Problemstellung 4](#_Toc345185861)

[3. Architektur 5](#_Toc345185862)

[3.1. System 5](#_Toc345185863)

[3.2. Implementierung und Funktionalität 6](#_Toc345185864)

[3.3. Layouts 7](#_Toc345185865)

[3.4. Serververbindung 8](#_Toc345185866)

[4. Bedienung 8](#_Toc345185867)

[4.1. Einstellungen 8](#_Toc345185868)

[4.2. Informationen und blockierte UI 8](#_Toc345185869)

[4.3. Abschlüsse-Ansicht 9](#_Toc345185870)

[*Unterschiedliche Bildschirmgrößen* 9](#_Toc345185871)

[4.4. Noten-Ansicht 9](#_Toc345185872)

[4.5. Noten-Detailansicht 11](#_Toc345185873)

[5. Bekannte Probleme 11](#_Toc345185874)

[5.1. Geschwindigkeit 11](#_Toc345185875)

[5.2. Reguläre Ausdrücke 12](#_Toc345185876)

[6. Verbesserungsmöglichkeiten 12](#_Toc345185877)

[6.1. Anordnung der Noten 12](#_Toc345185878)

[6.2. Filtern falscher Datensätze 12](#_Toc345185879)

[6.3. Speichern der gefundenen Daten 12](#_Toc345185880)

[6.4. Auslesen weiterer Daten 13](#_Toc345185881)

[6.5. Test-Login Funktion 13](#_Toc345185882)

[6.6. Anzeige für Tablet und Smartphone 13](#_Toc345185883)

# Einleitung

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

Da sich die beiden Teammitglieder mit einer möglichst realistischen Anwendung beschäftigen wollten, welche zudem einen echten Zweck erfüllt, haben sie sich 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.

# Problemstellung

Die Schwierigkeiten des Projekts liegen zum einen in der Verbindung zum Server und dem Auslesen der Daten, zum anderen in der möglichst einfachen Darstellung der Daten.

Das Studententerminal (QIS-Server) bietet keinen Web-Service oder Datenbankzugang für den Datenzugriff an. Um an die Abschlüsse und Noten eines Studenten zu gelangen, muss eine HTTPS-Verbindung erstellt werden. Um anschließend an die gewünschten Informationen zu gelangen, 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 3.1 Layouts 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.

# 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.

## 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 ist Rechte 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 eine Liste mit Abschlüssen, die Studentendaten sowie eine Liste der Kurse des ersten gefundenen Abschlusses nebeneinander 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 3.3 Serververbindung genauer erklärt.

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

Abbildung 2: Nutzung der unterschiedlichen Darstellungsmöglichkeiten

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.

## Serververbindung

TODO: Manu

# Bedienung

In diesem Kapitel wird die UI der App sowie deren Bedienung beschrieben.

## 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.

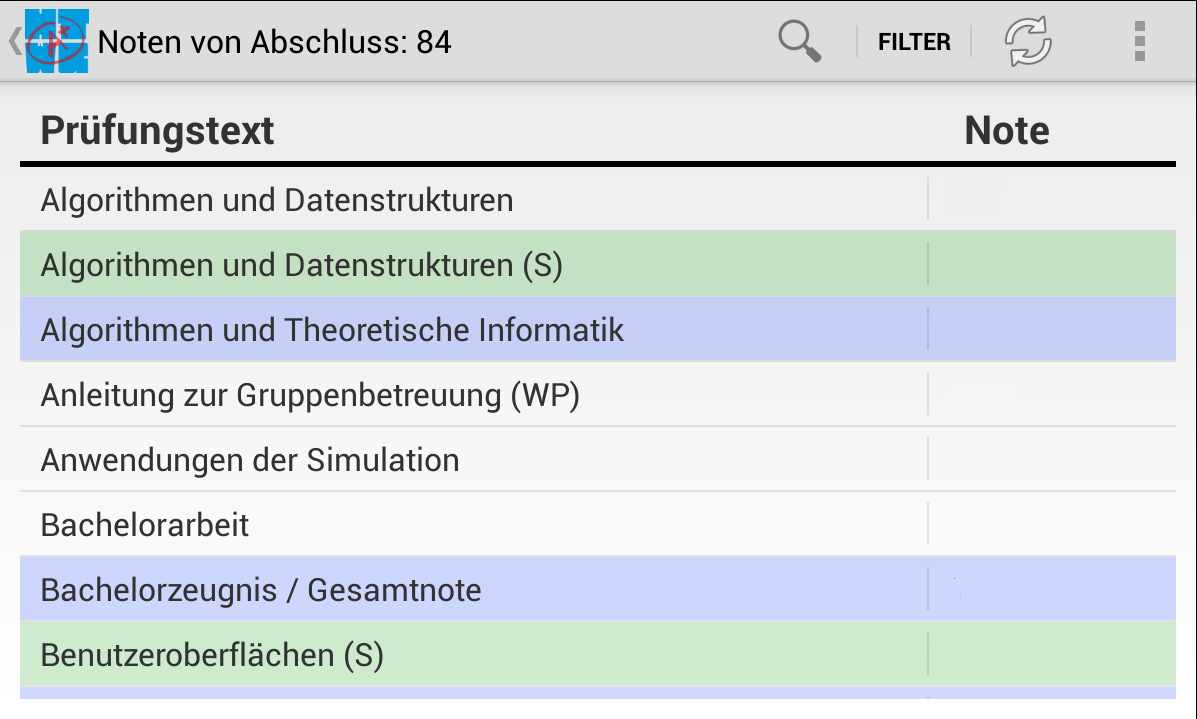


Abbildung 5: Notenansicht. Farbzuordnung der Spalten:  
*Weiß*: benoteter Kurs. *Grün*: Schein. *Blau*: Modul.  
1: Suchfunktion zum suchen nach einem Prüfungstext  
2: Filter zur Filterung nach benoteten Kursen, Scheinen oder Modulen  
3: Aktualisieren  
4: Weitere Optionen (in diesem Fall "Suchverlauf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

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

## 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

# 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 lange 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 langsame 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. Die Datenabruf ist zwar gefühlt langsam, aber immer noch um ein vielfaches schneller als ein Mensch der die nötigen Webseiten von Hand abruft. Für einen schnellere 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 umschiffen.

## 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 sind 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 des Programmes oder in einstellbaren Intervallen prüfen ob Internetzugang vorhanden ist. Ist die 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. Beispiele hierfür sind: Matrikelnummer, Adresse und eine Liste mit den angemeldeten Prüfungen des eingeloggten Benutzers.

## 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.