Entwicklung einer Organizer-App für Android mithilfe von Android Studio

Maturaarbeit von Bogdan Gadzhylov
im Fach Informatik
Betreut von Raphael Riederer

Kantonsschule Schaffhausen
2016

Inhaltsverzeichnis

1. Vorwort

1.1 Überblick

Im Rahmen meiner Maturaarbeit habe ich eine Android-App mithilfe von Android Studio entwickelt. Im schriftlichen Teil meiner Arbeit versuche ich den Entwicklungsprozess Schritt für Schritt darzustellen und zu erklären, damit auch Leute ohne grosses Fachwissen einen guten Überblick über die Entwicklung einer Android-App bekommen. Ich werde aber nur die wichtigsten Schritte erklären, ohne zu sehr ins Detail zu gehen, da ich in meinem gesamten Projekt mehr als 10 000 Zeilen Quellcode habe und es mir somit unmöglich wäre, im Rahmen des schriftlichen Teils meiner Arbeit alles zu erläutern.

1.2 Motivation

Die Idee, eine Android-App zu entwickeln, hatte ich bereits vor dem Beginn der Maturaarbeit gehabt, jedoch hatte ich damals noch nicht an die Realisierung des Projektes gedacht. In der Maturaarbeit habe ich eine Möglichkeit gesehen, diese Idee in die Praxis umzusetzen. Die Tatsache, dass ich gerne programmiere und mich sehr für die moderne Technologie und Elektronik interessiere, hat mich schlussendlich dazu gebracht, dieses Gebiet zu thematisieren. Ausserdem habe ich in der selbständigen Entwicklung einer App eine Herausforderung gesehen, da ich vorher noch gar keine Erfahrungen in diesem Gebiet gehabt hatte und hatte selber alle Grundlagen von Java, XML, SQLite und Android Studio erlernen müssen.

1.3 Warum Android?

Ich habe mich aus folgenden Gründen entschieden, in Android zu programmieren:
1. Ich habe selber ein Smartphone, welches Android als Betriebssystem hat.
2. Die meisten Smartphones haben Android als Betriebssystem (z.B. Sony, Samsung, HTC). [1]
3. Die Entwicklungssoftware für Android ist komplett kostenfrei.
4. Mit Android kann man nicht nur für Smartphones sondern auch für Smart-TVs und Smartwatches Programme entwickeln.

[1] Stand 2014, Quelle: https://de.wikipedia.org/wiki/Android_(Betriebssystem)

2. Funktionalität der App

Nachdem ich mir darüber Gedanken gemacht habe, welche Funktion diese App haben soll, habe ich entschieden, dass es nicht einfach unterhalten, sondern auch im Alltag nützlich sein soll. So bin ich auf die Idee gekommen, ein Organizer zu programmieren, der Schülern und Lernenden für folgende Punkte dienlich sein soll: Diese App kann sie an ihre Termine und bevorstehende Prüfungen erinnern, ihnen automatisch die Durchschnittsnote in einem Fach berechnen, sowie den Promotionsstatus anzeigen. Ausserdem hat es eine Agenda, in die man den eigenen Stundenplan eintragen und die Hausaufgaben aufschreiben kann. Schliesslich hat man noch ein Notizbuch, in das man seine Gedanken zu einem bestimmten Thema aufschreiben kann und einen Einkaufszettel, der beim Einkaufen hilft.

Bild 1: Startbildschirm der App

2.1 Kalender

In diesem Teil der App können die Benutzer ihre Termine verwalten. Das heutige Datum erscheint hellgrün und die Daten, an denen ein Termin stattfindet dunkelgrün. Durch langes Klicken auf ein Datum erscheint ein Fenster, welches dem Benutzer erlaubt, einen neuen Termin an diesem Datum hinzuzufügen. Wenn ein neuer Termin hinzugefügt wird, wird dem Nutzer jeweils am vorherigen Tag um 11:00 Uhr eine Benachrichtigung angezeigt, welche ihn an den Termin erinnert.
Durch das Klicken auf die grünen Pfeile gelangt man jeweils zum vorherigen oder zum nächsten Monat. Unterhalb des Kalenders werden sämtliche Termine aufgelistet und nummeriert. Falls man einen Termin löschen will, schreibt man die Nummer des jeweiligen Termins hin und drückt auf „Event löschen“. Falls man alle vorhandenen Termine löschen will, kann man das durch Drücken auf den Button „Datenbank löschen“ erreichen.

Bild 2: Kalender

2.2 Notizbuch

Hier steht dem Benutzer ein Notizbuch zur Verfügung. Darin kann er seine Gedanken oder Ideen zu einem bestimmten Thema aufschreiben. Somit sind die Notizen immer nach Themen sortiert. In dieser App gibt es standardmässig vier Themen, zu denen man seine Gedanken aufschreiben kann: Arbeit, Familie, Freizeit und Sport. Wenn man aber zu einem anderen Thema Notizen machen will, kann man auf das rote Pluszeichen klicken und ein neues Thema hinzufügen, oder ein bereits vorhandenes Thema löschen.

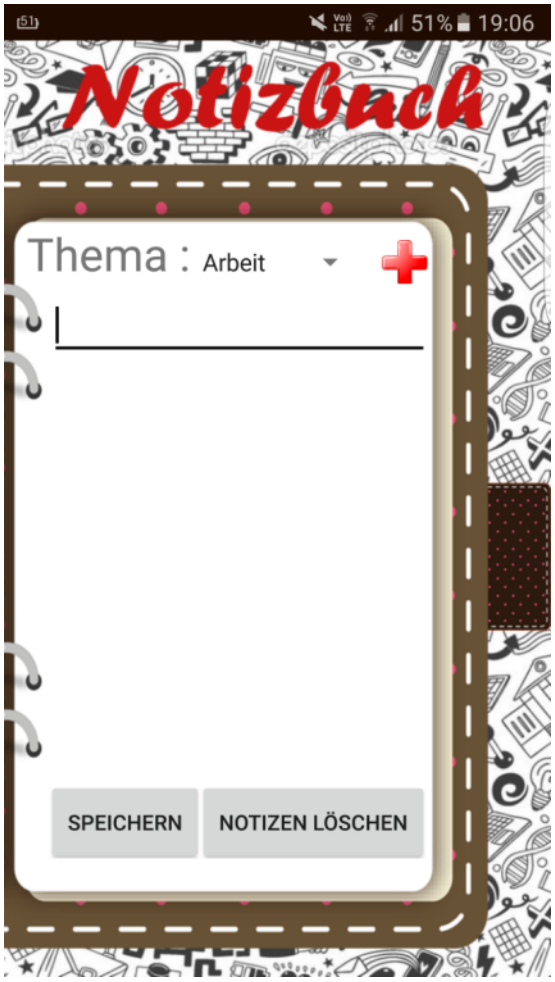


Bild 3: Notizbuch

Bild 4: Neues Thema hinzufügen

2.3 Schule

Der Bereich „Schule“ hilft Schülern und Lernenden, ihre Noten, Prüfungen und Hausaufgaben zu koordinieren und zu verwalten. Somit hat dieser Bereich die Funktion einer Agenda. In diesem Bereich der App gibt es drei Unterbereiche: Noten, Prüfungen und Aufgaben.

Bild 5: Schule

2.3.1 Noten

Hier kann der Nutzer seine Noten in den verschiedenen Fächern eintragen. Die Durchschnittsnote in diesem Fach wird automatisch berechnet und auf zwei Nachkommastellen gerundet. Selbstverständlich kann man auch die Fächernamen ändern, oder in einer anderen Reihenfolge aufschreiben. Insgesamt hat es Platz für 16 Fächer. Für jedes Fach haben sieben Noten Platz. Wenn man auf den gelben Pfeil drückt, gelangt man zu den Informationen über die Promotion. Dort wird angezeigt, wie viele Pluspunkte man erzielt und welche Durchschnittsnote von allen Fächern man erreicht hat.

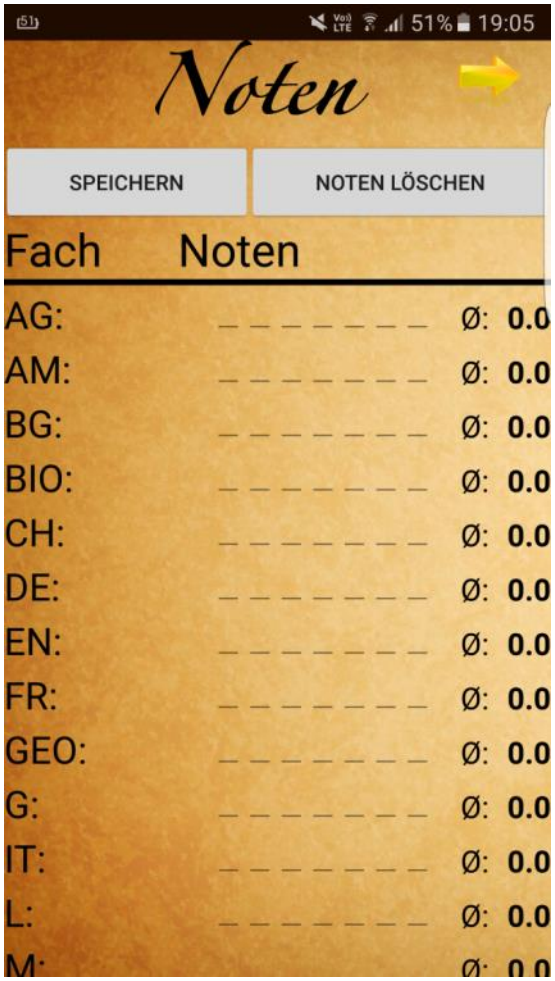


Bild 6: Noten

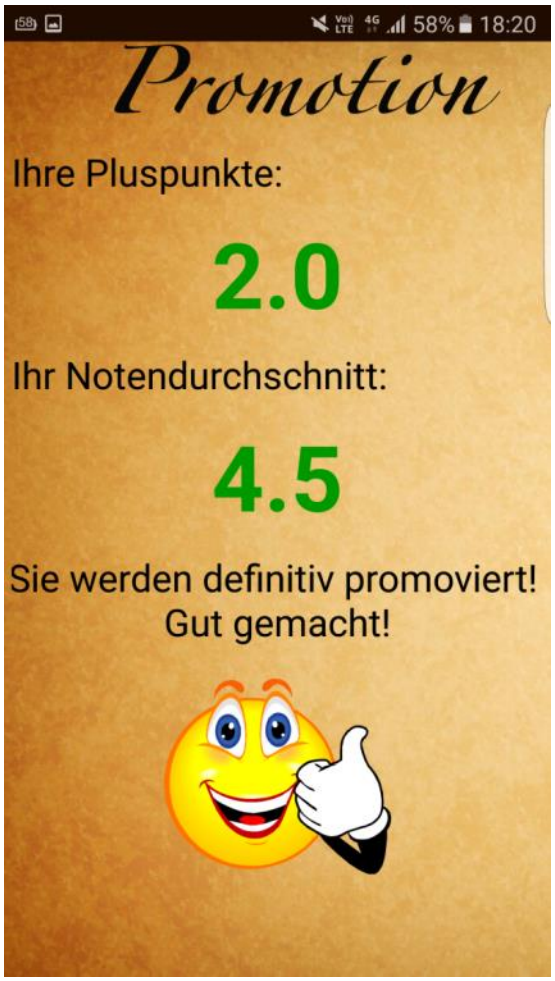


Bild 7:Promotion

2.3.2 Prüfungen

In diesem Bereich steht dem Nutzer ein ähnlicher Kalender zur Verfügung wie in „2.1 Kalender“ schon beschrieben wurde. Dieser Kalender ist aber nur für Prüfungen gedacht. Somit sind die Prüfungstermine von den alltäglichen Terminen getrennt und man hat eine bessere Übersicht. Hier wird der Nutzer ebenfalls durch eine Benachrichtigung jeweils am vorherigen Tag um 11:00 Uhr an einen Prüfungstermin erinnert.

Bild 8: Prüfungen

2.3.3 Stundenplan/Hausaufgaben

Hier kann der Nutzer seinen eigenen Stundenplan für jeden Wochentag eintragen. Rechts davon kann man die Hausaufgaben für die jeweilige Lektion eintragen. Wenn man auf die blauen Pfeile drückt, gelangt man jeweils zum vorherigen oder zum nächsten Tag. Mit der Taste „Aufgaben löschen“ werden alle Aufgaben an diesem Tag gelöscht.

Bild 9: Aufgaben

2.4 Einkaufszettel

Hier kann der Nutzer einen Einkaufszettel erstellen, indem er die Menge und das Produkt angibt und auf das Pluszeichen drückt. Bei der Mengenangabe kann man die Einheit auswählen: x (Stück), kg (Kilogramm), g (Gramm) oder l (Liter). Nachdem das Produkt hinzugefügt worden ist, erscheint es auf dem Einkaufszettel in Form einer Liste. Die bereits gekauften Artikel kann der Nutzer anklicken und diese somit durchstreichen, um eine bessere Übersicht während des Einkaufens zu bekommen. Ausserdem sind die Produkte nach Kategorien sortiert. Als Standard gibt es vier Kategorien: Apotheke, Bäckerei, Metzgerei und Supermarkt. Falls man eine andere Kategorie hinzufügen oder löschen möchte, kann man das in ähnlicher Weise wie beim Notizbuch tun. Wenn man auf die rote Taste „Löschen“ drückt, werden alle Artikel in dieser Kategorie gelöscht. Falls man aber nur einen Artikel löschen will, erreicht man das durch langes Klicken auf den jeweiligen Artikel.

Bild 10: Einkaufszettel

Bild 11: Kategorie hinzufügen

3. Theorie

3.1 Sprachen

Die wichtigsten Programmiersprachen, die man kennen muss, um eine Android-App zu entwickeln sind Java, XML und SQLite.

3.1.1 Java

Mithilfe von Java wird die gesamte Funktionalität der App programmiert. Alles was die App tun kann, ist mit Java programmiert worden.
Java ist eine vergleichsweise einfach zu erlernende Programmiersprache, die an Popularität immer mehr zunimmt. Dadurch, dass in Java auf die Zeigerarithmetik, die C/C++-Programmierer häufig zur Verzweiflung treibt, verzichtet wurde, entfallen komplizierte und fehleranfällige Sprachkonstrukte. Diese Vereinfachung trägt auch enorm zur Stabilität von Programmen bei.
In Java ist es vorgesehen, dass man objektorientiert programmiert. Dies fällt allein schon dadurch auf, dass keine Methoden ausserhalb von Klassen programmiert werden können. Java bietet eine sehr hohe Sicherheit, da ungewollter Zugriff auf Arbeitsspeicherbereiche und dessen Manipulation durch den Interpreter (Java Virtual Machine, kurz JVM) unterbunden wird.
Der grösste Vorteil ist allerdings, dass ein Java-Programm unter jedem Betriebssystem gleich abläuft. Der Java-Quellcode wird in einen Bytecode umgewandelt, der nur von der JVM interpretiert wird. Daher muss einzig die JVM auf das jeweilige Betriebssystem angepasst sein.
Das bedeutet, dass Programme, die mit Java programmiert wurden, nicht nur auf einem Computer oder Smartphone laufen können, sondern auch auf einer Smartwatch, einem Fernseher, in einem Auto oder sogar in einem Toaster in der Küche.
Zwar hatte ich bereits Erfahrungen in Javascript und Python, dafür aber noch gar keine Erfahrungen in Java gehabt und habe die Grundlagen selber lernen müssen. Ich habe Java mithilfe von verschiedenen YouTube-Videos, dem Openbook „Java ist auch eine Insel“ von Christian Ullenboom und der Lern-App „Sololearn“ erlernt.
Code-Beispiel:

```
System.out.println(„Hello World“);
```

3.1.2 XML

Die Extensible Markup Language (engl. „erweiterbare Auszeichnungssprache“), abgekürzt XML, ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien.
Mithilfe von XML wird das Aussehen der App programmiert. Man definiert beispielsweise, wo ein Button, also eine Taste, platziert wird, wie gross der Text in einem Textfeld ist, was sich im Hintergrund befindet und alles was mit dem Aussehen der App verbunden ist.
Ich habe XML durch YouTube-Videos gelernt. Ich habe mir verschiedene Tutorials angesehen und habe das Gelernte selber in Android Studio ausprobiert.
Da ich in der Schule den Web-Informatikkurs besucht habe, habe ich einiges Wissen über HTML und Javascript gesammelt und habe festgestellt, dass die Zusammenhänge zwischen HTML und Javascript sehr ähnlich sind zu den Zusammenhängen zwischen XML und Java. HTML und XML definieren das Aussehen, das Design. Java und Javascript definieren die Funktionalität und die Wechselbeziehungen zwischen einzelnen Elementen.
Zur besseren Übersicht zeige ich hier ein einfaches Beispiel. Dieser XML-Code definiert folgendes Design auf dem Bildschirm.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="TEXTFELD 1"
    android:textStyle="bold"
    android:textSize="20sp"/>

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="BUTTON"
    android:layout_marginTop="20dp" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="TEXTFELD 2"
    android:textStyle="bold"
    android:textSize="20sp"
    android:layout_marginTop="20dp"/>
```

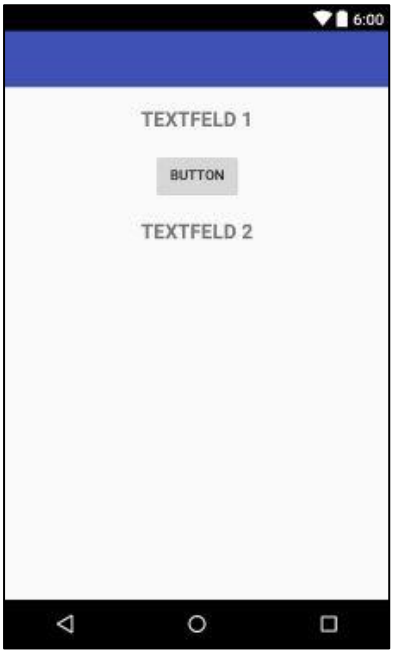


Bild 12: XML Beispiel

3.1.3 SQLite

SQLite ist eine Programmbibliothek, die ein relationales Datenbanksystem enthält.
Die SQLite-Bibliothek lässt sich direkt in entsprechende Anwendungen integrieren, sodass keine weitere Server-Software benötigt wird. Dies ist der entscheidende Unterschied zu anderen Datenbanksystemen. Durch das Einbinden der Bibliothek wird die Anwendung um Datenbankfunktionen erweitert, ohne auf externe Softwarepakete angewiesen zu sein.
Google nutzt für seinen Browser Chrome SQLite, um unter anderem Benutzerdaten lokal zu sichern. SQLite ist das am meisten verbreitete und meistverwendete Datenbanksystem der Welt.
Mithilfe von SQLite werden die Daten der App gespeichert und aufbewahrt. Ohne SQLite würden alle Daten nach jedem Schliessen der App verloren gehen, deshalb habe ich fast in jedem Teil meiner App SQLite verwendet. Zum Beispiel, um die eingetragenen Noten, Termine, Hausaufgaben und Notizen zu speichern.
Code-Beispiel:

```
CREATE TABLE employees
( employee_id INTEGER PRIMARY KEY AUTOINCREMENT,
  last_name VARCHAR NOT NULL,
  first_name VARCHAR,
  hire_date DATE
);
```

Hier wird eine SQLite-Tabelle erstellt mit vier Spalten:
1) employee_id vom Typ INTEGER, also ganze Zahl und als Primärschlüssel
2) last_name vom Typ VARCHAR, also Text
3) first_name auch vom Typ VARCHAR
4) hire_date vom Typ DATE, also Datumsformat

3.2 Entwicklungsumgebung

Eine integrierte Entwicklungsumgebung (Abkürzung IDE, von englisch integrated development environment, auch - als Teilaspekte - integrated design environment oder integrated debugging environment) ist eine Sammlung von Anwendungsprogrammen, mit denen die Aufgaben der Softwareentwicklung (SWE) möglichst ohne Medienbrüche bearbeitet werden können.

3.2.1 Android Studio

Android Studio ist eine freie, integrierte Entwicklungsumgebung (IDE) von Google und offizielle Entwicklungsumgebung für Android. Android Studio basiert dabei auf IntelliJ IDEA.
Ich habe mich aus folgenden Gründen entschieden, in Android Studio zu programmieren:

1. Android Studio ist komplett kostenfrei.
2. Live Preview Funktion: Beim Verändern des Designs in XML sieht man sofort die grafischen Veränderungen in einem kleinen Fenster daneben. Somit muss man nicht wegen jeder Kleinigkeit die gesamte App neu installieren, um das Design auf dem Smartphone zu überprüfen.

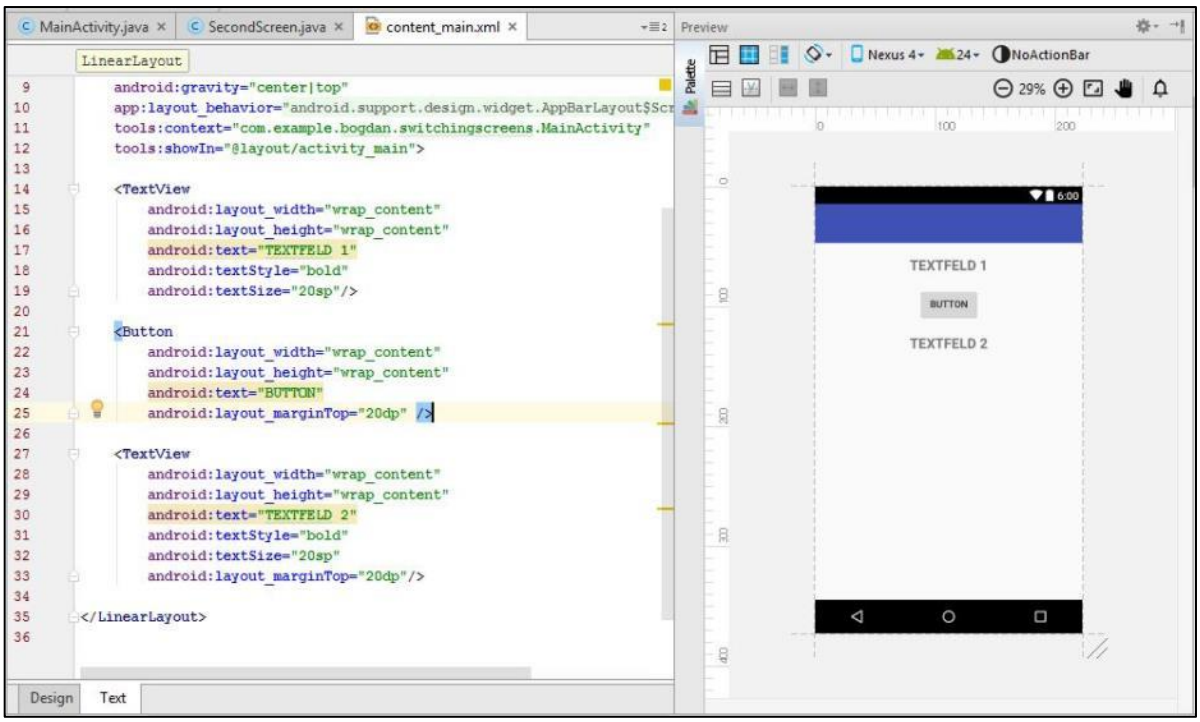


Bild 13: Live Preview

3. Code Completion: Android Studio gibt beim Eintippen des Codes immer Vorschläge, indem es die zur jeweiligen Situation passenden Funktionen und Variablen anzeigt. Somit wird die Schreibarbeit immens verkleinert, die Effizienz gesteigert und die Wahrscheinlichkeit sich zu vertippen sehr klein gehalten. Bei den Funktionen wird auch angezeigt, welche Parameter die Funktion benötigt.

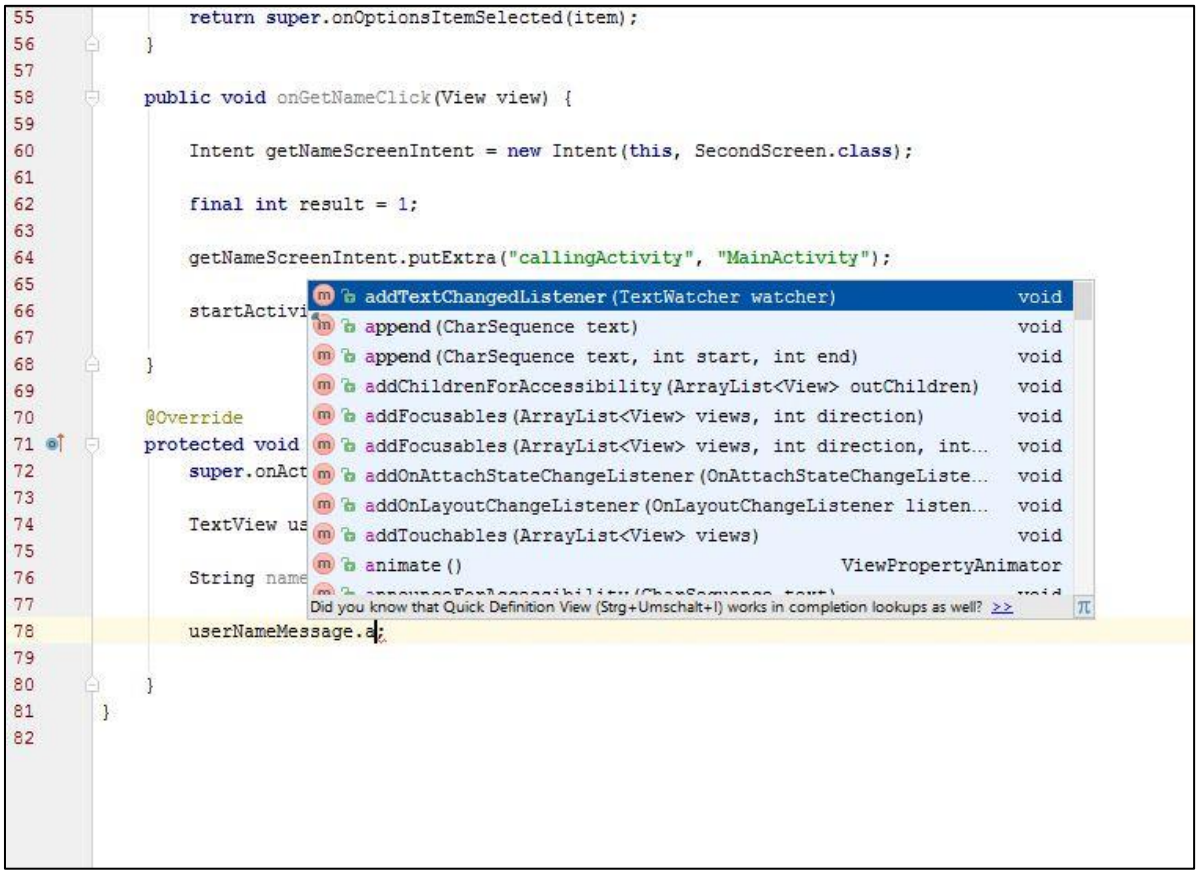


Bild 14: Code Completion

4. Android Studio hat einen eingebauten Emulator, auf dem man seine App testen kann. Man kann verschiedene Smartphones emulieren mit verschiedenen Bildschirmauflösungen, Android-Versionen und Speicherkapazitäten. Man kann sogar Smart-TVs, Smartwatches und Tablets emulieren. Zwar ist dieser Emulator nicht besonders schnell und es wird empfohlen, ein echtes Android-Gerät anzuschliessen, um die App zu testen, und dennoch kann diese Funktion sehr hilfreich sein, besonders wenn man kein Android-Gerät zur Verfügung hat.

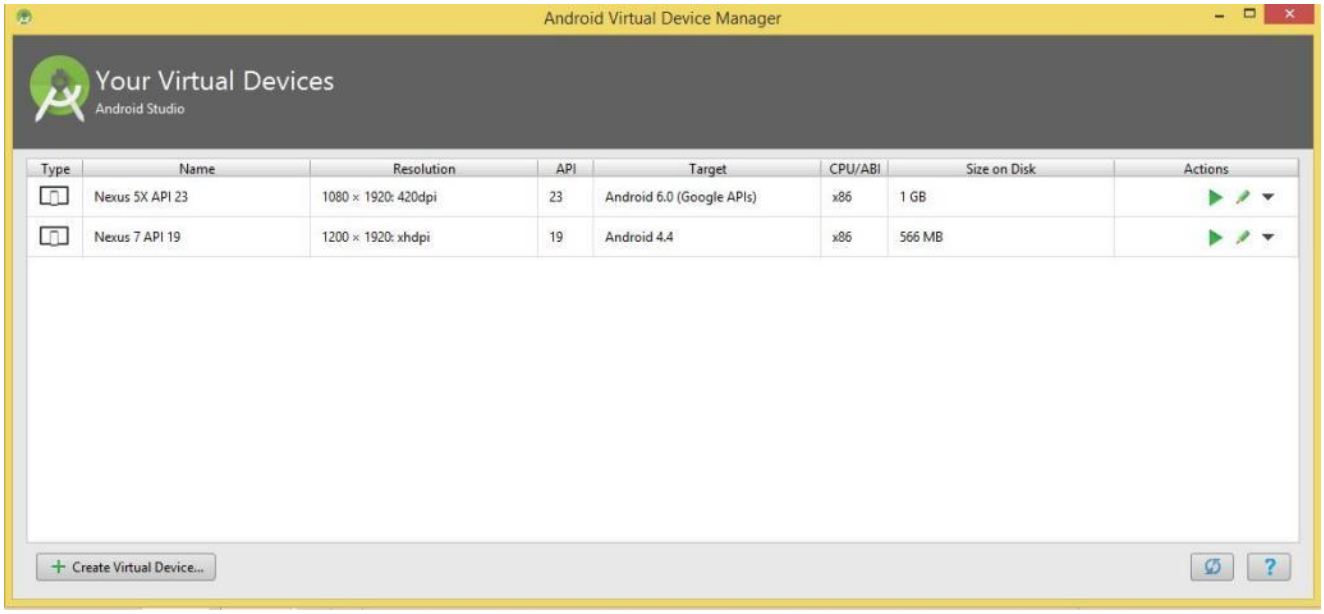


Bild 15: Android Virtual Device Manager (AVD). Hier kann man das gewünschte Gerät auswählen.

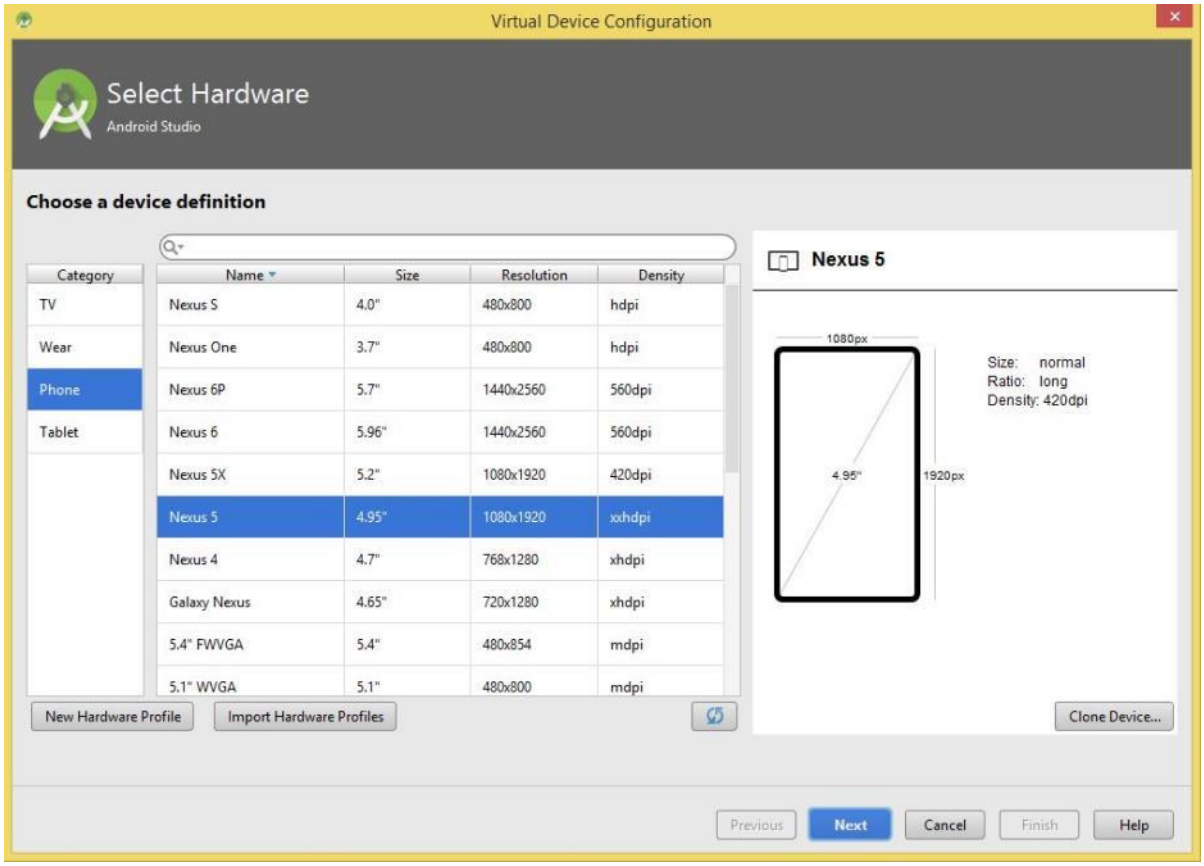


Bild 16: Fenster, um ein neues Gerät zu emulieren. Hier kann man die gewünschte Hardware auswählen. welche dann emuliert wird.

Bild 17: Auf dem ausgewählten Gerät kann man die App wie auf einem echten Smartphone testen.

Android Studio Übersicht

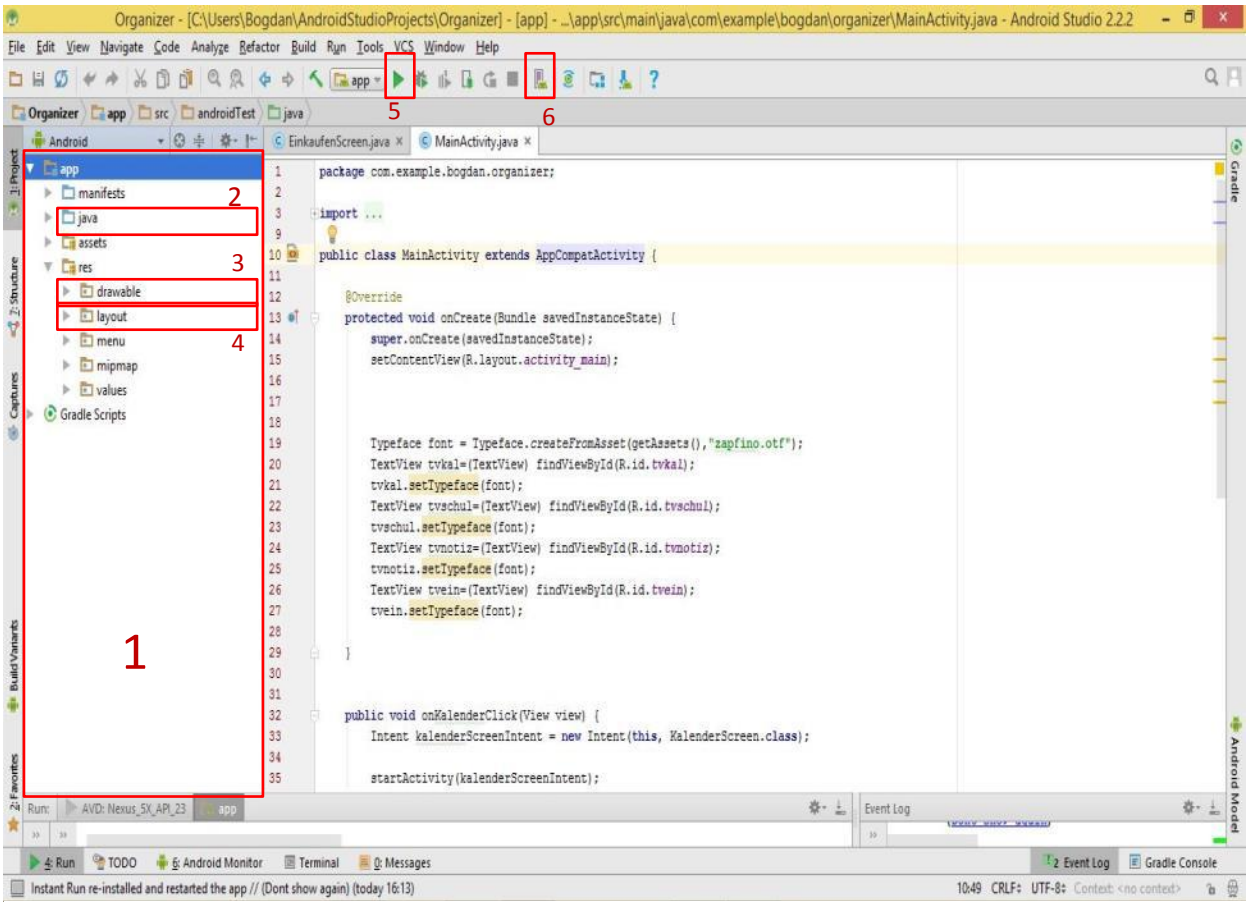


Bild 18: Android Studio

1. Das ist die gesamte Übersicht der App
2. Hier sind alle Java-Dateien dieser App aufgeführt
3. In „drawable“ befinden sich alle Bilder, die man in der App benutzt
4. In „layout“ befinden sich alle XML-Dateien der App
5. Button, um die App auf einem Emulator oder Smartphone zu starten
6. Das ist der Android Virtual Device Manager, den man braucht, um ein virtuelles Gerät zu erstellen

4. Entwicklungsprozess

In diesem Teil meiner schriftlichen Arbeit versuche ich den Entwicklungsprozess möglichst einfach und verständlich darzustellen und die Schwierigkeiten, Probleme und Fehler, welche während der Entwicklung aufgetreten sind, zu erläutern. Den Entwicklungsprozess teile ich genauso auf, wie auch die App selber aufgebaut ist, in die vier Bereiche: Kalender, Agenda, Notizbuch und Einkaufszettel.

4.1 Kalender

Da ich schon vor dem Programmieren vorausgesehen habe, dass die Kalenderfunktion am anspruchsvollsten sein wird, habe ich mich mit dieser als Erstes befasst.
Ich habe die Idee gehabt, dem Nutzer einen Kalender anzuzeigen, welcher die bevorstehenden Termine anzeigt und gleichzeitig erlaubt, einen neuen Termin hinzuzufügen. Ich habe sofort in Android Studio einen Standardkalender gesehen, den man einfach hinzufügen kann, ohne etwas zu programmieren. Später aber habe ich gemerkt, dass ich mich zu früh gefreut hatte. Nach langem Suchen, wie man denn in diesem Android-Standardkalender einen Termin hinzufügen kann, habe ich feststellen müssen, dass es keinen Weg gibt, diesen Standardkalender zu verändern. Dieser Kalender ist nur dafür gedacht, das Datum anzuzeigen, ohne dass man die Tage irgendwie markieren kann.
Das ist der Standardkalender von Android:

```
<CalendarView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

Bild 19: Standardkalender von Android

Deshalb habe ich selber einen Kalender programmieren und diesen grafisch darstellen müssen. Hierzu habe ich zuerst die Theorie zu Kalendern in Java und zur Darstellungsart in der GridView gelesen. Somit habe ich einen Kalender erstellt und die Daten selber in einer GridView anzeigen lassen. Die GridView ist nichts anderes als ein Raster.

Code dazu:

```
//So erstelle ich einen Kalender und Adapter
public GregorianCalendar cal_month = (GregorianCalendar)
GregorianCalendar.getInstance();
cal_adapter = new
CalendarAdapter(this,cal_month,CalendarCollection.date_collection_arr);
```

Das Raster fülle ich mit meinem Adapter, den ich selbst definiert habe.

Code dazu:

```
GridView gridview = (GridView) findViewById(R.id.gv_calendar);
gridview.setAdapter(cal_adapter);
```

Der Adapter dient dazu, mehrere Elemente in einer Liste (ListView) oder einem Raster (GridView) anzeigen zu lassen. Der Adapter geht durch das Raster und fügt an jeder Stelle ein Element ein. Die Elemente müssen in einem Array angegeben werden. Da ich die Elemente, in diesem Fall sind das die Monatstage, nicht einfach im Raster verteilen, sondern diese auch unterschiedlich darstellen will, muss ich selber einen Adapter definieren. In diesem Adapter definiere ich, dass das aktuelle Datum hellgrün erscheinen soll, die Monatstage des vorherigen bzw. des nächsten Monats grau erscheinen sollen und das Datum mit einem Termin dunkelgrün dargestellt werden soll.

Das ist der Code dazu:

```
/* Beim Füllen des Rasters überprüfe ich ob der Monatstag auf dieser Position im
Raster dem heutigen Monatstag entspricht und stelle die Schriftgrösse, Hintergrund-
und Schriftfarbe ein */

if (day_string.get(position).equals(curentDateString)) {

    v.setBackgroundColor(Color.parseColor("#00343434"));
    dayView.setTextSize(20);
    dayView.setTextColor(Color.parseColor("#00FF00"));
} else {
    v.setBackgroundColor(Color.parseColor("#00343434"));
}
```

```
/* Hier mache ich aus dem Datum-String einen Integer und überprüfe, ob dieser zum
aktuellen Monat gehört. Falls nicht, setze ich die Schriftfarbe auf grau ,
andernfalls auf weiss. */

if ((Integer.parseInt(gridvalue) > 1) && (position < firstDay)) {
    dayView.setTextColor(Color.GRAY);
    dayView.setClickable(false);
    dayView.setFocusable(false);
} else if ((Integer.parseInt(gridvalue) < 15) && (position > 28)) {
    dayView.setTextColor(Color.GRAY);
    dayView.setClickable(false);
    dayView.setFocusable(false);
} else {
    dayView.setTextColor(Color.WHITE);
}


/* Das ist die Funktion, um das Datum mit einem Termin dunkelgrün zu markieren.
Hier gehe ich durch jeden gespeicherten Eintrag und überprüfe, ob das Datum
übereinstimmt. Die Termine habe ich in einem Array gespeichert. Dieser Array
besteht aus Objekten der Klasse CalendarCollection, die ich erstellt habe. */

public void setEventView(View v,int pos){

    int len=CalendarCollection.date_collection_arr.size();
    for (int i = 0; i < len; i++) {
        CalendarCollection cal_obj=CalendarCollection.date_collection_arr.get(i);
        String date=cal_obj.date;

        if (day_string.get(pos).equals(date)) {
                TextView dayView = (TextView) v.findViewById(R.id.date);
                dayView.setTextColor(Color.parseColor("#009900"));

        }
    }
}


/* Hier noch die Klasse der Termine. Sie enthält ein Datum und die Nachricht des
Termins */
public class CalendarCollection {
    public String date="";
    public String event_message="";

    public static ArrayList<CalendarCollection> date_collection_arr=new
ArrayList<CalendarCollection>();
    public CalendarCollection(String date,String event_message){

        this.date=date;
        this.event_message=event_message;
    }

}
```

Durch das Klicken auf die grünen Tasten setze ich den erstellten Java Kalender jeweils auf den nächsten bzw. vorherigen Monat ein und lasse noch einmal das GridView füllen.

```
// Hier das Beispiel mit dem Linkspfeil
ImageButton previous = (ImageButton) findViewById(R.id.ib_prev);
previous.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        setPreviousMonth();
        refreshCalendar();
    }
});
```

Um einen Termin hinzuzufügen, muss man lange auf den gewünschten Monatstag klicken. Das habe ich mit der Funktion setOnItemLongClickListener erreicht. Das heisst, dass die App auf einen langen Klick auf ein Element des Rasters wartet. Danach wird ein kleines Fenster geöffnet, ein Dialog, in den man die Nachricht des Termins aufschreibt und hinzufügt. Gleichzeitig wird der hinzugefügte Termin in eine SQLite-Datenbank gespeichert und ein Alarm Manager gestartet, um uns am vorherigen Tag eine Benachrichtigung zur Erinnerung anzuzeigen.

```
// Ich füge meinem gridview einen OnItemLongClickListener hinzu
gridview.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {

    public boolean onItemLongClick(AdapterView<?> arg0, final View arg1,
                                   final int position, long arg3) {
// Erstelle ein Dialog, ein kleines Fenster, indem die Informationen erscheinen
        final Dialog dialog = new Dialog(KalenderScreen.this);
        dialog.setTitle("Add Event");
        dialog.setContentView(R.layout.add_event);
        dialog.show();
        final TextView datum = (TextView)dialog.findViewById(R.id.datum);
        final EditText event = (EditText)dialog.findViewById(R.id.write_event);
        Button cancel_btn = (Button)dialog.findViewById(R.id.cancel_event);
        Button add_btn = (Button)dialog.findViewById(R.id.add_event);
        datum.setText(CalendarAdapter.day_string.get(position));
// Wenn man auf „abbrechen“ drückt
        cancel_btn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                dialog.cancel();
            }
        });
// Wenn man auf „hinzufügen“ drückt
        add_btn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
// Eine Datenbank erstellen/öffnen
                createDataBase();
                String event_date = datum.getText().toString();
                String event_text = event.getText().toString();
// Hier wird der Termin in die SQLite-Tabelle „events“ hinzugefügt
                eventsDB.execSQL("INSERT INTO events (date, event) VALUES ('" +
event_date + "', '" + event_text + "');");
```

Die Funktion zum Erstellen der Datenbank „MyEvents“ :

```
public void createDataBase(){
    try{
        eventsDB = this.openOrCreateDatabase("MyEvents",MODE_PRIVATE,null);
        eventsDB.execSQL("CREATE TABLE IF NOT EXISTS events " + "(id integer
primary key, date VARCHAR, event VARCHAR);");
        File database = getApplicationContext().getDatabasePath("MyEvents.db");
        if(!database.exists()){
        } else {
// Ein Toast, also ein Pop-up zeigt “Database Missing” an
            Toast.makeText(this,"Database Missing",Toast.LENGTH_SHORT).show();
        }
    }
    catch (Exception e){
        Log.e("EVENTS ERROR", "Error Creating Database");
    }
}
```

Danach wird der Kalender aktualisiert und die hinzugefügten Termine werden nun sichtbar.
Das Löschen der Termine erfolgt durch das Angeben der ID. Somit wird nur der Eintrag mit der angegebenen ID aus der SQLite-Tabelle gelöscht.

```
public void deleteEvent(View view){
    EditText idEditText = (EditText) findViewById(R.id.idEditText);
    String id = idEditText.getText().toString();
    eventsDB.execSQL("DELETE FROM events WHERE id = " + id + ";");
    CalendarCollection.date_collection_arr.clear();
    finish();
    startActivity(getIntent());
    idEditText.setText("");
}
```

Oder es wird die ganze Tabelle mit allen Terminen gelöscht. Hier habe ich aber einen Zwischenschritt einbauen müssen, da der Nutzer vielleicht die Taste nur aus Versehen gedrückt hat. Deshalb wird vor dem Löschen nach einer Bestätigung gefragt. Dies habe ich mit einem AlertDialog gemacht.

```
// Wenn man auf „Datenbank löschen“ drückt wird ein AlertDialog angezeigt
public void deleteButtonClick(View view) {
    AlertDialog diaBox = AskOption();
    diaBox.show();
}

  private AlertDialog AskOption()
{
    AlertDialog myQuittingDialogBox =new AlertDialog.Builder(this)
            .setTitle("Datenbank löschen")
            .setMessage("Sind Sie sicher, dass Sie alle Events löschen wollen?")
// Wenn man auf „Ja“ drückt
            .setPositiveButton("Ja", new DialogInterface.OnClickListener() {

                public void onClick(DialogInterface dialog, int whichButton) {
                    deleteDatabase();
                    dialog.dismiss();
                }

            })
// Wenn man auf „Nein“ drückt
            .setNegativeButton("Nein", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                    dialog.dismiss();
                }
            })
            .create();
    return myQuittingDialogBox;
}
```

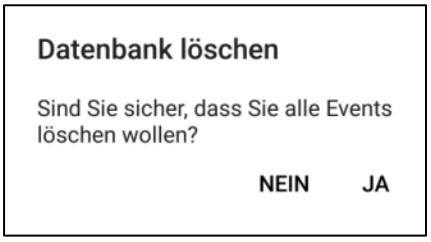


Bild 20: AlertDialog, um die Bestätigung einzuholen

Nun bleibt nur noch die Benachrichtigungsfunktion übrig, welche den Nutzer an einen Termin erinnert. Solche Benachrichtigungen werden in Android Notifications oder auch Push-Notifications genannt. Es gibt eine Art von Notifications, welche vom Entwickler an alle Nutzer dieser App gleichzeitig gesendet wird. Dies wird dann genutzt, wenn der Entwickler beispielsweise alle Nutzer auf ein Update aufmerksam machen will, oder auch sonst etwas mitteilen will. Ich habe aber eine andere Art von Notifications benötigt. Ich habe eine Notification verwendet, welche dem Nutzer von seinem eigenen Smartphone aus gesendet wird.
Nach dem Hinzufügen eines neuen Termins wird ein neues Kalenderobjekt in Java erstellt. Dieses Kalenderobjekt wird auf das Datum des vorherigen Tages vor dem angegebenen Termin eingestellt. Die Uhrzeit wird auf 11:00 Uhr eingestellt.
Das ist der Code dazu:

```
/* Hier überprüfe ich gleichzeitig, ob es der Erste Tag vom Monat ist. Falls der Termin am ersten Tag des Monats gesetzt
wird, muss die Benachrichtigung am letzten Tag des vorherigen Monats angezeigt werden.*/
Calendar c = Calendar.getInstance();
if ( day == 1){
   c.set(year, month-1, cal_adapter.getMaxP() );
   c.set(Calendar.HOUR_OF_DAY, 11);
   c.set(Calendar.MINUTE, 0);
   c.set(Calendar.SECOND, 0);
} else {
   c.set(year, month, day-1 );
   c.set(Calendar.HOUR_OF_DAY, 11);
   c.set(Calendar.MINUTE, 0);
   c.set(Calendar.SECOND, 0);
}
```

Danach wollen wir, dass an diesem Datum eine Benachrichtigung erscheinen soll. Wir wollen quasi dem Programm einen Wecker stellen. Damit dieser Wecker auch funktioniert, wenn die App geschlossen ist, brauchen wir einen Service. Ein Service in Android dient dazu, damit das Programm etwas im Hintergrund macht. Somit können wir unser Programm auch schliessen und der Wecker ist immer noch gestellt. Wenn dieser Wecker „läutet“, wird dem Nutzer eine Benachrichtigung geschickt. Aus Zeit- und Platzgründen kann ich leider nicht alle Details genau erklären und jeden einzelnen Schritt zeigen. Ausserdem braucht es auch etwas mehr Vorkenntnisse, um alle Schritte zu verstehen. Deshalb zeige ich nur die wichtigsten Schritte und versuche, diese verständlich zu kommentieren.

Das ist der Code dazu:

```
// Wir sagen dem scheduleClient, dass er einen "Wecker" stellen soll und geben ihm als Parameter unser Kalenderobjekt
scheduleClient.setAlarmForNotification(c);
```

Die eigentliche Funktion, die unsere Benachrichtigung erscheinen lässt, wenn der „Wecker“ läutet:

```
private void showNotification() {
    // Der Titel der Benachrichtigung
    CharSequence title = "Terminerinnerung";
    // Das Bild in der Benachrichtigung
    int icon = R.drawable.kalender;
    // Der Text der Benchrichtigung
    CharSequence text = "Sie haben morgen einen Termin! ";
    /* Um welche Zeit die Benachrichtigung angezeigt wird. In diesem Fall genau
dann, wenn der Wecker läutet.   */
    long time = System.currentTimeMillis();



    /* Falls der Nutzer auf die Benachrichtigung klickt, wird der App-Bereich mit
dem Kalender geöffnet*/

    PendingIntent contentIntent = PendingIntent.getActivity(this, 0, new
Intent(this, KalenderScreen.class), 0);

    // Die Gestaltung der Benachrichtigung

    NotificationCompat.Builder builder = new NotificationCompat.Builder(
            this);
    Notification notification = builder.setContentIntent(contentIntent)
            .setSmallIcon(icon).setTicker(text).setWhen(time)
            .setAutoCancel(true).setContentTitle(title)
            .setContentText(text).build();
    mNM.notify(NOTIFICATION, notification);

    // Die Benachrichtigung löschen, falls sie angeklickt wurde

    notification.flags |= Notification.FLAG_AUTO_CANCEL;

    // Das eigentliche Senden der Benachrichtigung zum System, also zum Smartphone

    mNM.notify(NOTIFICATION, notification);

    // Den Service wieder stoppen, nachdem die Benachrichtigung angezeigt wurde

    stopSelf();
}
```

Bild 21: So sieht die Benachrichtigung aus, wie sie auf dem Smartphone erscheint.

4.2 Agenda

Nach dem Erstellen des Kalenders, habe ich mich mit der Agenda befasst. Als Erstes habe ich den Bereich „Prüfungen“ erstellt, da dieser fast identisch mit dem Bereich „Kalender“ ist. Dabei bin ich genau gleich vorgegangen. Die einzigen Unterschiede waren das Design und eine andere Datenbank für die Prüfungen. Somit werden im Prüfungskalender nur die Prüfungstermine und im anderen Kalender die alltäglichen Termine angezeigt. Dies ermöglicht eine bessere Übersicht, da die Termine so nicht miteinander vermischt werden. Auch die Benachrichtigungen haben einen eigenen Service und sehen etwas anders aus.

Bild 22: Prüfungsbenachrichtigung

Danach habe ich den Bereich „Hausaufgaben“ programmiert. Zuerst habe ich das Design gestaltet. Ich habe ein Hintergrundbild festgelegt und darüber eine Tabelle in XML erstellt. Da die Standardtabelle keine Linien für die Unterteilung der Zeilen und Spalten hat, habe ich die Tabelle selber bearbeiten müssen. Die Linien habe ich erstellt, indem ich für jedes einzelne Element der Tabelle ein Rechteck aus schwarzen Linien als Hintergrund festgelegt habe.
XML-Code dazu:

```
android:background="@drawable/cell_shape"
```

Hier wird beim Definieren des Hintergrundes auf ein „drawable“ verwiesen, cell_shape.xml. Dieses habe ich selber erstellt und im „drawable“ Ordner gespeichert. Darin definiere ich eine Form für die Zelle:

```
<?xml version="1.0" encoding="utf-8"?>
    <shape
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:shape= "rectangle"  >
        <solid android:color="#00000000"/>
        <stroke android:width="1dp"  android:color="#000000"/>
    </shape>
```

Da die Tabelle zwölf Zeilen aufweist, hat sie nicht auf dem Bildschirm gepasst. Deshalb habe ich die gesamte Tabelle in eine ScrollView verpackt. Somit kann man die Tabelle hinauf- und hinunterscrollen, um alle Zeilen zu sehen. Die Tabelle hat drei Spalten: Die erste Spalte ist für die Uhrzeit, die zweite Spalte für das Fach und die dritte Spalte für die Hausaufgaben.
Beim Speichern wird eine SQLite-Tabelle erstellt und jede Zeile und jede Spalte werden analog übernommen. Dazu habe ich jeder Zelle eine eigene ID geben müssen, was sehr viel Zeit und Aufwand gekostet hat. Die SQLite-Datenbank und SQLite-Tabelle habe ich genau gleich wie im Bereich „Kalender“ erstellt, nur hat diese Tabelle eine andere Anzahl Spalten, einen anderen Tabellennamen und andere Spaltenbezeichnungen. Beim Löschen wird nur die dritte Spalte mit den Aufgaben gelöscht, da man den Stundenplan in den meisten Fällen behalten möchte.
Das habe ich jeweils für den Montag, Dienstag, Mittwoch, Donnerstag und Freitag gemacht. Um zwischen den Tagen zu navigieren, habe ich blaue Pfeile erstellt. Durch das Klicken auf den Pfeil wird der aktuelle Tag geschlossen und jeweils der nächste oder der vorherige Tag geöffnet.
Im XML definiere ich ein Bild (ImageView) wie folgt:

```
<ImageView
    android:layout_width="45dp"
    android:layout_weight="1"
    android:layout_height="45dp"
    android:onClick="montagNext"
    android:src="@drawable/arrow_right_blue"/>
```

Mittels “onClick=montagNext” wird nach dem Klicken auf das Bild die Funktion „montagNext“ aufgerufen. Diese habe ich im dazugehörenden Java-File definiert:

```
/* Diese Funktion schliesst die geöffnete Activity(die aktuelle Seite) und öffnet
eine andere mithilfe eines Intents*/

public void montagNext(View view) {
    Intent dienstagScreenIntent = new Intent(this, DienstagScreen.class);
    finish();
    startActivity(dienstagScreenIntent);
}
```

Nach dem Bereich „Aufgaben“ habe ich den Bereich Noten programmiert. Dieser Teil war nicht sehr schwierig zu programmieren, aber sehr aufwendig. Ähnlich wie bei den „Aufgaben“ habe ich eine Tabelle mit XML erstellt. Die Tabelle besteht aus 16 Zeilen und jeweils 2 Spalten. Jede Zeile ist für ein Schulfach zuständig. Die erste Spalte beinhaltet den Namen des Fachs und die zweite Spalte die dazugehörigen Noten sowie die Durchschnittsnote in diesem Fach. Das Speichern und Löschen der Noten erfolgt analog zum Bereich „Aufgaben“. Die grösste Herausforderung hier ist gewesen, den vielen Feldern eine eigene ID zuzuweisen. Da ich 16 Zeilen und in jeder Zeile jeweils 9 Felder habe, auf die ich zugreifen muss, habe ich 16 * 9, also 144 IDs vergeben müssen. Somit sind die XML- und Java-files von diesem Bereich die umfangreichsten in meiner App: Das XML-file hat rund 1600 und das Java-File rund 11 000 Zeilen.
Beim Eintragen der Noten wird die Durchschnittsnote sofort berechnet. Das habe ich mit einem TextChangedListener erreicht. Dieser überprüft ständig, ob der Text im Notenfeld verändert wurde oder nicht. Falls der Text verändert wurde, also eine Note eingetragen oder gelöscht wurde, werden die Noten in einer Liste gespeichert. Danach werden alle Noten in dieser Liste addiert und durch die Anzahl Noten geteilt. Schliesslich wird noch auf zwei Nachkommastellen gerundet und im letzten Feld ausgegeben.
Das ist der Code dazu:

```
//Hier füge ich dem Notenfeld f1n1 einen TextChangedListener hinzu
f1n1.addTextChangedListener(new TextWatcher() {
//Das wird ausgeführt, wenn der Text verändert wird
    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
        ArrayList<Double> ar = new ArrayList<Double>();

/* Hier überprüfe ich, ob das Notenfeld leer ist. Falls es nicht leer ist, füge ich
die Note in die Liste „ar“ hinzu */
        if(f1n1.getText().toString().trim().isEmpty()){} else{
            String as = f1n1.getText().toString();
            double a = Double.parseDouble(as);
            ar.add(a);
        }
```

Das Gleiche habe ich für jedes einzelne Feld gemacht (für f1n2, f1n3…). Aus Platzgründen lasse ich es weg

```
/*Hier definiere ich eine Variable sum vom Typ double, iteriere durch jeden Element
der Liste ar, berechne den Durchschnitt und runde schliesslich auf 2
Nachkommastellen. Das Ergebnis schreibe ich ins Feld resultat_1*/
        double sum = 0.00;
        for (Double i : ar){
            sum += i;
        }
        double resultat = Math.round(100.0 * (sum/ar.size())) / 100.0;
        resultat_1.setText(Double.toString(resultat));
```

Gleich danach habe ich den Bereich „Promotion“ programmiert. Hier werden einfach die Pluspunkte und die Durchschnittsnote von allen Fächern berechnet und angezeigt. Bei den Pluspunkten wird die doppelte Kompensation miteinberechnet. Das heisst, dass die negative Differenz zu einer Vier (erreichte Note - 4) doppelt und eine positive Differenz einfach gezählt wird. Ausserdem wird angegeben, ob man definitiv promoviert wird oder nicht. Im Falle einer definitiven Promotion, erscheint ein fröhlicher Smiley. Wenn man aber Minuspunkte hat, erscheint ein trauriger Smiley. Falls man null Pluspunkte hat, erscheint ein zufriedener Smiley.
Der Code dazu sieht wie folgt aus:

```
// Verschiedene Bilder, je nach Pluspunkten
if (Double.parseDouble(pluspunkte.getText().toString())>0){
    pluspunkte.setTextColor(Color.parseColor("#009900"));
    gratulation.setText("Sie werden definitiv promoviert! \nGut gemacht!");
    promotionsbild.setImageResource(R.drawable.good_notes);

} else if (Double.parseDouble(pluspunkte.getText().toString())<0) {
    pluspunkte.setTextColor(Color.parseColor("#990000"));
    gratulation.setText("Sie werden bestenfalls provisorisch promoviert. \nGeben
Sie Gas!");
    promotionsbild.setImageResource(R.drawable.bad_notes);
} else {
    pluspunkte.setTextColor(Color.parseColor("#000000"));
    gratulation.setText("Sie werden knapp definitiv promoviert!");
    promotionsbild.setImageResource(R.drawable.ok_notes);
}

//Verschiedene Textfarbe, je nach Durchschnittsnote
if (Double.parseDouble(notendurchscnittTV.getText().toString())>4){
    notendurchscnittTV.setTextColor(Color.parseColor("#009900"));

} else if (Double.parseDouble(notendurchscnittTV.getText().toString())<4) {
    notendurchscnittTV.setTextColor(Color.parseColor("#990000"));
} else {
    notendurchscnittTV.setTextColor(Color.parseColor("#000000"));
}
```

Bild 23: 2 Pluspunkte

Bild 24: 0 Pluspunkte

Bild 25: -2 Pluspunkte

4.3 Notizbuch

Im Bereich „Notizbuch“ habe ich als Hintergrundbild ein kleines, offenes Notizbuch festgelegt. Zunächst habe ich ein Textfeld und zwei Buttons erstellt: „Speichern“ und „Löschen“. Beim Speichern wird der Inhalt des Textfeldes in eine SQLite-Tabelle gespeichert und beim Löschen wieder entfernt. Das habe ich analog zum Bereich „Aufgaben“ gemacht.
Während einer Besprechung mit meinem Betreuer hat er mich darauf aufmerksam gemacht, dass es sehr nützlich wäre, die Notizen auf irgendeine Art nach Themen zu sortieren.
Schon bald bin ich auf die Idee gekommen, mit einem Spinner zu arbeiten. Ein Spinner ist einfach eine Drop-Down-Liste, aus der man ein Element auswählen kann. Ich habe die Idee gehabt, die Themen in einem Spinner aufzulisten und jeweils ein Thema aus dem Spinner auszuwählen, um zu diesem dann die Notizen zu schreiben. Ich habe bereits die Idee gehabt, aber ich habe nicht gewusst, wie man diese Idee umsetzen kann.
Nach langem Überlegen bin ich schlussendlich zu einer Lösung gekommen: Sie war gar nicht so schwierig, wie ich es mir zuvor vorgestellt hatte.
Ich habe einfach einen Spinner mit Themen gefüllt. Als die Notizen dann gespeichert wurden, habe ich in der SQLite Tabelle eine Spalte „Thema“ hinzugefügt. Somit wurden die Notiz und das Thema gemeinsam gespeichert.
Um die Notizen aus der SQLite-Tabelle zu holen, habe ich den Befehl „WHERE“ gebraucht. Je nach ausgewähltem Element im Spinner, habe ich verschiedene Notizen anzeigen lassen.
Code dazu:

```
// Hier hole ich die Notiz aus der Datenbank mit dem ausgewählten Thema
final EditText notizen = (EditText) findViewById(R.id.notizen_edit_text);
Spinner spinner = (Spinner) findViewById(R.id.notizbuch_spinner);
String thema = spinner.getSelectedItem().toString();
Cursor cursor = notizbuchDB.rawQuery("SELECT notiz FROM notizen WHERE thema = '" +
thema + "'", null);
if (cursor.moveToFirst()) {
    do {
        String notiz = cursor.getString(0);
        notizen.setText(notiz);
    } while (cursor.moveToNext());
}
cursor.close();
notizbuchDB.close();
```

Später habe ich noch eine zusätzliche Activity erstellt, um die Themen zu verwalten. Man soll auch Themen hinzufügen oder löschen können und zu eigenen Themen Notizen machen. Wenn man auf das rote Pluszeichen klickt, kommt man zur Avtivity, in der man die Themen bearbeiten kann. Dazu habe ich eine neue SQLite-Tabelle erstellt, in der ich nur die Themen speichere. Diese werden dann aus der Datenbank direkt in den Spinner geholt. Das Löschen funktioniert gleich wie beim Kalender, indem man das Thema, das man löschen will, aus dem Spinner auswählt und auf den Button „Löschen“ drückt.
Code dazu:

```
/* Funktion, um die Themen aus der Datenbank zu holen und in den Spinner zu legen*/

private void loadSpinnerData() {

    ThemaHinzufugenDbHelper db = new
ThemaHinzufugenDbHelper(getApplicationContext());

    List<String> lables = db.getAllLabels();

    ArrayAdapter<String> dataAdapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_spinner_item, lables);

    dataAdapter

.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

    spinner.setAdapter(dataAdapter);
}


// Wenn man auf „Thema hinzufügen“ drückt

btnAdd.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View arg0) {
        String label = inputLabel.getText().toString();

        if (label.trim().length() > 0) {
            ThemaHinzufugenDbHelper db = new ThemaHinzufugenDbHelper(
                    getApplicationContext());
// Hinzufügen eines neuen Themas in die Datenbank
            db.insertLabel(label);
            db.insertNotiz(label);
// Inputfeld wieder leeren
            inputLabel.setText("");

// Die Tastatur wieder verstecken
            InputMethodManager imm = (InputMethodManager)
getSystemService(Context.INPUT_METHOD_SERVICE);
            imm.hideSoftInputFromWindow(inputLabel.getWindowToken(), 0);
// Noch einmal den Spinner mit neu hinzugefügtem Thema füllen
            loadSpinnerData();
        } else {
/* Eine Fehlermeldung anzeigen, falls das Feld mit dem Thema beim Hinzufügen leer
ist */
            inputLabel.setError("Das Feld darf nicht leer sein");
        }
    }
});
```

4.4 Einkaufszettel

Danach habe ich den Bereich „Einkaufen“ programmiert. Hier habe ich das Hintergrundbild so festgelegt, dass oben ein Feld platziert ist, in das man die Menge und den Artikel hineinschreiben kann und diesen Artikel per Button-Klick in den Einkaufszettel hinzufügen kann. Die Mengeneinheiten sind in einem Spinner standardmässig wie folgt definiert: x für Stück, kg für Kilogramm, g für Gramm und l für Liter. Die Funktion, eine Einheit hinzuzufügen, habe ich weggelassen, da diese Mengeneinheiten für das alltägliche Einkaufen genügen sollten. Unterhalb ist ein Einkaufszettel platziert, auf dem die Kategorie und die Artikel angezeigt werden. Die Artikel werden in einer ListView angezeigt. Die Kategorien sind ähnlich wie im Bereich „Notizbuch“ in einem Spinner dargestellt. Es gibt vier Standardkategorien zur Auswahl: Apotheke, Bäckerei, Metzgerei und Supermarkt. Falls man eine Kategorie hinzufügen oder löschen möchte, kann man das analog zur Notizbuch-Activity tun. Der grösste Unterschied zur Notizbuch-Activity besteht darin, dass es nicht nur eine grosse Notiz gibt, sondern mehrere Einträge aufgelistet werden. Das habe ich folgendermassen erreicht: Zuerst, wenn man einen neuen Artikel zur Einkaufsliste hinzufügt, wird dieser sofort in einer SQLite-Tabelle, zusammen mit seiner Kategorie, gespeichert.
So sieht der Code dazu aus:

```
// Die Funktion zum Speichern des Artikels in der SQLite-Datenbank

public void einkaufenSpeichern() {

    createDataBase();

    EditText menge_einkaufen =(EditText) findViewById(R.id.menge_einkaufen);
    Spinner spinner_einheit = (Spinner) findViewById(R.id.spinner_einheit);
    AutoCompleteTextView artikel_einkaufen =(AutoCompleteTextView)
findViewById(R.id.artikel_einkaufen);
    Spinner einkaufen_spinner = (Spinner) findViewById(R.id.einkaufen_spinner);

/* Der gesamte String, der in die SQLite Tabelle als ein Eintrag in die Spalte
„Einkauf“ gespeichert wird, ist zusammengesetzt aus der Anzahl, der Mengeneinheit
und dem Artikelnamen. Die Kategorie wird in der benachbarten Spalte gespeichert. */

    String einkauf = menge_einkaufen.getText().toString() + " "
            + spinner_einheit.getSelectedItem().toString() + " "
            + artikel_einkaufen.getText().toString();
    String kategorie = einkaufen_spinner.getSelectedItem().toString();
    einkaufenDB.execSQL("INSERT INTO einkaufen (kategorie, einkauf) VALUES ('" +
kategorie + "', '" + einkauf + "');");

    einkaufenDB.close();

}
```

Danach ermittelt das Programm, welche Kategorie momentan im Spinner ausgewählt ist. Schliesslich werden alle Einträge aus der SQLite-Tabelle, die dieser Kategorie angehören, ausgewählt und der Reihe nach in eine Liste gefüllt. Da wir nicht wissen, wie viele es am Schluss sind, müssen wir mit einer Liste arbeiten. Ein Array muss hingegen immer wissen, aus wie vielen Elementen es bestehen wird. Erst nach dem Befüllen der Liste wandeln wir diese in ein Array um. Zum Schluss ergänzen wir unsere ListView mithilfe eines Adapters. Wie bereits erwähnt, dient ein Adapter dazu, mehrere Elemente in Form eines Arrays, in eine Liste oder ein Raster einzufügen.
Das ist der Code dazu:

```
// Erstellen einer Liste
ArrayList<String> einkauflist = new ArrayList<String>();
// Auf den Spinner zugreifen und das ausgewählte Element unter der Variable
//„kategorie" speichern
Spinner spinner = (Spinner) findViewById(R.id.einkaufen_spinner);
final String kategorie = spinner.getSelectedItem().toString();
// Alle Einträge mit der Kategorie, die im Spinner ausgewählt ist, aus der
//Datenbank auswählen und der Reihe nach in die erstellte Liste hineinfügen.
Cursor cursor = einkaufenDB.rawQuery("SELECT einkauf FROM einkaufen WHERE einkauf
IS NOT NULL AND kategorie = '" + kategorie + "'", null);
if (cursor.moveToFirst()) {
    do {
        einkauflist.add(cursor.getString(0));
    } while (cursor.moveToNext());
}
```

Die ListView, welches die Artikel anzeigt, habe ich mit einer Multiple-Choice-Box ergänzt. Somit kann man die gekauften Artikel abhaken.
Das ist der Code dazu:

```
EinkaufenListView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
```

Ausserdem habe ich zur besseren Übersicht die angeklickten Artikel grau und durchgestrichen erscheinen lassen. Das habe ich erreicht, indem ich eine Liste der abgehakten Artikel erstellt habe. Mit einem onItemClickListener, also jedes Mal, wenn ein Artikel angeklickt wird, wird er in dieser Liste gespeichert. Danach wird das angeklickte Element der ListView grau und durchgestrichen dargestellt. Falls dieser Artikel bereits in der Liste vorhanden ist, wird er wieder herausgelöscht und das angeklickte Element wird schwarz und nicht durchgestrichen angezeigt. Somit kann der Nutzer das Durchstreichen jederzeit rückgängig machen.

Der Code dazu sieht wie folgt aus:

```
// Der ListView mit den Artikeln einen onItemClickListener hinzufügen

EinkaufenListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long
id) {
        String selectedItem = ((TextView)view).getText().toString();
        TextView selectedItemTV = ((TextView)view.findViewById(R.id.textView1));

/* Überprüfen, ob das angeklickte Element bereits in der Liste vorhanden ist, also
bereits angeklickt wurde oder nicht. */

        if(selectedItems.contains(selectedItem)){

// Das Element aus der Liste löschen und den Text wieder schwarz machen
            selectedItems.remove(selectedItem);
            selectedItemTV.setPaintFlags(selectedItemTV.getPaintFlags() &
(~Paint.STRIKE_THRU_TEXT_FLAG));
            selectedItemTV.setTextColor(Color.parseColor("#000000"));
            selectedItemTV.setTypeface(Typeface.DEFAULT_BOLD);
        }
        else {

// Den Text durchgestrichen und grau darstellen
            selectedItems.add(selectedItem);
            selectedItemTV.setPaintFlags(selectedItemTV.getPaintFlags() |
Paint.STRIKE_THRU_TEXT_FLAG);
            selectedItemTV.setTextColor(Color.parseColor("#cccccc"));
            selectedItemTV.setTypeface(Typeface.DEFAULT);
        }
    }
});
```

Wenn man auf den roten Knopf „Löschen“ drückt, wird ähnlich wie in allen anderen Bereichen der App zuerst nach einer Bestätigung gefragt und danach die gesamte ListView mit allen Artikeln dieser Kategorie gelöscht. Falls man aber nur einen Artikel aus der Liste entfernen will, kann man das mit einem langen Klicken auf das Element erreichen. Dabei erscheint ein kleines Fenster, um den Nutzer zu fragen, ob er diesen Artikel wirklich löschen will.
Das ist der Code dazu:

```
// Den ausgewählten Artikel aus der Datenbank löschen
        einkaufenDB.execSQL("DELETE FROM einkaufen WHERE einkauf = '" +
selectedItem+ "' AND kategorie = '" + kategorie + "';");
        selectedItems.remove(selectedItem);
// Die ListView neu laden, damit das gelöschte Element nicht mehr angezeigt wird
        getEinkaufen();
        einkaufenDB.close();
        dialog.cancel();
```

Am Schluss habe ich noch eine kleine Funktion hinzugefügt, die das Erstellen von Einkaufszetteln ein kleines bisschen einfacher macht. Ich habe die TextView, wo man den Artikelnamen eintippen muss, zu einer AutoCompleteTextView gemacht. Eine AutoCompleteTextView bringt beim Eintippen Vorschläge, die man auswählen kann. Somit geht das Eintippen viel schneller.
Dazu habe ich ein Array mit Wörtern erstellen müssen, welche als Vorschläge beim Eintippen erscheinen.
Das ist der Code dazu:

```
// Array mit den Wörtern für die Vorschläge
String[] produkte=

{"Kopfsalat","Eisbergsalat","Chicorée","Endivie","Rucola","Spinat","Kohl","Steckrübe","Blumenkohl","Broccoli","Kohlrabi","Rotkohl","Weisskohl","Artischocke","Zucchini","Wassermelone","Gurken","Gewürzgurke","Kürbisse","Tomaten","Paprika","Aubergine","Avocado","Rote
Bete","Meerrettich","Radieschen","Wasabi","Kartoffeln","Petersilie",
"Sellerie","Süsskartoffel","Zwiebel","Knoblauch","Lauch","Bohnen","Erbsen","Linsen","Äpfel","Birnen","Aprikose","Kirschen","Pflaumen","Zwetschgen","Pfirsiche","Nektarine","Himbeere","Brombeere","Erdbeere","Johannisbeere","Heidelbeere","Erdnüsse",
"Cashew","Kastanien","Haselnüsse","Kokosnuss","Mandeln","Pistazien","Walnüsse","Ananas","Bananen","Limetten","Mandarinen","Orangen","Grapefruit","Zitronen","Feige",
"Granatapfel","Kaki","Kiwi","Mango","Melone","Müsli","Wasser",
Wasser","Mineralwasser","Bier","Limonade","Coca-
Cola","Fanta","Sprite","Apfelsaft","Orangensaft","Trauben","Traubensaft",
"Multivitaminsaft","Tee","Kaffee","Wein","Milch","Joghurt","Schokolade","Gummibärchen","Pralinen","Brot","Butter","Käse","Wurst","Schweinefleisch","Hühnerfleisch","Lamm","Rindfleisch","Hackfleisch","Pizza","Pommes Frites","Ketchup","Mayonnaise",
"Senf","Salz","Pfeffer","Sojasauce","Chips","Schinken","Würstchen","Brötchen","Rahm","Popcorn","Mineralwasser","Sirup","Kaugummi","Fisch","Lachs","Hering","Toilettenpapier","Zahnpasta","Shampoo","Seife","Haushaltspapier","Putzmittel","Waschmittel"};

//Auf die AutoCompleteTextView zugreifen
AutoCompleteTextView artikel_kaufen = (AutoCompleteTextView)
findViewById(R.id.artikel_einkaufen);
//Einen Adapter erstellen
ArrayAdapter adapterProdukte = new
ArrayAdapter(this,android.R.layout.simple_list_item_1,produkte);
//Den Adapter der AutoCompleteTextView zuweisen
artikel_kaufen.setAdapter(adapterProdukte);
//Definieren, ab wieviel Zeichen ein Vorschlag erscheinen soll
artikel_kaufen.setThreshold(1);
```

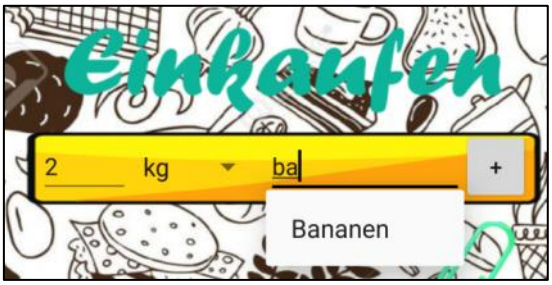
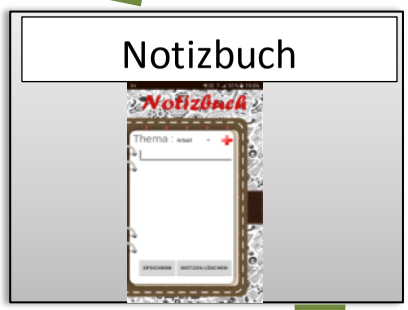


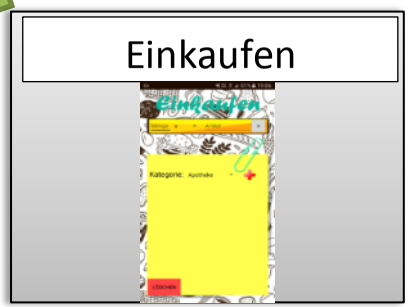
Bild 26: AutoCompleteTextView

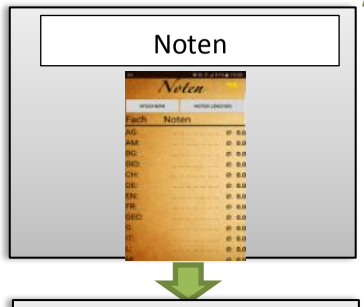
5. Zusammenfassung

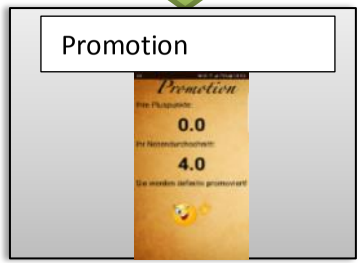
Damit nach all diesen Erklärungen und Quellcode-Beispielen die Übersicht über die App nicht verloren geht, fasse ich zusammen:
Ich habe selber einen Organizer in Android Studio programmiert. Dieser Organizer hat vier Hauptfunktionen: „Kalender“, „Schule“, „Notizbuch“ und „Einkaufen“.
Im Bereich „Kalender“ kann man Termine eintragen. Die Tage mit einem Termin werden grün dargestellt. Der Nutzer wird jeweils am vorherigen Tag um 11:00 Uhr an den Termin mit einer Benachrichtigung erinnert.
Im Bereich „Schule“ hat es drei Unterbereiche: „Noten“, „Prüfungen“ und „Aufgaben“. Im Bereich „Noten“ kann der Nutzer seine Noten eintragen. Die Durchschnittsnote, sowie die Informationen über den Promotionsstatus werden automatisch angezeigt. Im Bereich „Prüfungen“ hat man einen ähnlichen Kalender wie im Bereich „Kalender“. Somit werden die alltäglichen Termine von den Prüfungsterminen getrennt. Im Bereich „Aufgaben“ kann der Nutzer seinen eigenen Stundenplan eintragen und die Hausaufgaben aufschreiben.
Im Bereich „Notizbuch“ kann man seine Gedanken und Ideen zu verschiedensten Themen aufschreiben.
Im Bereich „Einkaufen“ kann man schnell und einfach einen Einkaufszettel erstellen. Da die App komplett offline ist und alle Daten lokal gespeichert werden, kann man beim Einkaufen den Einkaufszettel hervorholen und die gekauften Artikel durchstreichen. Somit hat man eine bessere Übersicht beim Einkaufen.
Auf der nächsten Seite zeige ich grafisch die Zusammenhänge der verschiedenen Bereiche.

Zusammenhänge der verschiedenen Bereiche der App









6. Erfahrungen

Im Laufe dieser Arbeit habe ich sehr viel gelernt: Ich habe gelernt, wie Android-Apps funktionieren, wie ich mit Fehlern und Problemen beim Programmieren umgehen soll und vieles mehr. Ich habe sehr viel Erfahrungen gesammelt und bin nun viel vertrauter mit dem Programmieren für Android.
Ich bin froh, dass ich dieses Thema ausgewählt habe, da ich eigentlich mein Ziel erreicht habe. Ich habe mein erstes Android-App in meinem Leben programmiert. Ausserdem habe ich auch sehr viel Spass gehabt, an diesem Projekt zu arbeiten und es in die Realität umzusetzen.
Falls jemand, der meine Maturaarbeit liest, auch eine Android-App programmieren möchte, kann ich es nur weiterempfehlen. Es macht sehr viel Spass, selber etwas entwickeln und es selber an seinem eigenen Smartphone nutzen zu können. Dabei muss man weder die Entwicklungsumgebung kaufen noch sonst irgendetwas. Alle Informationen sind kostenfrei im Internet verfügbar. Es gibt viele Openbooks, viele YouTube-Tutorials und auch viele Informationen auf der Google-Entwickler Website. Alles was man braucht, ist ein Smartphone, ein PC, die Entwicklungsumgebung, die eigene Phantasie und Geduld. Und schon hat man selber eine eigene Android-App programmiert.
Die wichtigsten Tipps, die ich geben könnte, sind:
1. Vor dem Programmieren immer einen kurzen Plan erstellen. Einfach den Algorithmus Zeile für Zeile in Wörtern formulieren. Das verschafft eine bessere Übersicht über den eigenen Code und spart extrem viel Zeit beim Debugging, da man mit solch einem kurzen Plan garantiert viel weniger Fehler macht.
2. Sauber zu programmieren, um die Übersicht bei viel Code zu behalten.
3. Als Anfänger das Android Manifest nicht vergessen, in dem man alle Activities definieren muss.
4. Bei Fehlern und Problemen die Website „stackoverflow“ besuchen, dort sind sehr viele Probleme besprochen und gelöst worden.
5. Falls man sehr lange einen Fehler nicht findet oder nicht lösen kann, sollte man einfach mal eine Pause einlegen, oder etwas komplett anderes programmieren. Das Programmieren soll schliesslich Spass machen!

7. Quellenverzeichnis

Openbooks:

- Pant, Marcus und Becker, Arno (2009). *Android - Grundlagen und Programmierung* (S. 354). dpunkt-Verlag.

- Ullenboom, Christian. (2011). *Java ist auch eine Insel* (S. 1308). Rheinwerk Verlag.

Internetquellen:

- http://www.java-tutorial.org/java-eigenschaften.html [23.11.2016]
- https://de.wikipedia.org/wiki/Extensible_Markup_Language [23.11.2016]
- https://www.techonthenet.com/sqlite/tables/create_table.php [23.11.2016]
- https://de.wikipedia.org/wiki/SQLite [23.11.2016]
- https://de.wikipedia.org/wiki/Integrierte_Entwicklungsumgebung [23.11.2016]
- https://de.wikipedia.org/wiki/Android_Studio [23.11.2016]
- http://www.tutorialspoint.com/android/android_studio.htm [23.11.2016]
- http://developer.android.com/guide/index.html [23.11.2016]
- https://developers.google.com/android/for-all/vocab-words/?utm_source=udacity&utm_medium=course&utm_campaign=android_basics [23.11.2016]
- http://www.programmierenlernenhq.de/sqlite-datenbank-upgrade-in-android/ [23.11.2016]
- http://blog.blundellapps.co.uk/notification-for-a-user-chosen-time/ [23.11.2016]
- https://www.youtube.com/user/derekbanas/featured [23.11.2016]
- https://classroom.udacity.com/courses/ud837/lessons/4027328704/concepts/44414294070923 [23.11.2016]
- http://stackoverflow.com/questions/ [24.11.2016]
- https://www.sololearn.com/Course/Java/ [24.11.2016]