MyKanti

Die Entwicklung eines Android-Apps für die Kantonsschule Schaffhausen

Persönlicher Stundenplan mit Fach, Zeit, Raum, Lehrkraft und Notizfunktion

Maturaarbeit von Fabian Fischer im Fach Informatik
Betreut von Raphael Riederer
Kantonsschule Schaffhausen
04.12.2012

Inhaltsverzeichnis

1. Vorwort

1.1. Motivation

Ich hatte mir lange überlegt, welches Thema ich wählen soll. Ich wusste schon früh, dass ich meine Maturaarbeit über ein Informatikthema machen wollte, doch ich wollte nicht einfach irgendetwas machen. Ich wollte etwas schreiben, das später möglicherweise auch einmal gebraucht wird, und ich hoffe das habe ich mit meiner App erreicht. Wenn es auch vorläufig sonst niemand braucht, ich verwende es fast täglich.

1.2. Systematik

Es ist immer schwierig, Informatikthemen jemandem zu erklären, der keinerlei Vorkenntnisse hat. Ich habe versucht meine Arbeit so zu schreiben, dass man auch ohne Informatik Kenntnisse versteht, wie es grob funktioniert. Ich gehe zwar teilweise sehr stark auf den Quellcode ein, doch sollte es auch ohne ihn verständlich sein.

1.3. Dank

Bevor ich mit der Arbeit starte, will ich noch zwei Leuten danken, ohne die diese Arbeit, zumindest nicht in dieser Form, entstanden wäre. Zum einen Herr Rainer Steiger, der mir die benötigten Daten unkompliziert zu Verfügung gestellt hat. Zum anderen Sebastian Peyer, der mich beraten hat und mich erst auf die Software Ubuntu Server aufmerksam gemacht hat. Des Weiteren will ich mich natürlich auch bei meinem Betreuer Herr Raphael Riederer bedanken.

1.4.Persönliche Erfahrung

Ich habe im Laufe dieser Arbeit extrem viel gelernt und hatte viel Spass. Ich verstehe nun besser, wie die Apps auf meinem Handy funktionieren. Ich erhielt auch einen tieferen Einblick in den Aufbau eines Servers und in Datenbanken. Wenn ich ehrlich bin, habe ich mir die Entwicklung etwas leichter vorgestellt. Im Laufe dieser traf ich auf so viele Probleme und Fehler, dass ich sie unmöglich alle in dieser Arbeit erwähnen kann. Trotzdem hat es mir sehr viel Spass gemacht, und ich kann mit Sicherheit sagen, dass dies nicht die letzte App sein wird, die ich programmiert habe.

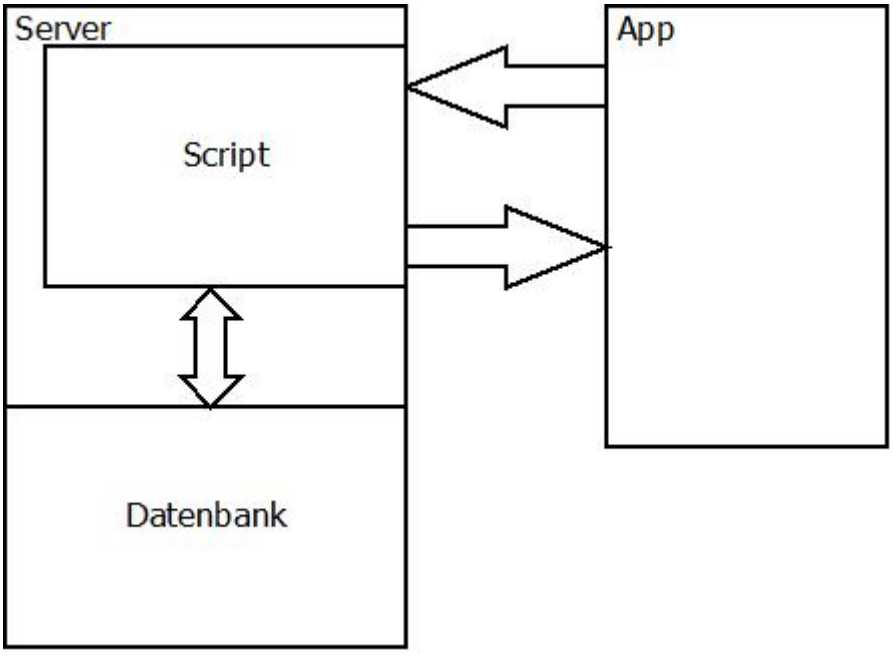
2. Übersicht

2.1. Das Projekt – Was kann es?

Die Funktion meiner App lässt sich relativ knapp zusammenfassen. Jedem Schüler der Kantonsschule Schaffhausen wurden ein Benutzername und ein Passwort zugewiesen. Mit diesen Angaben kann er sich anmelden und er erhält seinen aktuellen und speziell auf ihn zugeschnittenen Stundenplan, ohne dass er Angaben zu seinem Profil, Schwerpunkt- oder Wahlfächern machen muss. Des Weiteren ermöglicht ihm die App, Notizen für jede Lektion zu erstellen. Ausserdem kann er sich abmelden und mit einem anderen Login anmelden oder sein Passwort ändern.

2.2. Das Projekt – Wie funktioniert es?

Die Funktionen der App scheinen auf den ersten Blick sehr simpel und sind schnell erklärt, die Ermöglichung dieser Funktionen ist jedoch ungemein komplexer. Dazu ist nämlich eine Verbindung mit einem Server notwendig. Diese Verbindung lässt sich wohl am besten mit Hilfe eines Diagramms erklären.



Zuerst einmal muss zwischen Front-End und Back-End unterschieden werden. Dies sind zwei gebräuchliche Begriffe in der Informatik, wenn man von Verbindungen spricht. Das Front-End beschreibt typischerweise das Ende der Verbindung, das näher beim Benutzer liegt, in

unserem Fall die App. Das Back-End beschreibt nun konsequenterweise das Ende, welches näher beim System liegt, das die Daten verarbeitet, in unserem Fall das auf dem Server laufende Programm, auch Script genannt.
Der Nutzer gibt seine Login-Daten in die App ein. Diese erstellt eine Verbindung zum Server und übermittelt die Daten. Das Back-End, also das Script, bekommt nun diese Daten und ruft mit ihnen die Informationen zu denjenigen Lektionen ab, die von Interesse sind. Zuletzt werden die Daten zum Front-End geschickt und in eine lesbare Form gebracht.
Dies ist natürlich immer noch eine starke Vereinfachung der eigentlichen Geschehnisse, doch es gibt einen gewissen Überblick. In dieser Arbeit werde ich die einzelnen Aspekte genauer betrachten, und mit Hilfe dieser groben Übersicht und der Grafik, welche später in komplexerer Form wieder auftauchen wird, soll ein Gesamtüberblick entstehen.

3. Theorie

3.1. Sprachen

3.1.1. Java

Java ist eine objektorientierte Programmiersprache, entwickelt von Sun Microsystems. Erklären, was objektorientiert genau heisst, ist etwas schwierig und meist erkennt man das auch erst wirklich, wenn man einmal damit arbeitet. Vereinfacht gesagt, heisst objektorientiert, dass der Quellcode in Klassen aufgebaut ist. Klassen kann man sich wohl am einfachsten als eine Art Maschinen vorstellen. Meist werden sie mit Autos verglichen. Jede Klasse hat verschiedene Attribute, wenn man dies mit einem Auto vergleicht, wären das Informationen zu Farbe, Form, Gewicht, etc., und Methoden, beim Auto bremsen, Gas geben, lenken, etc.. Diese Art des Programmierens lässt den Quelltext sehr übersichtlich erscheinen und spart Code. Java ist eine sehr beliebte Programmiersprache, da man mit ihr Programme schreiben kann, die auf fast allen Systemen funktionieren. Erreicht wird das durch die Java-Technologie, die grundsätzlich aus dem Java Development Kit (JDK), also dem Java-Entwicklungswerkzeug, und der Java Runtime Environment (JRE), also der Java-Laufzeitumgebung aufgebaut ist.
Java wird verwendet, um Programme zu formulieren. Dieser von Menschenhand geschriebene Code, auch Quellcode genannt, kann jedoch nicht direkt ausgeführt werden, denn er ist für das System unverständlich. Erst der im Java Development Kit enthaltene Java Compiler übersetzt diesen dann in Java-Bytecode. Hier liegt auch der Unterschied zu den

meisten anderen Programmiersprachen. Normalerweise ist dieser Bytecode speziell auf das System abgestimmt. Das heisst, wenn man dasselbe Programm auf einem Linux System und auf einem Windows System kompiliert, so sind die Bytecodes unterschiedlich. Bei Java ist das anders, der Java-Bytecode ist auf allen Systemen gleich und läuft somit auch auf allen Systemen. Dies ist extrem nützlich und wohl auch der Hauptgrund für die schnelle Verbreitung von Java.

Die Voraussetzung dafür, dass dieser Bytecode aber auch wirklich läuft, ist die Java Runtime Environment. Die meisten Leute kennen dieses Programm und haben es sehr wahrscheinlich auf ihrem Computer installiert, doch die wenigsten wissen was es genau macht. Die JRE simuliert ein virtuelles System im eigentlichen System, welches dann das Programm ausführt. Jedes Gerät, welches nun über diese Laufzeitumgebung verfügt, kann jedes Programm ausführen, welches in Java geschrieben wurde.

Jede App in Android wird in Java geschrieben. Deswegen verwende ich die Sprache für die eigentliche Applikation auf dem Smartphone. Das Folgende ist ein kleiner Ausschnitt des Apps, geschrieben in Java

```
final Button Do = (Button) findViewById(R.id.Do);
Do.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
       Intent in = new Intent(montagActivity.this,donnerstagActivity.class);
       startActivity(in);
       finish();
       }
});
```

3.1.2. XML

XML oder Extensible Markup Language, zu Deutsch „erweiterbare Auszeichnungssprache“, ist an sich keine Programmiersprache, sondern, wie der Name schon sagt, eine Auszeichnungssprache. Das heisst, sie wird nicht verwendet, um Programme zu definieren, sondern nur zur Darstellung verschiedenster Daten.

In meiner Arbeit wurde XML für die Definition der Benutzeroberfläche des Apps verwendet, sowie zur Deklaration verschiedener Daten. Als Entwicklungsumgebung verwendete ich wieder Eclipse. Es folgt ein kleiner Ausschnitt einer Benutzeroberfläche, geschrieben in XML.

```
<Button
	android:id="@+id/Mo"
	android:text="@string/Montag_kurz"
	android:textColor="#FFFFFF"
	android:gravity="center"
	android:layout_width="fill_parent"
	android:layout_height="fill_parent"
	android:background="#012B16"
	android:layout_weight="1"
	/>
```

3.1.3. Python

Python ist eine leicht zu erlernende Programmiersprache, die üblicherweise interpretiert wird. Eine interpretierte Programmiersprache heisst, dass das Programm nicht wie zum Beispiel bei Java in ein Bytecode kompiliert und dann ausgeführt wird, sondern dass der Quelltext während des Ausführens fortlaufend übersetzt wird. Python wurde mit dem Ziel entwickelt, einfach und übersichtlich zu sein. So wurde die Syntax, also der Aufbau der verschiedenen Befehle, sehr einfach gehalten, und man versuchte möglichst wenige Schlüsselwörter zu gebrauchen. Speziell an Python ist ausserdem, dass man keine Klammern oder Schlüsselwörter zum Abgrenzen von Blöcken braucht, wie das bei anderen Sprachen üblich ist, sondern diese direkt durch das gleiche Einrücken definiert werden. Dies erhöht die Lesbarkeit enorm, da man gezwungen ist, Blöcke genau zu strukturieren. Wegen dieser Einfachheit ist Python vor allem zum Schreiben von Scripts, also kleinen Programmen, die meist auf Servern laufen und aufgrund ihrer Grösse keine komplexen Sprachelemente benötigen, sehr beliebt.

Python wird als Scriptsprache auf meinem Server verwendet. Geschrieben wurden diese Scripts mit der Entwicklungsumgebung IDLE. Unten ist ein kleiner Ausschnitt eines meiner Scripts in Java.

```
Stundenplanstr = ""
for lektionlist in stundenplanstrlist:
        lektionstr = ""
        for infostr in lektionlist:
                lektionstr = lektionstr + "%" + infostr
        stundenplanstr = stundenplanstr + "/" + lektionstr[1:]
```

3.1.4. SQL

SQL ist, genau wie XML, auch keine Programmiersprache, sondern eine Datenbanksprache. Datenbanksprachen werden zur Definition des Aufbaus einer Datenbank sowie zum Bearbeiten und Abfragen von Daten verwendet.

Die Syntax von SQL ist relativ simpel aufgebaut und stark an die englische Sprache angelehnt. SQL wird von fast allen üblichen Datenbanksystemen unterstützt, jedoch unterscheidet sich die Sprache minimal, abhängig vom verwendeten System, man spricht hier üblicherweise von Dialekten.

SQL verwende ich in zwei sehr ähnlichen Dialekten für die MySQL-Datenbank auf dem Server sowie für die SQLite Datenbank auf dem Smartphone.

3.2. Zeichencodierung

In der Informatik wird die Zeichencodierung benutzt, um jedem Schriftzeichen einen eindeutig zugeordneten Zahlenwert zuzuweisen. Dadurch werden Schriftzeichen zur Übertragung oder Speicherung brauchbar. Der dadurch entstehende Zahlenwert ist zwar eigentlich eindeutig, es existieren jedoch verschiedene Zeichencodierungen, und dadurch entstehen immer wieder skurril anmutende Fehler. Man nimmt an, ein „ü“ wird in UTF-8 codiert. UTF-8 ist die am weitesten verbreitete Codierung und wird im Internet und in den meisten Programmen verwendet. Dieses „ü“ wird nun von einem mit Latin-1 arbeitenden Programm gelesen. Latin-1 ist ebenfalls eine weit verbreitete Codierung und wird von vielen unter Windows laufenden Programmen verwendet. Das mit Latin-1 arbeitende Programm weiss nun nicht, dass das ü in UTF-8 codiert wurde und versucht es, als Latin-1 Code zu lesen. Es erhält nun natürlich kein „ü“ sondern die zwei Zeichen „Ã¤“. Man sieht also:

Codierung ist ein Thema, mit dem sich jeder Programmierer früher oder später befassen muss. Auch ich musste mich im Laufe meiner Arbeit ziemlich intensiv mit Zeichencodierung befassen.

3.3. Linux

Spricht man von Linux, denken die meisten Leute an ein Desktop-Betriebssystem, das ist jedoch nicht ganz richtig, es gibt nämlich kein Betriebssystem, das Linux heisst. Linux ist nur ein Betriebssystemkern, also der zentrale Bestandteil eines Betriebssystem, der direkt auf die Hardware zugreift und auf dem der Rest des Betriebssystems aufgebaut wird. Linux ist Open-Source, das heisst, jeder darf auf den Quellcode zugreifen und ihn verwenden. Deswegen gibt es wohl auch so viele Betriebssysteme, die darauf aufbauen.
Ich werde zwei Betriebssysteme, die auf Linux basieren und ich in meiner Arbeit verwendet habe, vorstellen. Das Smartphone Betriebssystem „Android“ und das Server Betriebssystem „Ubuntu Server“.

3.3.1. Android

2005 übernahm Google die Firma Android, welche zuvor bereits Handy-Betriebssysteme entwickelte, um ein offenes Betriebssystem für Mobilgeräte bereitzustellen. Im November 2007 gründete Google die Open Handset Alliance zusammen mit namhaften Netzbetreibern, Geräteherstellern und Software-Firmen. Am 21. Oktober 2008 war es dann vollbracht und Android wurde für jeden Hersteller von Mobilgeräten verfügbar. Seit 2010 ist Android das mit Abstand am weitesten verbreitete Betriebssystem für Smartphones.
Der Aufbau von Android ist etwas speziell, das Wissen darüber ist jedoch für den weiteren Verlauf der Arbeit wenigstens im Ansatz von Nöten. Das eigentliche System läuft auf der Basis vom Linux Betriebssystemkern. Darauf laufen die Befehlsbibliotheken und andere notwendige Grundprogramme. Zum Schluss laufen dann darauf aufbauend die Applikationen, also die Apps. Alles was man sieht, wenn man ein Android-Gerät bedient, sind Apps, also nicht nur diese Programme, die man herunterlädt, sondern auch das Menü sind Apps. Das spezielle an diesen Apps ist, dass sie alle abgeschottet voneinander laufen und sie jegliche Zugriffe auf Dinge ausserhalb der App, wie das Internet oder Datenbanken anderer Apps, im sogenannten Android Manifest anmelden müssen, und diese vom Benutzer

bewilligt werden müssen. Dies macht Android entgegen seines Rufs zu einem sehr sicheren Betriebssystem.

Eine App wiederum ist hauptsächlich aus verschiedenen Activities aufgebaut. Eine Activity ist eine Java Klasse, welche nur einen kleinen Teil der eigentlichen App steuert. Normalerweise steuert eine Activity genau eine Benutzeroberfläche. Also eigentlich eine „Seite“, die der Benutzer sieht, diese ist in XML definiert. Eine App ist also nicht eine Einheit, sondern jedes Mal, wenn die Benutzeroberfläche sich markant ändert, ist eine andere Activity am Werk. Neben den Activities gibt es noch andere Klassen, welche als Schnittstelle dienen oder Arbeiten im Hintergrund verrichten. Von diesen kriegt der Benutzer selbst eigentlich nie etwas mit.

3.3.2. Ubuntu Server

Ubuntu ist ein auf Linux basierendes Open-Source Betriebssystem für PCs. Ubuntu ist die meist genutzte Linux-Distribution der Welt und ist wegen ihrer leichten Bedienbarkeit beliebt. Ubuntu Server ist, wie der Name schon sagt, eine speziell für Server abgestimmte Version des Betriebssystems. Das bedeutet, es besitzt keinerlei graphische Oberfläche, was es schneller macht. Des Weiteren ist es verstärkt auf Sicherheit ausgelegt.

4. Entwicklung

4.1. Aufsetzen eines Servers

Schon zu Beginn meiner Arbeit war mir klar, dass ich die Daten von einem Server abrufen will. Das ist zwar eigentlich nicht zwingend notwendig, da sich der Stundenplan nur relativ selten ändert und man das vermutlich auch anders lösen könnte, doch mich interessierte die Kommunikation mit einem Server. Ich sah mich also nach einem Server um. Einen Platz auf einem externen Server zu mieten, war mir zu teuer, also beschloss ich, nach einem Tipp eines Freundes, mir aus einem alten Dell PC und der Open Source Software Ubuntu Server selbst einen Server zu basteln. Da Ubuntu Server eigentlich jegliche Software mitbringt, die ich benötige (MySQL, Apache, Python), war der Server schnell aufgesetzt, und ich hatte nur kleinere Probleme mit der Hardware. Das Laufwerk meines PCs wurde nicht erkannt. Jedoch konnte ich dieses Problem leicht umgehen, indem ich das Betriebssystem von einem USB-Stick installiert habe.

Der Server war nun funktionsfähig, doch erreichbar war er noch nicht. Genaugenommen erreicht man ihn nur innerhalb des LANs, des Lokal Area Network, also dem lokalen Netzwerks, welches durch einen Router verbunden wird. Versuchte man, ihn über das Internet zu erreichen, funktionierte das nicht. Der Grund war folgender: eine IP, welche zur Identifikation im Internet dient, verweist nicht direkt auf einen Computer, sondern nur auf das LAN, in dem er sich befindet. Wenn man also versucht den Server zu erreichen, erreicht man eigentlich den Router des LANs. Um dieses Problem zu umgehen brauchte ich einen neuen Router, welche einer Funktion namens IP-forwarding unterstützt. Damit kann man Anfragen auf gewisse Ports, das sind Teile der Netzwerkadresse, welche für verschiedene Verbindungen verwendet werden, auf einen bestimmten Rechner im LAN weiterleiten. Also leitete ich den Port 80, welcher normalerweise für Anfragen auf Webserver verwendet wird, auf meinen Server weiter, und schon war er auch über das Internet erreichbar.

Nun war der Server soweit bereit, jedoch war er immer noch schlecht erreichbar, denn ich hatte noch keine Domain. Das heisst, er ist nur über seine IP-Adresse erreichbar, welche sich von Zeit zu Zeit ändern kann. Glücklicherweise unterstützt mein neuer Router eine Funktion namens dynamischer Namensservice, oder dynamic DNS. Mit Hilfe dieser Funktion lässt sich sicherstellen, dass der Server immer erreichbar ist, auch wenn sich die IP ändert. Dazu meldet man sich bei einem dynamic DNS Anbieter an und reserviert einen Domainnamen. Wenn sich nun die IP ändert, meldet das der Router dem Anbieter und leitet dann Anfragen auf die reservierte URL auf die geänderte IP weiter. Ich aktivierte also diese Funktion und reservierte die URL www.rietwies.com.

Nun war mein Server funktionsfähig und erreichbar, doch wirklich angenehm zu bedienen war er nicht. Denn damals war er nur über das Terminal bedienbar, was für mich zu diesem Zeitpunkt durchaus ein Problem war, denn ich hatte nur begrenzte Kenntnisse von der Serverstruktur und die Masse an neuen Befehlen überforderte mich. Ausserdem war der Austausch von Dateien noch sehr mühsam. Ich suchte also ein Tool, welches mir erlaubt, Einstellungen ausserhalb des Terminals zu verändern und Daten auszutauschen. Ich wurde fündig in einem Programm namens Webmin. Webmin erlaubt, nahezu die gesamte Administration über den Browser und von jedem anderen Computer zu tätigen. Man kann Software updaten, neue Users erstellen und vieles mehr. Ausserdem erlaubt es die Steuerung einer weiteren Software namens Samba. Samba erlaubt einen Austausch von Daten durch das LAN mit Windows Systemen. Nun konnte ich den Server leicht konfigurieren

und Daten austauschen. Doch ganz ohne Terminal funktioniert es nicht, und damit ich nicht ausschliesslich dafür eine Tastatur und einen Bildschirm brauche, installierte ich Putty. Putty ist ein Programm für Windows PCs, mit dem man das Terminal von ausserhalb des Servers bedienen kann. Der Bildschirm ist nun nicht mehr von Nöten, und ich kann den Server bequem von meinem Laptop aus bedienen.

4.2. Der Aufbau der Datenbank

Die MySQL-Datenbank auf dem Server bildet das Zentrum des Projekts. Auf ihr werden die notwendigen Daten gespeichert und abgerufen. Aufgebaut wird diese Datenbank nicht von Hand, sondern mittels eines Python Scripts. Dieses Script liest die Daten, welche ich als Text-Datei von der Schule erhalten habe, und speichert sie in Tabellen ab.

Die Daten erhielt ich als Textdateien. Genauer habe ich zu den Fächern, Klassen, Kurswahlen, Lehrern, Räumen, Schülern und zum eigentlichen Stundenplan je eine Textdatei bekommen. Diese Dateien sind automatische Exporte eines Programms zur Erstellung von Stundenplänen und im Allgemeinen gleich aufgebaut. Am Beispiel des Stundenplans kann ich erklären wie.

```
…
10238#"4nd"#"Su"#"SPM_PF"#"D_"#4#6##
10238#"4nd"#"Su"#"SPM_PF"#"D_"#2#9##
10241#"1sd"#"Bam"#"M_GF"#"367"#5#10##
ID#Klasse#Lehrer#Fach#Zimmer#Tag#Lektion##
…
```

Dies ist ein kleiner Ausschnitt[1] dieser 3066 Zeilen langen Textdatei. Wie man sieht, besteht jede Zeile aus Daten, die mit einem #-Zeichen getrennt sind. Die Zahl zu Beginn ist eine Programm interne ID und ist für mein Projekt soweit unerheblich. Danach kommt die Klasse, der Lehrer, das Fach, die Zimmernummer und die zwei letzten Zahlen geben an, die wievielte Lektion an welchem Tag es ist.

Nun sind diese Daten in dieser Form nicht brauchbar und müssen in eine Datenbank geschrieben werden. Dies wird erreicht durch ein Programm namens „database_update_mysql.py“. Dieses Programm wird immer dann aufgerufen, wenn die Datenbank erstellt oder erneuert wird. Also wird zuerst die alte Datenbank, falls sie existiert, gelöscht und eine neue aufgebaut. In dieser werden dann fünf Tabellen erstellt und gefüllt.

[1] Die Daten wurden zur Anonymisierung verändert

Je eine für die Schüler, den Stundenplan, die Kurswahl, die Fächer, die Lehrer und die Login-Daten. Wie zuvor werde ich das Prinzip an einer der Tabellen aufzeigen.

```
print("22_august_stundenplanreader start")
data = codecs.open("22_august_export_stundenplan.TXT","r", "iso-8859-15")
connection = mdb.connect("localhost",BENUTZERNAME,PASSWORT,"stundenplandb")
cursor = connection.cursor()

for zeile in data:
        stundenplan = zeile.split("#")
        ID = stundenplan[0].strip( "\"" )
        klasse = stundenplan[1].strip( "\"" )
        lehrer = stundenplan[2].strip( "\"" )
        fach = stundenplan[3].strip( "\"" )
        zimmer = stundenplan[4].strip( "\"" )
        tag = stundenplan[5].strip( "\"" )
        lektion = stundenplan[6].strip( "\"" )
        cursor.execute("insert INTO stundenplan (ID,klasse,lehrer,fach,zimmer,tag,lektion) VALUES
(%s,%s,%s,%s,%s,%s,%s)",(ID,klasse,lehrer,fach,zimmer,tag,lektion))
        connection.commit()

  print("stundenplanreader done")
  connection.close()
  data.close()
```

Dieser Programmabschnitt[2] liest die Stundenplan-Daten ein und schreibt sie in eine Tabelle der Datenbank. Mit der ersten Zeile wird eine Ausgabe gemacht, die informiert, dass dieser Teil des Programms begonnen hat. Die zweite öffnet die Textdatei, dann wird die Verbindung zur Datenbank und ein Cursor erstellt. Mit Hilfe einer for-Schleife wird nun jede Zeile gelesen, in ihre Bestandteile zerlegt und mit dem erstellten Cursor in die Tabelle geschrieben. Zuletzt werden die Verbindungen zur Datenbank und der Textdatei getrennt und es wird ausgegeben, dass dieser Teil zu Ende ist. Dies wird nun für jede der fünf Tabellen in analoger Weise wiederholt.

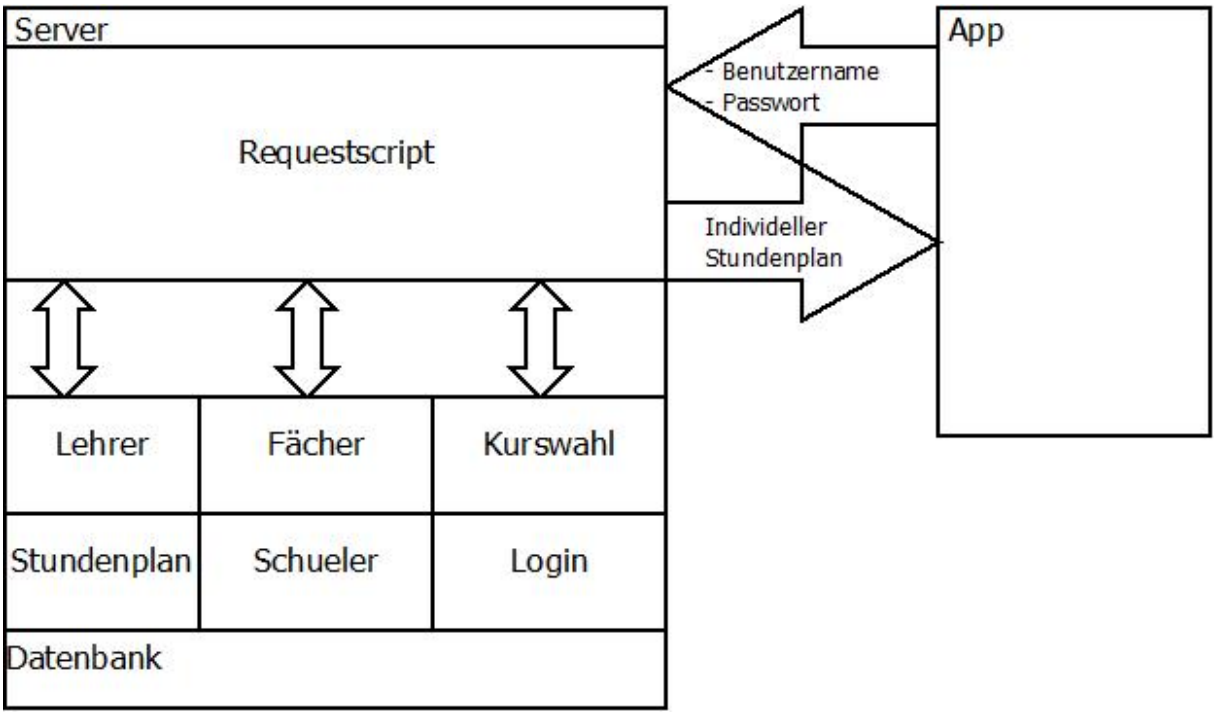
4.3.Script

Nun ist die Datenbank gefüllt, doch eine gefüllte Datenbank bringt nichts, wenn man sie nicht lesen kann. Von der Illusion, man könne einfach so über das Internet darauf zugreifen

[2] Benutzername und Passwort wurden zur Sicherheit entfernt

musste ich mich schon früh verabschieden. Ich fand heraus, dass ich für solch einen Zugriff speziell ein Script schreiben muss. Glücklicherweise ist dieses Problem nicht gerade selten, also gibt es Dutzende, wenn nicht Hunderte von Beispielen. Dummerweise sind solche Programme normalerweise in der Scriptsprache PHP geschrieben, welche ich überhaupt nicht kenne. Einfachheitshalber entschied ich mich, dieses Script in Python zu schreiben. Wie sich herausstellte, wäre das Lernen von PHP etwa gleich aufwendig gewesen, denn es gibt zwar sehr viele, die sich mit dem Schreiben solcher Scripts beschäftigen, hilfreiche Informationen zum Schreiben eines solchen in Python lassen sich wohl aber an einer Hand abzählen. Wie auch immer, ich hatte mich für Python entschieden und nach mehreren frustrierenden Tagen mit Ausprobieren und Internetsuche gelang es mir endlich, online auf die Datenbank zuzugreifen.

Nun war der Grundstein gelegt und ich konnte ein Script schreiben, welches die Daten abruft, die ich benötige. Dieses Programm hat sich im Laufe der Entwicklung oft verändert, vor allem deswegen, weil es stark davon abhängt, was die App sendet und was sie benötigt. Ich werde mich also darauf beschränken zu erklären, wie es im Moment aufgebaut ist. Dafür werde ich die Grafik wiederverwenden, welche ich zu Beginn gezeigt habe.



Die Grafik hat sich nicht wirklich gross verändert, doch sie zeigt ein paar wichtige Details und ist schon etwas komplexer geworden. Zuerst bekam unser Script einen Namen,

naheliegender Weise *Requestscript*, denn es macht „Requests“ an die Datenbank. Des Weiteren sehen wir, welche In- und Outputs das Script hat. Es erhält einen Benutzernamen und das dazugehörige Passwort und sendet den individuellen Stundenplan der anfragenden Person. Die Darstellung der Datenbank hat sich auch verändert, sie zeigt die verschiedenen Tabellen, aus denen sie besteht und lässt erahnen, wie das Script funktioniert.

Ich werde in diesem Fall darauf verzichten, genauer auf den Quelltext einzugehen. Der gesamte Quelltext ist auskommentiert auf der angehängten CD und kann dort betrachtet werden.

Ich werde aber trotzdem versuchen, die Arbeitsweise des Programms zu erläutern. Zu Beginn erhält das Script den Benutzernamen und das Passwort durch einen „HttpPost Request“ von der App, wobei ich im Laufe der Erklärung der App noch genau auf diesen Request zurückkommen werde. Diese eingegebenen Daten werden dann zuerst überprüft und falls diese nicht stimmen, wird sofort eine Fehlermeldung zurückgegeben. Falls sie aber stimmen, wird mit dem Benutzername die ID des Schülers abgerufen. Mit dieser ID werden dann der Vor- und Nachname sowie die Klasse des Schülers geholt. Mit der Klasse kann dann der klassenspezifische Stundenplan gelesen und in eine Liste geschrieben werden. Dieser muss jedoch noch individualisiert werden. Deswegen werden jetzt die Kurswahl und das Geschlecht des Schülers bestimmt. Nun kommt der entscheidende Punkt: Mittels verschiedene Abfragen wird für jedes Fach bestimmt, ob dieses für den Schüler relevant ist. So gibt es zum Beispiel Abfragen, die auf Grund des Geschlechts prüfen, ob die Person in den Frauen- oder Männer-Sport geht, oder ob die Person dieses Schwerpunktfach gewählt hat. Falls dies der Fall ist, werden diese Fächer nochmals abgerufen und in einen String geschrieben. Ein String ist eine Zeichenkette, also eine Art, um Text zu speichern und zu verarbeiten. In diesem String werden die Abkürzungen der Fächer und Lehrer ersetzt durch die ausgeschriebenen Namen, zumindest diejenigen, die in der Datenbank erfasst sind. Zu guter Letzt wird der String zusammen mit der ID, dem Namen sowie der Klasse des Schülers ausgegeben.

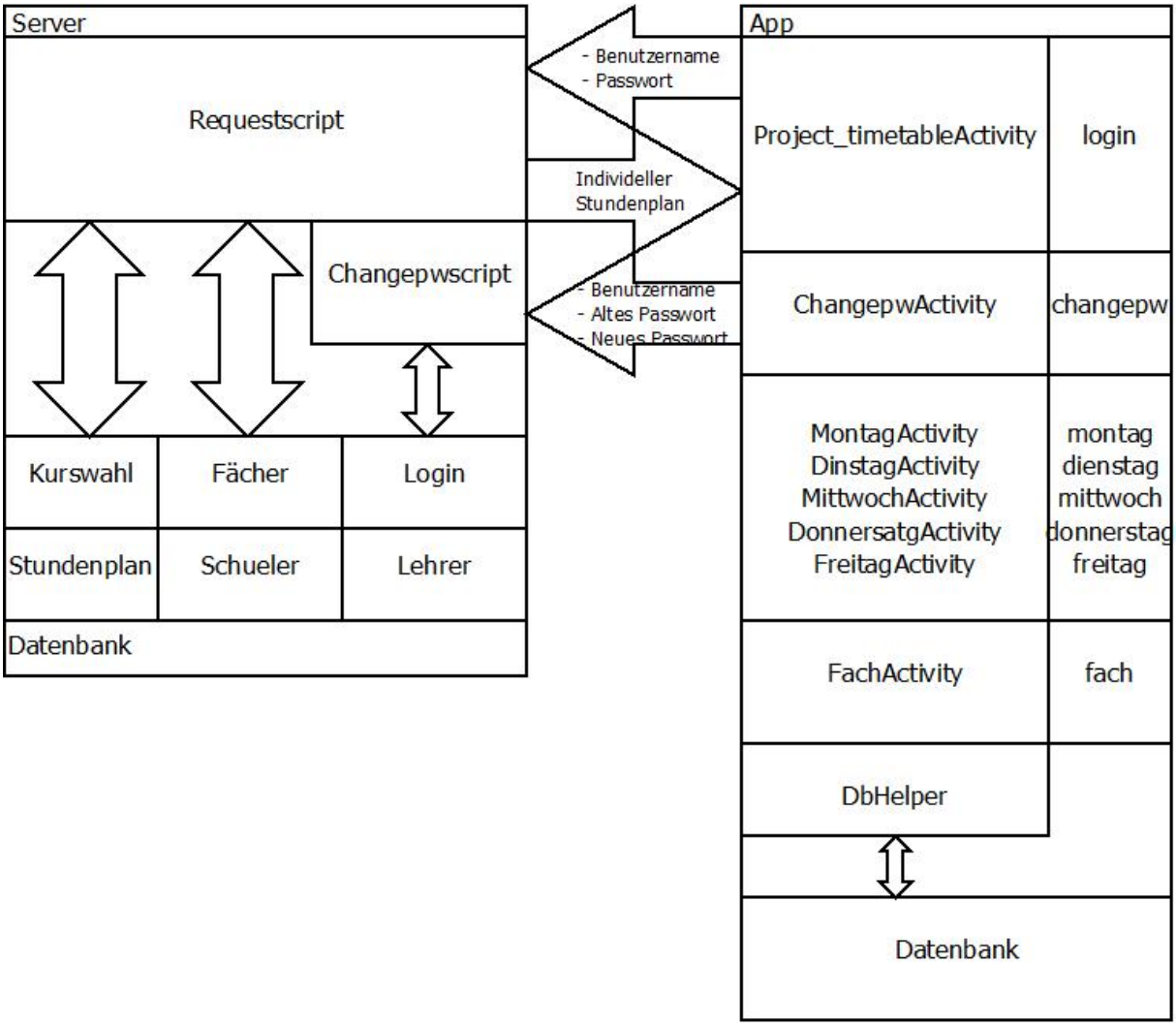
Was ich nun in dieser Erklärung komplett unterschlagen habe, ist die Zeichencodierung, welche ich schon im Kapitel 3.2 angesprochen habe. Die Daten, die ich erhalten habe, sind in „Latin-1“ codiert und wurden auch so auf der Datenbank gespeichert, Android jedoch versteht nur „utf-8“-codierte Daten. Das Requestscript muss also diese Daten noch umcodieren. Zum Glück ist Python dafür gut geeignet und kann dieses Problem mit einer

Zeile Code lösen. Was mir jedoch lange nicht klar war ist, dass Python die Codierung falsch interpretiert, falls man Listen direkt in Strings verwandelt. Dies macht das Schreiben etwas aufwändiger. Die zusätzlichen Zeilen die dadurch entstehen, werde ich hier aber nicht weiter erläutern.

Dieses Programm ist wohl der heikelste Teil meiner Arbeit. Durch Dutzende von Abfragen und durch das eher komplizierte Fächersystem der Kantonsschule gibt es sehr viel Raum für Fehler. Ausserdem werden oft die Lektionen nicht ganz genau so ausgeführt, wie ursprünglich geplant, vor allem mit den Halbklassen. Ich habe lange daran gearbeitet, kann aber leider immer noch nicht 100% garantieren, dass es keinerlei Fehler gibt. Ich werde auch in Zukunft daran arbeiten, falls doch noch ein Fehler auftaucht.

4.4. App

Im bisherigen Text wurde die App als eine Einheit dargestellt, welche einfach so funktioniert. Dies soll sich nun ändern.



Die altbekannte Grafik taucht wieder auf, aber sie ist noch etwas komplexer geworden. Man sieht nun die App in seine Einzelteile aufgetrennt. In diesem Kapitel werde ich auf jeden einzelnen Bestandteil etwas genauer eingehen.

4.4.1. Server Kommunikation

Die beste App bringt nichts ohne eine solide Kommunikation mit dem Server, der sie mit Daten versorgt. Deswegen behandle ich dies auch zu Beginn. Die Serverkommunikation wird fast ausschliesslich von der Activity *Project_timetableActivity* durchgeführt. Diese Activity wird im Android Manifest als Launcher festgelegt, das heisst, dass diese Activity als Erstes

ausgeführt wird, sobald die App gestartet wird. Die eigentliche Kommunikation wird von einer separaten Klasse namens „httpconnect“ übernommen. Verbindungen mit dem Internet, sowie auch mit einer internen Datenbank, müssen von der eigentlichen Activity, welche vor allem das Erstellen der Benutzeroberfläche übernimmt, getrennt sein, sonst führt dies in neueren Android-Versionen zu einem Fehler. Dieser Fehler wird durch das System erzwungen, da es sonst bei längeren Ladezeiten nicht reagieren würde. Bei älteren Versionen gibt es keinen Fehler, jedoch reagiert die Oberfläche, während das Programm lädt, nicht. Durch die Trennung von Kommunikation und Benutzeroberfläche wird erreicht, dass das System immer noch reagiert, auch wenn es grössere Datenvolumen herunterlädt. Das Verschieben grösserer Aufgaben in andere parallel laufende Klassen, auch Threads genannt, ist nicht nur in Android Apps verbreitet, sondern wird in fast jedem Programm verwendet, das über das Internet kommuniziert. Dieses Verschieben nennt man Threading, und Threading ist in Java leider relativ umständlich. Glücklicherweise stellt Android hier ein Hilfsmittel namens *AsyncTask* zur Verfügung. *AsyncTask* erlaubt einem, auf relativ einfache Weise Prozesse in den Hintergrund zu verschieben und Daten zwischen den Threads auszutauschen.

```
public class httpconnect extends AsyncTask<String, Integer, String>{

	@Override
	protected String doInBackground(String... arg0) {
		//eigentlicher Prozess
		return null;
	}

	@Override
	protected void onPostExecute(String result) {
		//Aufgabe nach dem eigentlichen Prozess, Änderung der
		//Benutzeroberfläche hier möglich
		super.onPostExecute(result);
	}

}
```

Dies ist die Grundstruktur einer einfachen *AsyncTask*-Klasse und auch die Grundstruktur der der Klasse *httpconnect*. Man sieht hier zwei von fünf möglichen Methoden einer *AsyncTask*-Klasse, *doInBackground()* und *onPostExecute()*. Die Funktion dieser zwei Methoden steckt eigentlich schon im Namen. *doInBackground()* führt den eigentlichen langen Prozess im Hintergrund aus und *onPostExecute()* wird ausgeführt, sobald der Prozess beendet ist und

kann, im Gegensatz zu *doInBackground*(), eine Änderung der Benutzeroberfläche vornehmen.

Im Falle von *httpconnect* wird mit Hilfe der *doInBackground()* Methode mit dem Server kommuniziert. Dies ist ein eher komplizierter Vorgang, ich werde versuchen, es etwas zu vereinfachen. Zu Beginn erhält die *doInBackground()*-Methode von der Project_timetableActivity Informationen zu Benutzername und Passwort. Wie dies genau geschieht wird später im Kapitel 4.4.3.1 Login behandelt. Diese Informationen werden dann in ein Array, also eine Liste, geschrieben. Nun hilft eine weitere bereits in Java enthaltene Klasse namens *HttpClient*. *HttpClient* erlaubt einem, verschiedene Requests an einen Host zu tätigen. In unserem Fall machen wir einen *HttpPost*-Request. Ich will hier nicht genauer auf das http, also das Hypertext Transfer Protocol eingehen, da ich selbst auch kein Experte auf dem Gebiet bin. Wichtig ist nur, dass ein *HttpPost*-Request Daten an einen Server schickt, dieser kann sie empfangen, verarbeiten und andere Daten zurückschicken. Mit Hilfe des *HttpClient* können wir also nun einen *HttpPost*-Request an das R*equestscript* auf dem Server vorbereiten, und das erstellte Array mit den Login Daten senden. Das Script wird nun ausgeführt, und die Daten zum Stundenplan werden zurückgeschickt. Jedoch leider nicht in String-Form, sondern als sogenannter Stream. Die Daten werden nämlich in kleine Stücke, sogenannte Chunks, zerlegt und fortlaufend gesendet. Dieses Vorgehen namens Chunked Transfer-Encoding hat den Vorteil, dass der Empfänger, also die App, nicht wissen muss wie gross das gesendete Datenpaket ist. Der Nachteil ist, dass man den empfangenen Stream von Chunks noch zusammensetzen muss. Dies ist jedoch, zumindest in Java, relativ einfach. Mit Hilfe der in Java enthaltenen Klassen *BufferedReader* und *StringBuilder* lässt sich der Stream in einen String verwandeln. Dieser String wird dann noch aufgeteilt und in die lokale Datenbank geschrieben. Wie dies funktioniert, wird im nächsten Kapitel erläutert.

Sobald die *doInBackground()* Methode beendet ist, wird die *onPostExecute()* Methode ausgeführt. In dieser wird mit Hilfe der in Java enthaltenen Klasse *calendar* de Wochentag bestimmt und aufgrund dessen die nächste Activity, die *MontagActicity* bis *FreitagActivity*, aufgerufen.

4.4.2. Lokale Datenbank – SQLite

Damit die App auch funktioniert, wenn man einmal keinen Internetzugriff hat, ist es von Nöten, dass man eine lokale Datenbank einrichtet. Android hat bereits ein integriertes Datenbanksystem Namens SQLite. SQLite ähnelt MySQL, welches serverseitig gebraucht wird und ebenfalls in SQL geschriebene Befehle verwendet. Der grosse Vorteil gegenüber anderen Datenbanksystemen ist, dass es keinerlei weitere Server-Software benötigt und vor allem sehr platzsparend arbeitet. Ausserdem werden alle Tabellen, Triggers etc. in einer wenige hundert Kilobyte grossen Datei abgespeichert.

Für die Kommunikation mit dieser Datenbank ist eine Klasse als Schnittstelle von Nöten, genauer gesagt eine *SQLiteOpenHelper*-Klasse, in diesem Fall als D*bHelper* bezeichnet. Diese Klasse ist wie eine *AsyncTask*-Klasse nach einem speziellen Muster aufgebaut und hat vorgegebene Methoden. Sie benötigt mindestens drei Methoden. Die erste Methode trägt denselben Namen wie die Klasse und legt gewisse Kontext-Werte fest wie der Name der Datenbank und deren Version. Die zweite heisst *onCreate()* und wird nur aufgerufen, wenn die App zum allerersten Mal auf die Datenbank zugreifen will. Normalerweise werden hier die Tabellen der Datenbank definiert und erstellt. Die dritte und letzte Methode heisst *onUpgrade()* und wird dann aufgerufen, wenn die Datenbankversionsnummer geändert wird.

```
public class dbHelper extends SQLiteOpenHelper {

	public static final String DB_NAME = "local.db";
	public static final int DB_VERSION = 2;
	public static final String TABLEFACH = "faecher";
	public static final String LEKTION = "lektion";
	public static final String NOTIZ = "notiz";
	public static final String STUNDENPLAN = "stundenplan";
	public static final String KLASSE = "klasse";
	public static final String TABLEINFO = "info";
	public static final String IDINFO = "ID";
	public static final String VORNAME = "vorname";
	public static final String NACHNAME = "nachname";
	public static final String USERNAME = "username";
	public static final String PASSWORD = "password";
	public static final String USED = "used";


	public dbHelper(Context context) {
		super(context, DB_NAME, null, DB_VERSION);
	}

	@Override
	public void onCreate(SQLiteDatabase db) {
		String sqlfach = String.format("CREATE TABLE %s (%s INT PRIMARY KEY
		UNIQUE, %s TEXT)", TABLEFACH, LEKTION, NOTIZ);
		db.execSQL(sqlfach);
		String sqlcount = String.format("CREATE TABLE %s (%s INT PRIMARY KEY
		UNIQUE, %s TEXT, %s TEXT UNIQUE, %s TEXT, %s TEXT, %s TEXT,
		%s TEXT)", TABLEINFO, USED, IDINFO, VORNAME, NACHNAME, KLASSE,
		USERNAME, PASSWORD, STUNDENPLAN);
		db.execSQL(sqlcount);
	}

	@Override
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
		db.execSQL("DROP TABLE IF EXISTS "+ TABLEFACH);
		db.execSQL("DROP TABLE IF EXISTS "+ TABLEINFO);
		this.onCreate(db);
	}

}
```

Dies ist der komplette Code der Klasse D*bHelper* und somit die kürzeste Klasse der ganzen App. Dies wird eben dadurch erreicht, dass der Grossteil der Arbeit vom *SQLiteOpenHelper* übernommen wird. Der obenstehende Code erstellt beim erstmaligen Aufrufen eine Datenbank namens *local.db* und zwei Tabellen. Eine Tabelle namens *faecher*, welche aus den zwei Spalten *lektion* und *notiz* aufgebaut ist. Auf die Funktion dieser Tabelle komme ich im Kapitel 4.4.3.4. Notizen zu sprechen. Die zweite Tabelle heisst *info* und speichert vor allem die Daten zum Stundenplan als einen String ab. Ausserdem beinhaltet sie Informationen zu Vor- und Nachname, Klasse, Login-Daten, Schüler ID und ob sie bereits beschrieben wurde. Letztere Information hat rein technische Gründe und soll auch als statisches Element dienen,

das bei jedem Schüler gleich ist. Bei Änderungen der Datenbankversionsnummer wird *onUpgrade()* aufgerufen. In diesem Falle werden die Tabellen einfach gelöscht und *onCreate()* nochmals aufgerufen.

Es fällt auf, dass der obenstehende Code nur eine Datenbank erstellt und sie nie füllt oder liest. Dies liegt daran, dass dieser Code in der Klasse steht, die etwas abrufen oder abspeichern will. Hier ein Beispiel aus der *Project_timetableActivity*.

```
dbHelper DbHelper = new dbHelper(Project_timetableActivity.this);
SQLiteDatabase db = DbHelper.getWritableDatabase();

ContentValues values = new ContentValues();
Values.put(DbHelper.IDINFO, antworte.split(";")[1]);
values.put(DbHelper.VORNAME, antworte.split(";")[2]);
values.put(DbHelper.NACHNAME, antworte.split(";")[3]);
values.put(DbHelper.KLASSE, antworte.split(";")[4]);
values.put(DbHelper.USERNAME, name[0]);
values.put(DbHelper.PASSWORD, name[1]);
values.put(DbHelper.STUNDENPLAN, antwort);
values.put(DbHelper.USED, i);
Log.i("TAG",Integer.toString(i));

db.insertWithOnConflict(DbHelper.TABLEINFO, null, values,
SQLiteDatabase.CONFLICT_REPLACE);

db.close();
DbHelper.close();
```

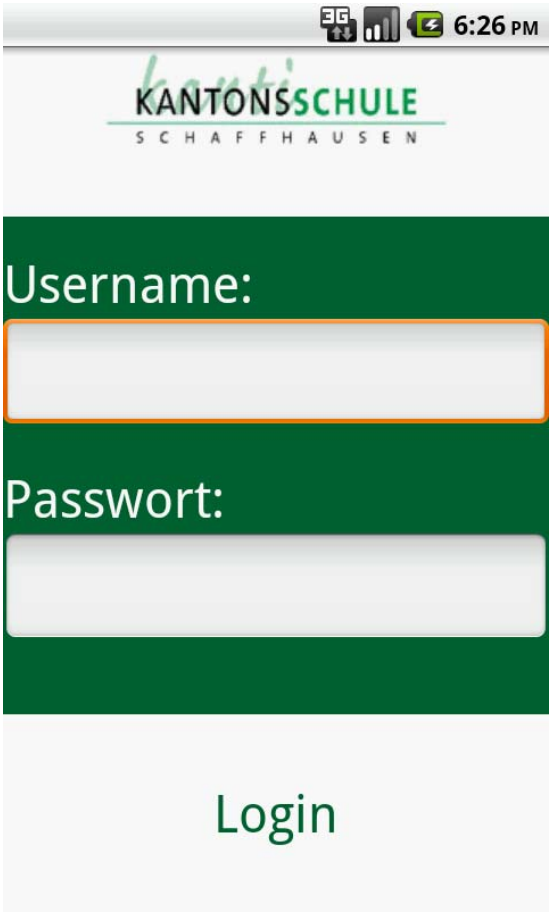
Man sieht hier, dass eine neue sogenannte Instanz von D*bHelper* aufgerufen wird. Dies ermöglicht es uns, Methoden und Objekte dieser Klasse zu verwenden. In diesem Fall benutzen wird die Methode *getWritableDatabase(),* welche zwar nicht in D*bHelper* definiert wird, aber bereits in *SQLiteOpenHelper* enthalten ist. Diese Methode erlaubt es, auf die Datenbank zu schreiben. Dann werden die einzelnen Werte für die Tablle definiert und mit *instertWithOnConflict()* eingesetzt. Zum Schluss werden die Datenbank und die Instanz der Klasse D*bHelper* geschlossen. Das Lesen einer Datenbank funktioniert in analoger Weise, nur mit anderen Methoden.

4.4.3. Benutzeroberfläche

Nun, da alle Hintergrundaktivitäten und Schnittstellen erklärt sind, geht es letztlich um die Benutzeroberfläche, also um denjenigen Teil des Projekts, der direkt mit dem Benutzer interagiert. Unter Benutzeroberfläche fasse ich die durch die XML-Dateien definierte Oberfläche sowie alle Activities, welche diese steuert, zusammen.

4.4.3.1. Login

Das Login sieht man als Erstes. Diese Oberfläche eignet sich gut, um die Grundlagen dieses Kapitels zu zeigen, da sie zum Einem statisch ist, sich also nicht verändert, und zum Anderen eher simpel ist.



login.xml; Der Login-Screen.

Ich habe zwar erwähnt, dass die Oberfläche sehr simpel sei, jedoch ist für die Definition dieser Oberfläche, siehe oben, trotzdem folgenden 79 Zeilen langer XML-Code nötig:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#006633" >


    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="100dp"
    android:orientation="vertical"
    android:background="#FFFFFF" >


         <ImageView
         android:id="@+id/test_image"
         android:src="@drawable/kanti_logo"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         />
    </LinearLayout>
    <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/Username"
    android:textSize="30dp"
    android:textColor="#FFFFFF"
    android:gravity="bottom|left"
    android:layout_weight="1"
    />


    <EditText
    android:id="@+id/editusername"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="30"
    android:layout_weight="1"
    />


    <TextView
    android:layout_width="fill_parent"
         android:layout_height="wrap_content"
    android:text="@string/pw"
    android:textSize="30dp"
    android:textColor="#FFFFFF"
    android:gravity="bottom|left"
    android:layout_weight="1"
    />


    <EditText
    android:id="@+id/editpw"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="textPassword"
    android:ems="30"
    android:layout_weight="1"
    />


    <TextView
    android:id="@+id/fehlerfeld"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="15dp"
    android:textColor="#FF0000"
    android:layout_weight="1"
    />


    <Button
    android:id="@+id/button1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#006633"
    android:background="#FFFFFF"
    android:textSize="30dp"
    android:gravity="center"
    android:text="@string/login"
    android:layout_weight="4"
    />
</LinearLayout>
```

Dies ist nur die Oberfläche ohne Funktion. Die Funktion gibt ihr erst die Activity, welche dahinter steckt, in diesem Fall die bereits im Kapitel 4.4.1 erwähnte *Project_timetableActivity*. Diese Activity wird in jedem Fall als Erstes aufgerufen, wenn die App startet. Dann wird zuerst bestimmt, ob die lokale Datenbank bereits gefüllt ist, also ob sich der User schon einmal eingeloggt hat. In diesem Fall wird das obige Userinterface nicht gezeigt, sondern es werden die Login-Daten von der lokalen Datenbank abgerufen und diese an die Klasse *httpconnect* weitergegeben. Wenn sich der Benutzer noch nie angemeldet oder abgemeldet hat, dann wird das Interface dargestellt, damit er die benötigten Login-Daten eingeben kann.

Betrachten wir zuerst die Oberfläche, welche durch den XML-Code definiert wird. Jedes Interface besteht aus verschiedenen Objekten, auch View genannt, angeordnet in verschiedene Layouts. Diese Layouts sind an sich selbst auch Views. Im Verlauf meiner Arbeit werde ich lediglich das sogenannte *LinarLayout* benutzen. Beim *LinarLayout* werden die beinhalteten Views untereinander oder nebeneinander angeordnet. Hier ein Codebeispiel dieses Layout-Typs.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#006633" >

…

</LinearLayout>
```

Hier werden verschiedene Attribute des Layouts definiert. Wie etwa die Breite und Höhe, welche hier beide mit *match_parent* bezeichnet werden, das heisst, sie nehmen den gesamten zur Verfügung stehenden Raum des Systems auf, in dem sich der View befindet, auch parent genannt, oder die Orientation, also wie die beinhalteten Views aneinander gereiht werden. Die letzte Zeile definiert ausserdem noch die Hintergrundfarbe.

Im Login-Interface werden hauptsächlich drei Arten von Views verwendet: *EditText*, *TextView* sowie *Button*. Ein *EditText* ist eigentlich eine Input Box und dient dazu, Text vom Benutzer aufzunehmen.

```
<EditText
android:id="@+id/editpw"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:inputType="textPassword"
android:ems="30"
android:layout_weight="1"
/>
```

Dies ist einer der enthaltenen *EditText*. Es werden ID, Input Type, Grösse des eingegebenen Textes sowie die gleichen Angaben zu Grösse und Form, wie im *LinarLayout* definiert. Hinzu kommt noch das „Gewicht“, welches die Grösse des Views im Vergleich zu den anderen regelt. Ausserdem wird dieses Mal die Höhe nicht so gross wie möglich, sondern durch den Ausdruck *wrap_content* nur so gross wie nötig. Damit der eingegebene Text nicht einfach irgendwann eingelesen wird, benötigt man noch einen Knopf. Dieser wird in der XML-Datei folgendermassen definiert.

```
<Button
android:id="@+id/button1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textColor="#006633"
android:background="#FFFFFF"
android:textSize="30dp"
android:gravity="center"
android:text="@string/login"
android:layout_weight="4"
/>
```

Die definierten Angaben ähneln stark denen eines *EditText*. Es werden aber zusätzlich noch der Text, seine Attribute, sowie die Hintergrundfarbe des Knopfs definiert. Der letzte Baustein dieser Oberfläche ist der sogenannte *TextView*. Der *TextView* ist eigentlich der einfachste View von allen, denn er ist einfach eine Textbox, in der man Informationen schreiben oder Daten ausgeben kann.

```
<TextView
android:id="@+id/fehlerfeld"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="15dp"
android:textColor="#FF0000"
android:layout_weight="1"
/>
```

Dies ist eine der *TextViews*, auf der später durch die Activity etwas ausgegeben wird. Die Attribute sind grundsätzlich sie gleichen wie bei einem Button.

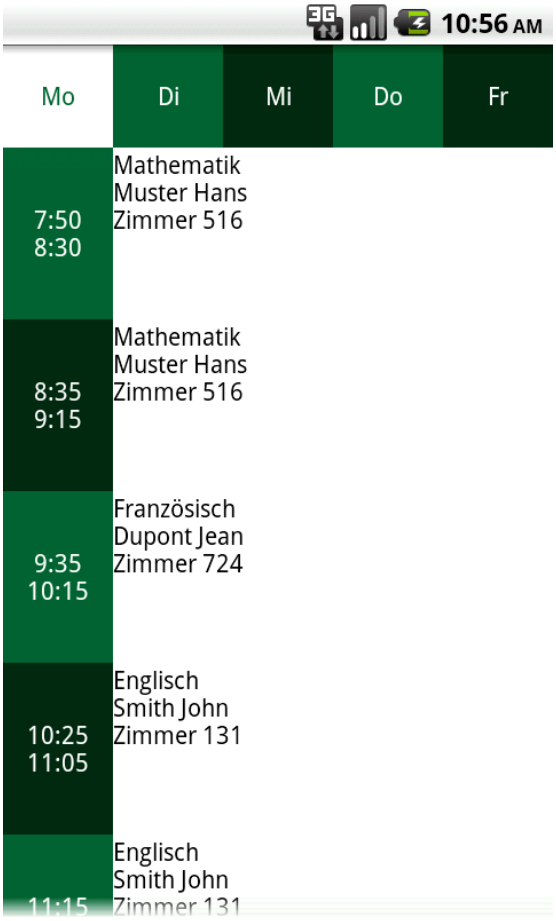
```
setContentView(R.layout.login);
final EditText username = (EditText) findViewById(R.id.editusername);
final EditText password = (EditText) findViewById(R.id.editpw);
final Button send = (Button) findViewById(R.id.button1);
send.setOnClickListener(new View.OnClickListener() {

    public void onClick(View v) {
       String Username = username.getText().toString();
       String Password = password.getText().toString();
       if(Password.equals("")){
             TextView fehlerfeld = (TextView)findViewById(R.id.fehlerfeld);
             fehlerfeld.setText("Passwort oder Username ist falsch");
       }else{
             new httpconnect().execute(Username,Password);
       }
    }
});
```

Dies ist der Ausschnitt der Activity, welche ausgeführt wird, wenn sich der Benutzer anmeldet und auf die Views verweist. Zu Beginn, mit der ersten Zeile, wird das Layout *login.xml* aufgerufen. Die nächsten zwei Zeilen beziehen sich auf die zwei enthaltenen *EditText* in der XML-Datei. Mit Hilfe der in XML definierten ID wird auf den richtigen View verwiesen. Mit der *getText()* Methode kann man den darin eingegebenen Text einlesen. Dies soll jedoch erst geschehen, wenn der Knopf gedrückt wird. Dafür verwendet man die in Android enthaltene Funktion *OnClickListener*. Mit Hilfe derer kann man die Methode *onClick()* benutzen um Code auszuführen, welcher auszuführen ist, sobald der Knopf gedrückt wird. In diesem Fall wird der Text aus den beiden EditText ausgelesen, mit der Methode *toString()* in einen String verwandelt und danach findet die erste Fehlerüberprüfung statt. Falls das Passwortfeld leer ist, wird der Text des TextViews *fehlerfeld* in eine Fehlermeldung geändert. Dies geschieht auf eine analoge Art, wie bei der Arbeit mit den anderen Views. Zuerst wird auf die richtige View verwiesen und dann mit der Methode *setText()* den Textinhalt verändert. Falls das Passwortfeld aber nicht leer ist, wird die Klasse *httpconnect* aufgerufen, und ihr die zwei entstandenen Strings „Username“ und „Password“ geschickt.

Dies sind eigentlich schon die Funktionen des Logins. Es enthält noch ein Bild, doch darauf werde ich nicht genauer eingehen; die Angabe im Quellcode sollten ausreichen um dies zu verstehen. Im späteren Verlauf werden EditText, Knöpfe sowie TextView nochmals vorkommen, ich werde dann das Verständnis dieser voraussetzen.

4.4.3.2. TagActivity

Das Login führt direkt zu den sogenannten *TagActivities* weiter, das sind *MontagActivity* bis *FreitagActivity*. Diese sind das Kernelement der App und der Benutzer wird sich vorwiegend in diesen Activities bewegen.



montag.xml; Der Stundenplan Montag

Wir betrachten wiederum zuerst die Oberfläche, definiert durch den XML-Code. Dieses Mal ist er etwas komplexer als beim Login, trotzdem ist er aus fast den gleichen Bestandteilen aufgebaut. Das Ganze UI(User-Inteface) ist in einem *LinarLayout* aufgebaut. Die obere Leiste ist fix und besteht aus vier Buttons und einer *TextView*, aufgereiht in einem horizontalen *LinarLayout*. Dann kommt das einzige neue Element, die *ScrollView*.

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
>

</ScrollView>
```

Die *ScrollView* ermöglicht, wie der Name sagt, durch die Views zu scrollen. Dies ist auch nötig, denn er Rest der Oberfläche besteht aus 12 Linien, die alle folgendermassen aufgebaut sind.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="100dp"
android:orientation="horizontal" >

    <TextView
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:text="@string/time1"
      android:textColor="#FFFFFF"
      android:gravity="center"
      android:background="#006633"
      android:layout_weight="4"/>

    <Button
        android:id="@+id/monfach1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="1"
        android:background="#FFFFFF"
        android:gravity="left"
        android:text="@string/fach1" />

</LinearLayout>
```

Jede Linie ist also ein *LinarLayout* mit fester Höhe. Darin befindet sich eine *TextView*, in der die Zeit der Lektion steht. Rechts davon, und immer viermal grösser, befindet sich ein Button. Dieser Button ist zu Beginn noch leer, später werden darauf die Informationen zur Lektion eingefügt.

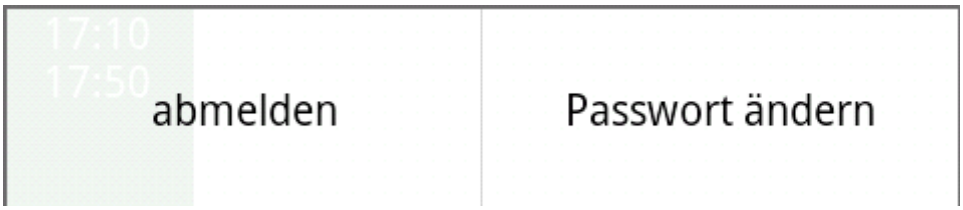
Dies ist die Hauptaufgabe der Activity, sie ruft die Daten von der lokalen Datenbank ab, sucht die passenden Lektionen heraus, und schreibt sie auf das entsprechende Feld.

```
String[] StundenArray = result.split("/");
int i;
for(i=0;i<StundenArray.length;i++){
	String Stunde = StundenArray[i];
	String[] StundeArray = Stunde.split("%");
	if (StundeArray[5].equals("1")){
		if (StundeArray[6].equals("1")){
			final Button feld1 = (Button) findViewById(R.id.monfach1)
			feld1.setText(feld1.getText().toString()+StundeArray[3]+"\n"+St
			undeArray[2]+"\n"+"Zimmer " + StundeArray[4]+"\n");
		}
		else if (StundeArray[6].equals("2")){
			TextView feld2 = (TextView)findViewById(R.id.monfach2);
			feld2.setText(feld2.getText().toString()+StundeArray[3]+"\n"+St
			undeArray[2]+"\n"+"Zimmer " +StundeArray[4]+"\n");
				}
…

	}
}
```

Und so funktioniert es: Das *Requestscript* hat die Daten schon so vorbereitet, dass jede Lektion mit einem "/" getrennt wird und die einzelnen Informationen mit einem "%". Zunächst wird aus dem String ein Array, also eine Liste, erstellt, wobei jede Lektion ein Element davon ist. Diese Elemente werden dann eines nach dem anderen mit einer for-Schleife ausgelesen und daraus ein neues Array gebildet, bestehend aus den einzelnen Informationen der Lektion. Element fünf dieser Liste ist eine Zahl zwischen 1 und 5 und steht für den Tag, an dem die Lektion stattfindet. Wenn also hier bei der MontagActivity keine 1 steht wird einfach zur nächsten Lektion übergegangen. Falls eine 1 steht wird, das sechste Element überprüft, es steht für die Zeit, zu der die Lektion stattfindet. Es wird also überprüft, welche Zahl darin steht und dann auf den richtigen Button geschrieben. Dies wird mit 12 If-Abfragen bewerkstelligt. Ausgegeben wird jedoch nicht einfach das gesamte Array, sondern nur das Element drei, das Fach, das Element zwei, der Name des Lehrers, und das Element vier, die Zimmernummer. Jedes dieser Elemente wird auf eine neue Linie geschrieben, „\n" steht für „Return", und ist gleichbedeutend mit einem Enterschlag. Manchmal kommt es vor, dass zwei Lektionen zur gleichen Zeit stattfinden, wenn sich Halbklassen zum Beispiel abwechseln. Damit in diesem Fall nicht einfach eine Lektion überschrieben wird und verschwindet, wird zuerst mit der Methode *getText()* der bereits auf dem Knopf befindlichen Text abgerufen und nochmals darauf geschrieben.

Die Activity hat noch andere Aufgaben. Zum einem steuert sie alle Knöpfe, dies funktioniert analog zur Activity des Logins. Was jedoch speziell ist, ist die Integration des Menu Buttons,

das ist der Knopf von Android Handys, der sich links des Home Buttons befindet. Wenn man diesen drückt, taucht folgendes am unteren Rand des Bildschirms auf:



Menu-Leiste

Ein kleines Menu mit zwei Knöpfen. Gesteuert werden diese durch folgenden Code.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  MenuInflater inflater = getMenuInflater();
  inflater.inflate(R.menu.menu, menu);
  return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
       case R.id.abmelden:// Die Datenbank wird überschrieben.
              Intent inlo = new Intent(montagActivity.this,Project_timetableActivity.class);
              startActivity(inlo);
              finish();
              break;

       case R.id.passwortandern:
              Intent inpw = new Intent(montagActivity.this,changepwActivity.class);
              startActivity(inpw);
              finish();
   }
  return true;
}
```

Der erste Abschnitt erstellt das Menu, sobald der Menu-Button gedrückt wird. Dabei wird auf die XML-Datei „menu.xml“ verwiesen. Diese ist sehr klein und definiert eigentlich nur die Namen der Knöpfe. Der zweite Abschnitt wird ausgeführt, sobald einer der beiden Knöpfe gedrückt wird. Falls der „abmelden“-Knopf gedrückt wird, werden jegliche Daten auf der Datenbank überschrieben. Aus Platzgründen wurde dieser Teil bei der Arbeit entfernt und durch einen Kommentar ersetzt. Nachdem die Datenbank überschrieben ist, wird mit einem sogenannten *Intent* die *Project_timetabelActivity* aufgerufen. Falls der „Passwort ändern“-Knopf gedrückt wird, wird mit einem anderen *Intent* die *ChangepwActivity* aufgerufen, welche im nächsten Kapitel behandelt wird.

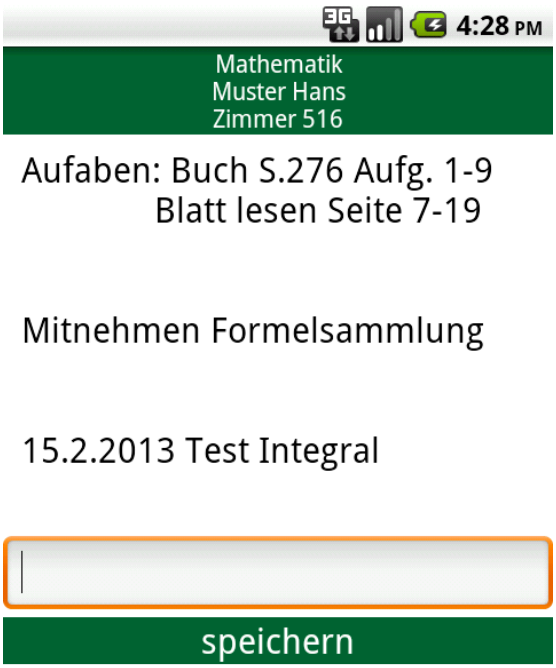
4.4.3.3. ChangePW

Die *ChangepwActivity* dient dazu, das Passwort zu ändern. Sie bringt an sich nichts Neues und somit gehe ich nicht genauer auf den Quellcode ein. Die Oberfläche ist stark an diejenige des Logins angelehnt und braucht die gleichen Views. Sie besitzt drei *TextEdit* für das alte Passwort, das neue Passwort und die Wiederholung des Passworts, sowie ein Knopf für die Bestätigung Eingabe. Auch die Activity ähnelt sehr stark der *Project_timetableActivity*. Sobald der Knopf gedrückt wird, wird der Inhalt der *TextEdit* ausgelesen und durch an eine *httpconnect*-Klasse geschickt. Dort werden das alte und neue Passwort, sowie der Benutzername, welche von der lokalen Datenbank abgerufen wird, an ein Script namens C*hangepwscript* gesendet. Bei diesem Script wird eine Abfrage auf die MySQL-Datenbank getätigt und überprüft, ob das Passwort richtig ist. Falls dieses stimmt, wird das neue Passwort in die Datenbank geschrieben.

4.4.3.4. Notizen

Die letzte integrierte Funktion ist die Notizfunktion. Sie ist relativ komplex und wird dynamisch erstellt. Für diese Funktion hatte ich nicht viel Zeit und die Chancen stehen gut, dass ich sie noch überarbeiten werde. Zum Zeitpunkt des Schreibens dieser Arbeit funktionierte sie folgendermassen.

Es ist vielleicht aufgefallen, dass bei der *TagActivity* jede Lektion nicht auf ein *TextView* sondern auf einen *Button* geschrieben wird. Der Grund liegt darin, dass mit dem *Button* die *FachActivity* gestartet wird. Dabei werden ausserdem die Informationen der Lektion weitergeleitet. Die *FachActivity* erstellt dann mit diesen Informationen folgende Oberfläche.



fach.xml; Die Notizen der Mathematik-Lektion

Mit der *FachActivity* kann man nun irgendwelche Notizen zu dieser Lektion eingeben. Diese werden auf einer Tabelle der lokalen Datenbank abgespeichert. Das Schwierige daran ist nicht das Eingeben und Abspeichern der Notizen, sondern das Darstellen und vor allem das Löschen. Denn diese Notizen müssen auf dynamisch erstellten Knöpfen angezeigt werden. Dynamisch, weil man von vorneherein nicht weiss, wie viele Notizen angezeigt werden müssen, also müssen sie fortlaufend von der Activity erstellt werden. Knöpfe müssen es sein, da sie wieder gelöscht werden können, sobald man auf die klickt. Das dynamische Erstellen von Köpfen ist etwas mühsam, jedoch gut zu bewerkstelligen. Das Problem war das Verknüpfen der erstellten Buttons mit ihrer Funktion, also das Löschen von sich selbst. Gelöst habe ich dieses Problem folgendermassen:

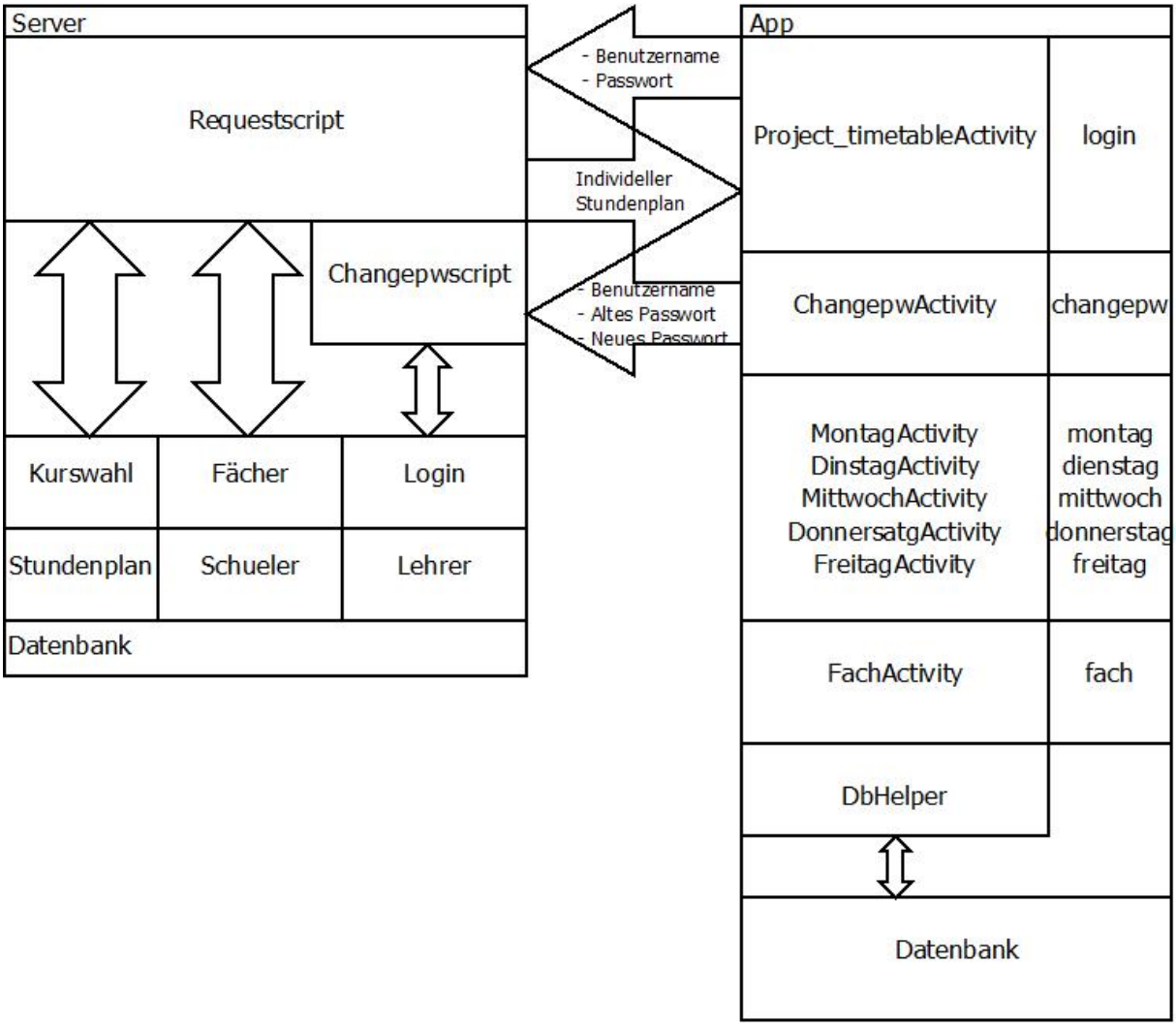
```
final String[] NotizArray = result.split("%%");
int i;
for(i=0;i<NotizArray.length;i++){
      if(NotizArray[i].split("//")[0].equals(lektion)){
            final int ifinal = i;
            View linearLayout =  findViewById(R.id.notizen);
            final Button notizview = new Button(this);
            notizview.setId(i);
            notizview.setTextSize(20);
            notizview.setBackgroundColor(0);
            notizview.setGravity(3);
            notizview.setText(NotizArray[i].split("//")[1] + "\n");
            notizview.setLayoutParams(new
            LayoutParams(LayoutParams.FILL_PARENT,LayoutParams.WRAP_CONTENT));
            notizview.setOnClickListener(new OnClickListener() {

                  public void onClick(View v) {
                      int j=0;
                      String notizenstring = "";
                      for(j=0;j<NotizArray.length;j++){
                            if(ifinal != j){
                               notizenstring = notizenstring +"%%"+NotizArray[j];
                               notizenstring = notizenstring.replace("%%%%","");
                            }

                      }
// Überschreiben des Notizstrings auf der Datenbank und Neustart der Activity
                  }
             });
           ((LinearLayout) linearLayout).addView(notizview);
      }
}
```

Dieser Code wird ausgeführt, wenn die Activity gestartet wird. Vorher wird noch von der Datenbank ein String abgefragt, der alle Notizen beinhaltet, er wird hier „result“ genannt. Dieser wird in ein Array aufgeteilt und ähnlich wie bei den *TagActivities* wird überprüft, welche Notiz auch wirklich zu diesem Fach gehört. Falls sie in dieses Fach gehört, wird ein Button mit dem Text erstellt. Nun kommt der Trick: der *OnClickListener()* wird in der for-Schleife benutzt. Das heisst, es wird zu jedem erstellten Knopf eine neue Instanz des *OnClickListners()* erstellt. Wenn dann der Knopf gedrückt wird, wird ein neuer String erstellt, der alle Notizen enthält, die sich im Array befinden, ausser der Notiz, die genau auf diesem Button steht, und der alte String auf der Datenbank wird überschrieben. Zuletzt wird die Activity neu gestartet.

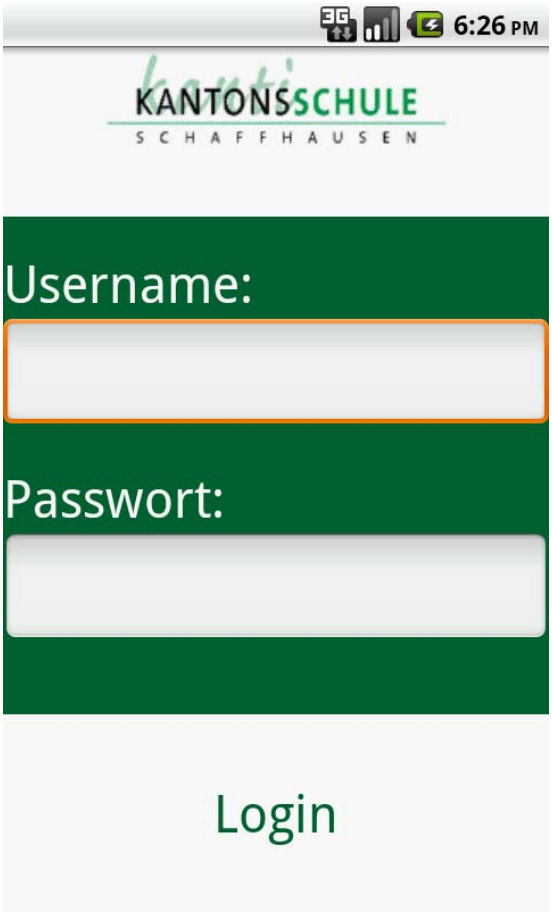
Das Speichern von Daten geschieht analog, es wird ein neuer String erstellt, der die neue Notiz enthält, der alte wird damit überschrieben, und die Activity wird neu gestartet.

5. Zusammenfassung

Nach dem teilweise sehr tiefen Einblick in den Aufbau des Projekts, besteht die Gefahr den Überblick zu verlieren. Deswegen führe ich die Fäden zum Schluss noch zusammen: Die App wird aus der Sicht eines Benutzers betrachtet, und dabei auf die im Hintergrund laufenden Prozesse verwiesen, die in dieser Arbeit erklärt wurden. Die untenstehende Grafik soll als Orientierung dienen und helfen sich an den erwähnten Prozess zu erinnern.



Sobald der Benutzer die App zum ersten Mal öffnet, sieht er einen Login-Screen, welcher ihn auffordert, seinen Benutzernamen sowie sein Passwort anzugeben. Nach der Eingabe wird er auf die Übersicht seines Stundenplans weitergeleitet.

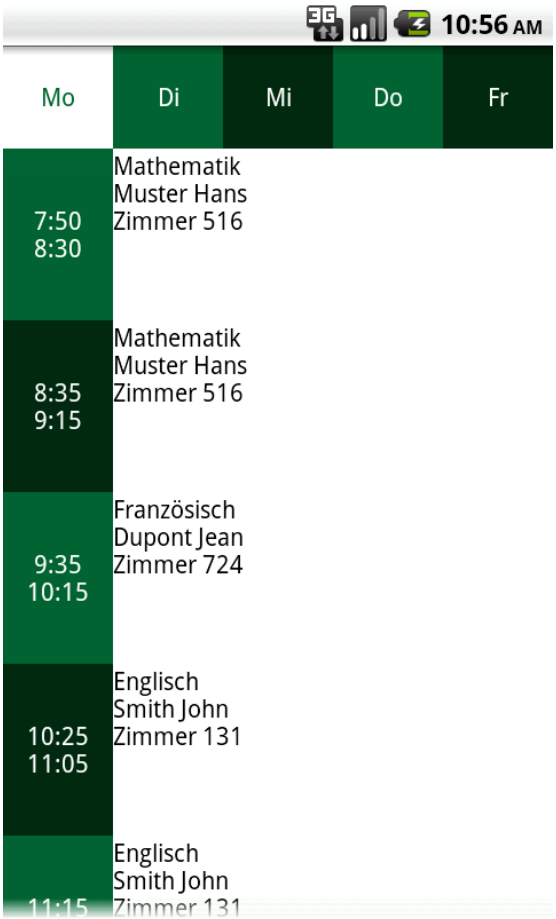


Login.xml; Login-Screen

Hinter den Kulissen geschieht jedoch mehr. Die Daten müssen grundsätzlich in lesbarer Form auf der Datenbank verfügbar sein. Beim Start des Apps wird die *Project_timetableActivity* aufgerufen, welche prüft, ob sich der Benutzer bereits schon einmal angemeldet hat. Dafür verwendet sie die Klasse *DbHelper*, welche als Schnittstelle zur Datenbank fungiert. Diese merkt, dass sie noch nie aufgerufen wurde, und erstellt eine Datenbank mit den benötigten Tabellen. Dann meldet sie der *Project_timetableActivity*, dass die Datenbank leer ist und sich der Benutzer noch nie angemeldet hat. Diese ruft somit die *login.xml* Datei auf und wartet auf den Input des Benutzers. Sobald dieser seinen Benutzernamen und Passwort angegeben hat, werden diese Daten an das *Requestscript* auf dem Server weitergegeben. Das Requestscript ruft mit Hilfe dieser Daten den Stundenplan

des Benutzers ab und schickt ihn zurück. Die *Project_timetableActivity* empfängt den Stundenplan und schreibt ihn in die lokale Datenbank. Zuletzt wird die Activity des aktuellen Tages aufgerufen.

Der Benutzer sieht nun seinen Stundenplan vor sich. Oben sieht er eine Leiste, mit deren Hilfe er zwischen den Wochentagen wechseln kann. Wenn er auf eine der Lektionen drückt, erscheint eine neue Seite. Dort kann er Notizen aufschreiben und wieder löschen, indem er auf sie klickt. Wenn er den Menu-Knopf drückt, kann er sich entweder abmelden, also er kommt wieder auf den Login-Screen, oder sein Passwort ändern. In letzterem Falle kommt er auf eine neue Seite, auf der er sein altes sowie neues Passwort eingeben muss.



montag.xml; Der Stundenplan Montag

Dass der Benutzer seinen Stundenplan sehen kann, ist das Werk der *TagActivities*, also *MontagActivity* bis *FreitagActivity*. Diese rufen die Daten des Stundenplans mit Hilfe der Klasse *DbHelper* von der lokalen Datenbank ab, und stellt diese auf Knöpfen der montag.xml

bis freitag.xml Datei dar. Wenn der Benutzer auf einen dieser Knöpfe drückt, wird die *FachActivity* aufgerufen und ihr die sich auf dem Knopf befindenden Daten zugeschickt. Die *FachActivity* ermöglicht Notizen, wieder über die Klasse *DbHelper*, in der lokalen Datenbank zu speichern. Wenn der Benutzer auf der *TagActivity* den Menu-Knopf drückt, öffnet sich ein kleines Menu mit zwei Knöpfen. Wenn der „abmelden“-Knopf gedrückt wird, werden alle Daten auf der lokalen Datenbank überschrieben und die *Project_timetableActivity* aufgerufen. Falls der „Passwort ändern“-Knopf gedrückt wird, wird die *ChangepwActivity* gestartet. Diese Activity fordert das alte sowie ein neues Passwort vom Benutzer und schickt diese Daten an das *Changepwscript* auf dem Server. Dieses überprüft das alte Passwort und überschreibt es mit dem neuen.

6. Ausblick

Ich werde auf jeden Fall die App in den nächsten Monaten auf den Android-Market bringen und das Problem der Passwort Verbreitung mit Hilfe eines Mail-Servers lösen. Ich hoffe, es gibt auch eine gewisse Nachfrage nach dem Programm. Falls dies so ist, werde ich auch in Zukunft noch an der App tüfteln und sie verbessern.

7. Quellenverzeichnis

7.1. Literaturquellen

- Felker, Donn(2011). *Android Apps Entwicklung für Dummies*. Aus dem Englischen von Gerhart Franken. WILLEY-VCH Verlag GmbH & Co. KGaA, Weinheim.
- Barry, Paul(2010). *Python von Kopf bis Fuss*. Aus dem Englischen von Lars Schulten. O'Reilly Verlag GmbH & Co. KG, Köln.

7.2. Internetquellen

- http://www.oracle.com/technetwork/java/index.html [Stand: 01.12.2012]
- http://de.wikipedia.org/wiki/Java_%28Programmiersprache%29 [Stand: 01.12.2012]
- http://www.w3.org/TR/2006/REC-xml-20060816/#sec-origin-goals [Stand: 01.12.2012]
- http://de.wikipedia.org/wiki/Extensible_Markup_Language [Stand: 01.12.2012]
- http://dev.mysql.com/ [Stand: 01.12.2012]
- http://de.wikipedia.org/wiki/SQL [Stand: 01.12.2012]
- http://de.wikipedia.org/wiki/Zeichenkodierung [Stand: 01.12.2012]
- http://developer.android.com/about/index.html [Stand: 01.12.2012]
- http://de.wikipedia.org/wiki/Android_%28Betriebssystem%29 [Stand: 01.12.2012]
- http://www.ubuntu.com [Stand: 01.12.2012]
- http://de.wikipedia.org/wiki/Ubuntu [Stand: 01.12.2012]
- http://wiki.ubuntuusers.de/Server_Installation [Stand: 01.12.2012]
- http://www.linuxquestions.org/questions/ubuntu-63/lucid-server-install-issue-during-install-the-base-system-846489/ [Stand: 01.12.2012]
- http://www.bytefresser.de/bitsnbytes/Hardware/Networking/PortForwarding+auf+einer+FritzBox+einrichten_438.html [Stand: 01.12.2012]
- http://dyn.com [01.12.2012]
- http://www.havetheknowhow.com [Stand: 01.12.2012]
- http://www.mysqltutorial.org/ [Stand: 01.12.2012]
- http://zetcode.com/databases/mysqlpythontutorial/ [Stand: 01.12.2012]
- http://stackoverflow.com/questions/5231575/android-httppost-not-working-with-python-cgi-server [Stand: 01.12.2012]
- http://wiki.python.de/Von%20Umlauten,%20Unicode%20und%20Encodings [Stand: 01.12.2012]
- http://mobileorchard.com/category/tutorials/android/ [Stand: 01.12.2012]
- http://www.daniweb.com/software-development/python/threads/318059/how-to-change-text-file-encoding-with-python [Stand: 01.12.2012]
- https://sites.google.com/site/kentapps/home/app_dev/multithreading [Stand: 01.12.2012]
- http://www.vogella.com/articles/AndroidSQLite/article.html#sqliteoverview_sqliteopenhelper [Stand: 01.12.2012]