

Dokumentation

Jannes Zilling

Ausgewählte Themen der Programmierung

Android Programmierung in Java

SoSe 2020

Abgabe:

04. Oktober 2020

Inhaltsverzeichnis

1. Einleitung	4
2. Grundlagen / Technologien	4
3. Konzept	4
3.1 Skizzen.....	4
4. Realisierung.....	5
4.1 Main Activity	5
4.2 Custom Navigation.....	5
4.3 Add Recipe Activity	7
4.4 Recipe List Activity	11
4.4.1 Zellen Layout.....	14
4.5 MyCursorAdapter Klasse.....	17
4.6 DBSQueries Klasse.....	18
4.7 DBSOpenHandler Klasse	19
5. Zusammenfassung	20
5.1 Erweiterungen.....	20

Abbildungsverzeichnis

Abbildung 1 App-Skizzen.....	4
Abbildung 2 Aufbau einer ListView-Struktur	5
Abbildung 3 ImageView XML Code.....	5
Abbildung 4 Custom Navigation	6
Abbildung 5 Custom Navigation XML Code	6
Abbildung 6 Java Navigationsbutton Methoden	7
Abbildung 7 Layout Add Recipe Activity	8
Abbildung 8 Spinner Array XML Datei	8
Abbildung 9 Spinner ArrayList.....	9
Abbildung 10 Multiline TextView mit ScrollView.....	9
Abbildung 11 UI-Elemente per ID holen	9
Abbildung 12 checkInputs() Methode Java.....	10
Abbildung 13 Speichern-Button Methode fügt Daten in Datenbank	11
Abbildung 14 Gradient per XML Code	11
Abbildung 15 Layout Recipe List Activity	12
Abbildung 16 Sichtbarkeit von mehr Informationen	13
Abbildung 17 ListView	13
Abbildung 18 insertAdapter() Methode	14
Abbildung 19 Zellen Layout	14
Abbildung 20 showRecipeToast() Methode.....	15
Abbildung 21 Java Code für Toggle-Button	15
Abbildung 22 Java Code Recipe Klasse	16
Abbildung 23 Java Code MyCursorAdapter	17
Abbildung 24 getCreateTables() Methode	18
Abbildung 25 insertRecipe() Methode.....	18
Abbildung 26 getRecipeCursor() Methode	19
Abbildung 27 onCreate() und onUpgrade() Methode	20

1. Einleitung

Durch mobile Geräte wird der Zugriff auf Datensätze sehr vereinfacht. So gut wie jeder besitzt heutzutage ein Smartphone. Diese App simuliert eine moderne Rezeptesammlung. Mit dieser App können die Anwender Rezepte verwalten. Mit einer Navigation kann man sich durch die App bewegen. In einer View können neue Rezepte angelegt werden. In einer anderen View wird eine Liste der Rezepte angezeigt. Die Daten (Rezepte) können mit SQLite gespeichert werden.

2. Grundlagen / Technologien

In diesem Android-Projekt wird eine ListView verwendet. Es wird mit der SQLite Technologie eine Datenbank verwendet. Des Weiteren gibt es drei verschiedene Views, die auf drei Activities aufgeteilt werden. Eine wird zum Navigieren genutzt, eine zum Einfügen von Daten, und eine die die Daten darstellt. Es wird eine eigene Navigationsleiste erstellt.

3. Konzept

Die Idee dieser App ist die Anzeige und Verwaltung von "Rezepten" in einer Liste. Zum Einsatz kommt eine ListView. In dieser ListView werden die Rezepte dargestellt. Durch das Klicken auf ein Rezept werden nähere Informationen angezeigt.

3.1 Skizzen

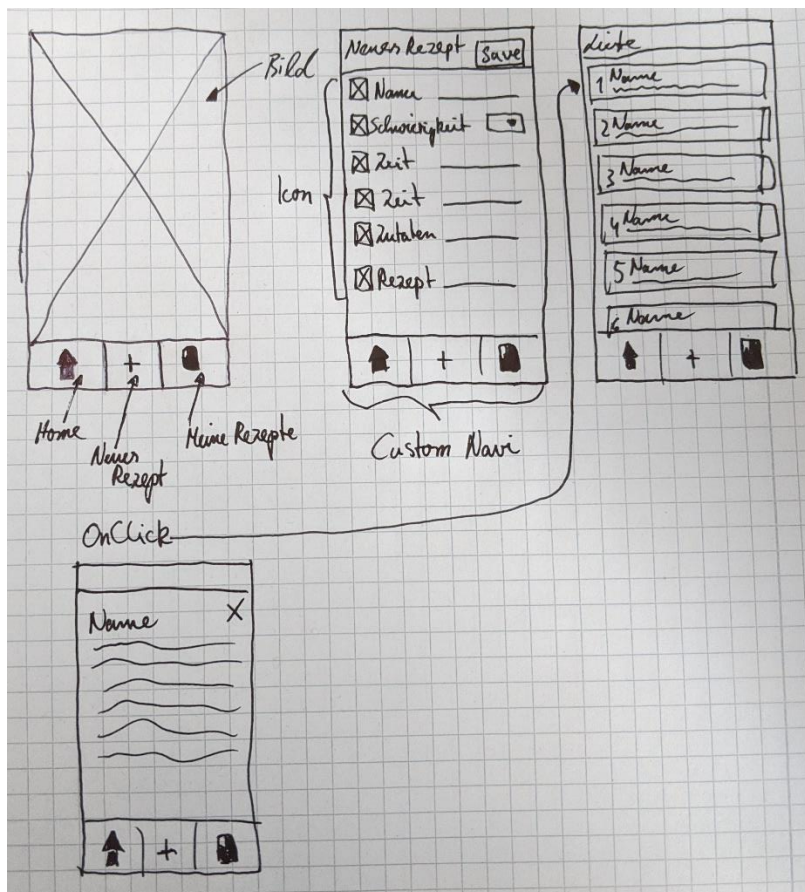


Abbildung 1 App-Skizzen

Ganz links ist die Main Activity zu sehen. Diese beinhaltet die Navigation, um durch diese App zu navigieren. Die Navigationsleiste ist eigens erstellt wurden. In der Navigationsleiste findet man den Home Button, der einen immer auf die Main Activity verweist. In der Mitte befindet sich der Neues Rezept Button, dieser verweist auf eine neue View, der AddRecipe. Mit dieser View können neue Rezepte angelegt werden. In dieser Activity findet man die verschiedenen EditText Name, Arbeitszeit, Koch- /Backzeit. Zutaten und Rezept. Des Weiteren gibt es einen Spinner, um den Schwierigkeitsgrad einzustellen. Mit dem Save Button werden die Daten in die Datenbank aufgenommen. Der rechte Button in der Navigation verweist auf die Activity mit der ListView. Dort werden die Rezepte in einer Liste dargestellt. Beim Klicken auf ein Element in der ListView werden mehr Informationen sichtbar.

4. Realisierung

Die ListView wird in einem Adapter an die Daten gekoppelt. In diesem Android-Projekt wird ein eigener Adapter, mit dem Namen „MyCursorAdapter“ verwendet.



Abbildung 2 Aufbau einer ListView-Struktur

Ein Adapter kapselt die interne Liste und gibt bei Anfrage die aktuelle Zelle als View-Object aus. Das muss mit dem Layout korrespondieren. Intern benötigt der Adapter noch eine Wrapperklasse für die einzelne Zelle.

4.1 Main Activity

Die Main Activity ist der Startpunkt der App. Hier befindet sich die Navigationsleiste. Auf der Navigationsleiste ist der aktuelle View durch ein farbiges Icon erkennbar. Die Main Activity zeigt das Logo der App und ein Hintergrundbild. Das Hintergrundbild wird durch ein imageView dargestellt, sowie das Logo der App.

```

<ImageView
    android:id="@+id/imageView4"
    android:layout_width="491dp"
    android:layout_height="733dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:srcCompat="@drawable/pexels_valeria_boltneva_842571" />

<ImageView
    android:id="@+id/imageView"
    android:layout_width="129dp"
    android:layout_height="193dp"
    app:layout_constraintBottom_toTopOf="@+id/linearLayout"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:srcCompat="@drawable/ic_local_dining_black_24dp" />
  
```

Abbildung 3 ImageView XML Code

4.2 Custom Navigation

Die Navigation der App, wurde eigens für die App entwickelt. Die Navigationsleiste besteht aus einem LinearLayout. In diesem LinearLayout befinden sich drei Button. Diese Buttons sind für die Navigation durch die App verantwortlich. Der erste Button ist der Home-Button mit diesem gelangt man immer auf die Main Activity. Der mittlere Button ist dafür da, um auf die Add Recipe Activity zu gelangen. Mit dieser Activity kann der Anwender ein

neues Rezept, mit bestimmten Spezifikationen, in die Datenbank einfügen. Der rechte Button ist für das Anzeigen der Activity mit der ListView zuständig.

Die Buttons zeigen ein Drawable Icon an. Da diese Icons zur Verständlichkeit meist nicht ausreichen, besitzt jeder Button noch eine Unterschrift.

Um eine bessere Orientierung in der App zu gewährleisten, ist der Button farblich hervorgehoben, der zur angezeigten Activity gehört.



Abbildung 4 Custom Navigation

Die Navigationsleiste ist in jeder Activity vorhanden, um schnell und einfach zu navigieren.

```
<LinearLayout
    android:id="@+id/linearlayout"
    android:layout_width="0dp"
    android:layout_height="60dp"
    android:background="#FFFFFF"
    android:orientation="horizontal"
    android:padding="10dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent">

    <Button
        android:id="@+id/bn_home"
        style=' style="?android:attr/buttonBarButtonStyle"'
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="#FFFFFF"
        android:backgroundTint="#FFFFFF"
        android:drawableTop="@drawable/ic_home_black_24dp"
        android:drawableTint="#FC8B56"
        android:onClick="bn_home_click"
        android:text="Home"
        android:textAllCaps="false"
        android:textColor="#FC8B56"
        android:textSize="10sp" />

    <Button
        android:id="@+id/bn_create"
        style=' style="?android:attr/buttonBarButtonStyle"'
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="@color/colorPrimary"
        android:backgroundTint="#FFFFFF"
        android:drawableTop="@drawable/ic_add_black_24dp"
        android:drawableTint="@color/colorTextDark"
        android:onClick="bn_create_click"
        android:text="Neues Rezept"
        android:textAllCaps="false"
        android:textColor="@color/colorTextDark"
        android:textSize="10sp" />

    <Button
        android:id="@+id/bn_list"
        style=' style="?android:attr/buttonBarButtonStyle"'
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="@color/colorPrimary"
        android:backgroundTint="#FFFFFF"
        android:drawableTop="@drawable/ic_book_black_24dp"
        android:drawableTint="@color/colorTextDark"
        android:onClick="bn_list_click"
        android:text="Meine Rezepte"
        android:textAllCaps="false"
        android:textColor="@color/colorTextDark"
        android:textSize="10sp" />
</LinearLayout>
```

Abbildung 5 Custom Navigation XML Code

Der Java-Code der Button wird im folgenden Beispiel noch einmal näher erklärt. Alle drei Button besitzen eine Methode mit der sie eine neue Activity aufrufen können. Um eine neue Activity aufzurufen, wird ein neues Intent erzeugt. Diesem Intent wird ein package Context übergeben, sowie die Java-Klasse, die als Activity aufgerufen werden soll. Mit der Methode `startActivity(intent)` wird eine neue Activity, mit den übergebenen Parametern, gestartet.

```
//Custom navigation
//placed at the bottom
//3 button methods to switch between the activities
public void bn_list_click(View view) {
    Intent intent = new Intent( packageContext: this, RecipeListActivity.class);
    startActivity(intent);
}

public void bn_create_click(View view) {
    Intent intent = new Intent( packageContext: this, AddRecipeActivity.class);
    startActivity(intent);
}

public void bn_home_click(View view) {
    /*Intent intent = new Intent(this, MainActivity.class);
    startActivity(intent);*/
}
```

Abbildung 6 JavaNavigationsbutton Methoden

4.3 Add Recipe Activity

In der Add Recipe Activity kann der Anwender ein neues Rezept einfügen. Dazu gibt es verschiedene Textfelder, welche bearbeitet werden können und einen Spinner, zur Auswahl eines bestimmten Wertes. Das Layout sieht folgendermaßen aus:

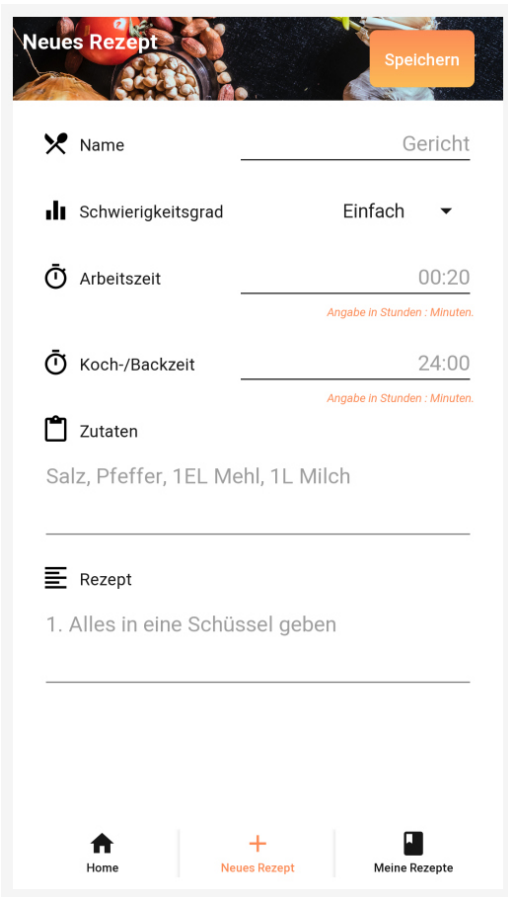


Abbildung 7 Layout Add RecipeActivity

Die ActionBar wurde mit folgenden Befehl versteckt:

```
getSupportActionBar().hide(); // hide the title bar
```

Dadurch konnte eine eigene ActionBar angelegt werden. Diese besitzt ein Hintergrundbild, welches mit imageView eingefügt wurde. Des Weiteren gibt es eine Überschrift „Neues Rezept“, welche noch einmal die aktuelle Activity View hervorheben soll. Der Speichern-Button befindet sich auch oben rechts in der neuen ActionBar. Dieser Button wird die eingegebenen Werte in die Datenbank einfügen.

In das erste Textfeld soll der Name des Rezepts eingegeben werden. Das zweite UI-Element ist ein Spinner mit diesem können die drei Schwierigkeitszustände ausgewählt werden: Einfach, Mittel, und Schwer. Diese Spinner-Werte können zum einen über ein Value Ressource File übergeben werden.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="headings">
        <item>Einfach</item>
        <item>Mittel</item>
        <item>Schwer</item>
    </string-array>
</resources>
```

Abbildung 8 SpinnerArray XML.Datei

Oder aber auch in Java über eine ArrayList.

```
difficultyList = new ArrayList<String>();
difficultyList.add("Einfach");
difficultyList.add("Mittel");
difficultyList.add("Schwer");

int indexDifficulty = spinner_difficulty.getSelectedItemPosition();
difficulty = difficultyList.get(indexDifficulty);
```

Abbildung 9 SpinnerArrayList

Auf der linken Seite werden die Werte, welche wiedergegeben werden sollen, zu der ArrayList hinzugefügt. Auf der rechten Seite werden erst die Positionen vom Spinner geholt und dann die Werte der ArrayList 'hinzugefügt'.

Als nächstes soll der Anwender die Arbeitszeit angeben und dann die Koch-/ Backzeit angeben. Durch die kleinen Tipps, orange gefärbt, wird ein Schema vorgeschlagen, was genutzt werden soll.

Die Zutaten- und Rezept-Textfelder sind multiline Textfelder, die in einer scrollView sind. Dadurch kann das Textfeld aus platzgründen klein bleiben, aber durch die scrollView kann durch das Textfeld navigiert (gescrollt) werden.

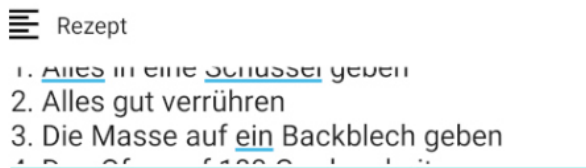


Abbildung 10 Multiline TextView mit ScrollView

Am Ende der Activity ist die Navigationsleiste zu sehen.

Zunächst werden die UI-Elemente per Id in Java implementiert und Variablen zugewiesen.

```
private EditText t_name = null;
private Spinner spinner_difficulty = null;
private EditText t_worktime = null;
private EditText t_cookingtime = null;
private EditText t_ingredients = null;
private EditText t_recipe = null;

t_name = findViewById(R.id.t_name);
spinner_difficulty = findViewById(R.id.spinner_difficulty);
t_worktime = findViewById(R.id.t_worktime);
t_cookingtime = findViewById(R.id.t_cookingtime);
t_ingredients = findViewById(R.id.t_ingredients);
t_recipe = findViewById(R.id.t_recipe);
```

Abbildung 11 UI-Elemente per ID holen

Die Methode checkInputs() überprüft, dass in allen Textfeldern ein Wert steht. Sollte das nicht der Fall sein, wird ein Toast mit einer Fehlermeldung gezeigt und es kann nicht fortgefahren werden. Hier könnten natürlich noch mehr Regel beschrieben werden. Zum Beispiel könnte ein Schema für das Einfügen der Arbeitszeit und Koch-/Backzeit mit einer Regel festgelegt werden. Wenn alle Bedingungen übereinstimmen, wird ein neues Objekt „Rezept“ mit den Parametern wiedergegeben.

```

private Recipe checkInputs(){
    String difficulty = "";

    String name = t_name.getText().toString();
    if (name.length()==0){
        Basis.showToast( context: this, text: "Der Name muss mindestens einen Buchstaben haben.");
        return null;
    }

    String worktime = t_worktime.getText().toString();
    if (worktime.length()==0){
        Basis.showToast( context: this, text: "Eine Zeit muss angegeben werden.");
        return null;
    }

    String cookingtime = t_cookingtime.getText().toString();
    if (cookingtime.length()==0){
        Basis.showToast( context: this, text: "Eine Zeit muss angegeben werden.");
        return null;
    }

    String ingredients = t_ingredients.getText().toString();
    if (ingredients.length()==0){
        Basis.showToast( context: this, text: "Es müssen Zutaten angegeben werden.");
        return null;
    }

    String recipe = t_recipe.getText().toString();
    if (recipe.length()==0){
        Basis.showToast( context: this, text: "Das Zubereitungsrezept muss angegeben werden.");
        return null;
    }

    int indexDifficulty = spinner_difficulty.getSelectedItemPosition();
    difficulty = difficultyList.get(indexDifficulty);

    return new Recipe(name, difficulty, worktime, cookingtime, ingredients, recipe);
}

```

Abbildung 12 checkInputs() Methode Java

Der Speichern-Button besitzt die Methode `bn_insert_click(View view)`. Mit dieser Methode wird überprüft ob das Objekt „Rezept“ der `checkInputs()` Methode entspricht. Wenn das Objekt nicht null ist wird es mit der Methode `insertRecipe(recipe)` in die Datenbank eingefügt. Sollte dies nicht der Fall sein, wird ein Toast mit Fehlermeldung angezeigt. Die Methode `insertRecipe(recipe)` kommt aus der Java-Klasse `DBSQueries`, welche noch näher erläutert wird.

```

public void bn_insert_click(View view) {
    Recipe recipe = checkInputs();
    if (recipe!=null){
        long rowid = DBSQueries.insertRecipe(recipe);

        if (rowid<1){
            Basis.showToast( context: this, text: "Datensatz konnte nicht eingefügt werden: "+rowid);
        }else {
            Basis.showToast( context: this, text: "RowId: "+rowid);
        }
    }
}
}

```

Abbildung 13 Speichern-ButtonMethode fügt Daten in Datenbank

4.4 Recipe List Activity

Auch bei dieser Activity wurde die ActionBar versteckt.

```

getSupportActionBar().hide(); // hide the title bar

```

Der Hintergrund dieser Activity ist ein Farbverlauf. Dieser kann mit einem Drawable Ressource File erstellt werden und wird dann dem listViewLayout in der XML Datei zugewiesen.

```

<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <!-- Gradient Bg -->
    <gradient
        android:startColor="#FC8B56"
        android:endColor="#FCBB55"
        android:angle="270" />
    </shape>

```

Abbildung 14 Gradient per XML Code

Das Layout der Activity sieht folgendermaßen aus:



Abbildung 15 Layout RecipeList Activity

Diese Activity besteht aus einem listViewLayout. In diesem ist zum einen die Überschrift „Meine Rezepte“ zu finden. Dann folgt die ListView. Dort werden die Listeneinträge später dargestellt. Am Ende der Activity befindet sich die Navigationsleiste.

Diese View verbirgt aber noch eine ScrollView. Sollte nämlich eine Zelle in der ListView angeklickt werden, kommt eine ScrollView, mit allen weiteren Informationen zum Vorschein.

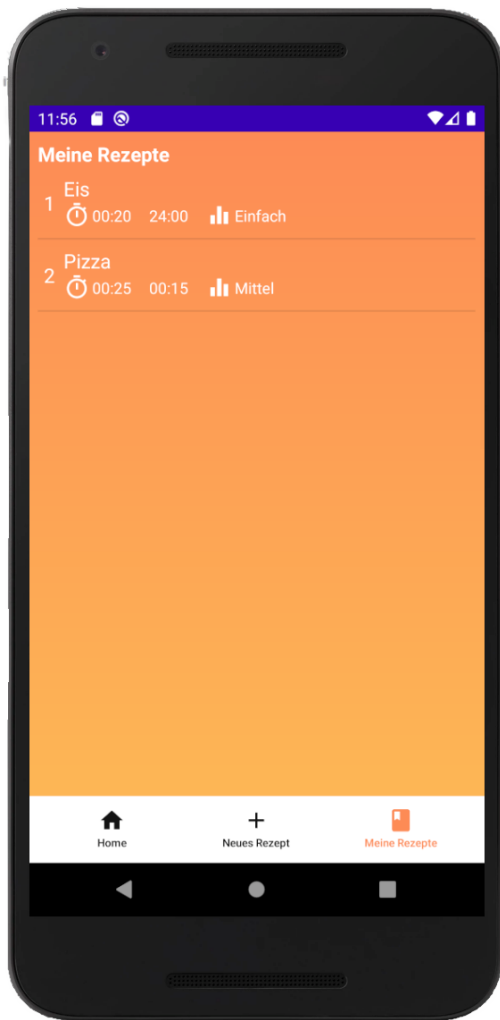


Abbildung 16 Sichtbarkeit von mehr Informationen

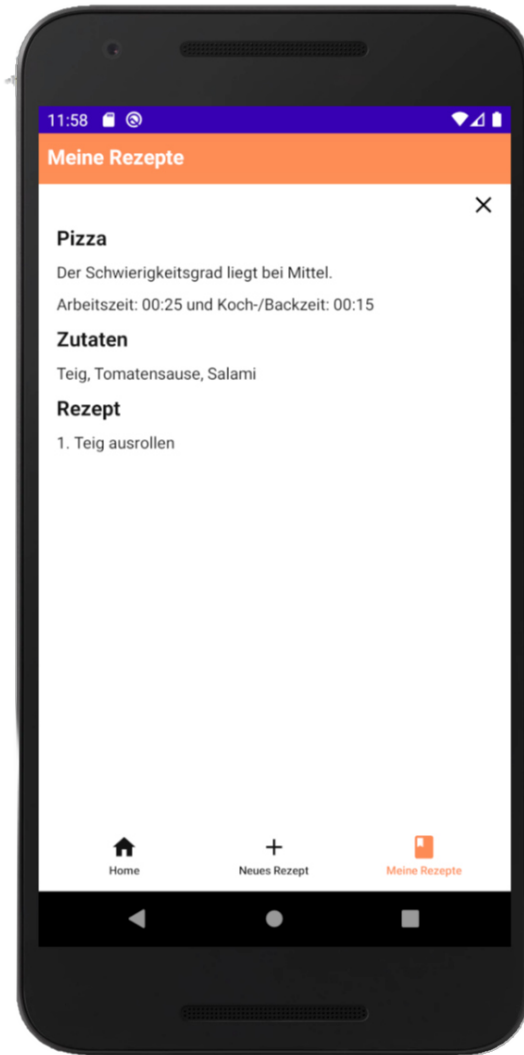


Abbildung 17 ListView

Um die Liste anzeigen zu können, wird ein CursorAdapter benötigt. Dieser wird in die Java-Klasse „RecipeListActivity“ importiert. Zudem muss die ListView per Id in die Java-Klasse übertragen werden.

Mit der Methode insertAdapter() wird der Adapter eingefügt. Der Cursor wird aus der Klasse DBSQueries mit der Methode getRecipeCursor() geholt. Der Adapter „MyCursorAdapter“ wird der ListView mit der Methode setAdapter zugeordnet. Beim Erstellen der View wird der Adapter der ListView zugewiesen.

```

private MyCursorAdapter dataAdapter;
listView = findViewById(R.id.listview);
insertAdapter();

private void insertAdapter(){

    Cursor cursor = DBSQueries.getRecipeCursor();
    dataAdapter = new MyCursorAdapter(
        context: this, cursor, autoRequery: false
    );
    listView.setAdapter(dataAdapter);
}

```

Abbildung 18 insertAdapter() Methode

Die Adapter-Klasse wird später noch näher erklärt.

4.4.1 Zellen Layout

Das Zellen Layout ist relativ komplex aufgebaut (siehe Abbildung). Links steht die Zeilen ID, danach folgt der Name des Rezepts. Unter dem Namen ist ein Icon platziert und nach dem Icon kommt die Arbeitszeit, gefolgt von der Koch-/Backzeit. Es folgt ein weiteres Icon und nach diesem der Schwierigkeitsgrad.

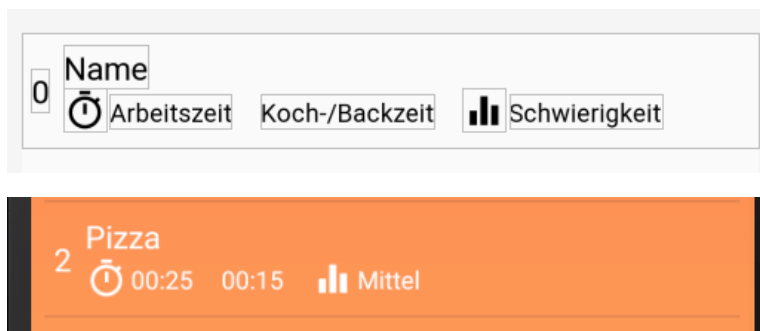


Abbildung 19 Zellen Layout

Die ListView besitzt die Methode „setOnItemClickListener“. In dieser Methode wird zum einen die Position des aktuellen Cursors geholt. Die ScrollView wird auf sichtbar gesetzt und die Methode „showRecipeToast(Cursor cursor)“ aufgerufen. Diese Methode zeigt einen Toast mit dem aktuellen Namen des Rezepts an und sucht die TextViews per Id, um diesen dann einen Wert aus der Datenbank zuzuteilen.

```

private void showRecipeToast(Cursor cursor) {
    Basis.showToast( context: this, cursor.getString(cursor.getColumnIndexOrThrow(DBSQueries.NAME)));

    TextView tvname = findViewById(R.id.tvnamedetail);
    TextView tvdifficulty = findViewById(R.id.tvdifficultydetail);
    TextView tvtime = findViewById(R.id.tvtimedetail);
    TextView tvingredients = findViewById(R.id.tvingredientsdetail);
    TextView tvrecipe = findViewById(R.id.tvrecipedetail);

    String name = cursor.getString(cursor.getColumnIndexOrThrow(DBSQueries.NAME));
    String difficulty = "Der Schwierigkeitsgrad liegt bei " + cursor.getString(cursor.getColumnIndexOrThrow(DBSQueries.DIFFICULTY)) + ".";
    String time = "Arbeitszeit: " + cursor.getString(cursor.getColumnIndexOrThrow(DBSQueries.WORKTIME)) + " und Koch-/Backzeit: " + cursor.getString(cursor.getColumnIndexOrThrow(DBSQueries.COOKINGTIME));
    String ingredients = cursor.getString(cursor.getColumnIndexOrThrow(DBSQueries.INGREDIENTS));
    String recipe = cursor.getString(cursor.getColumnIndexOrThrow(DBSQueries.RECIPE));

    tvname.setText((name));
    tvdifficulty.setText((difficulty));
    tvtime.setText((time));
    tvingredients.setText((ingredients));
    tvrecipe.setText((recipe));
}

```

Abbildung 20 showRecipeToast() Methode

Die ScrollView besitzt einen Button. Dieser Button ist ein Toggel-Button, um die ScrollView wieder zuschließen.

```

public void bn_invisible_click(View view) {
    if (scrollView.getVisibility() == View.GONE){
        scrollView.setVisibility(View.VISIBLE);
    }else{
        scrollView.setVisibility(View.GONE);
    }
}

```

Abbildung 21 JavaCode für Toggle-Button

4.4 Recipe Klasse

Die „Recipe“ Klasse ist die Blaupause für das Objekt „Recipe“. Die Klasse beinhaltet die Index, Name, Schwierigkeit, Arbeitszeit, Koch-/Backzeit, Zutaten und Rezept Variablen. Zudem gibt es die Methode toString(), um die Variablen in einen String zu verwandeln.

```
public class Recipe {  
  
    public int pindex = -1;  
    public String name = "";  
    public String difficulty = "";  
    public String worktime = "";  
    public String cookingtime = "";  
    public String ingredients = "";  
    public String recipe = "";  
  
    public Recipe(String name, String difficulty, String worktime, String cookingtime, String ingredients, String recipe){  
        this.name = name;  
        this.difficulty = difficulty;  
        this.worktime = worktime;  
        this.cookingtime = cookingtime;  
        this.ingredients = ingredients;  
        this.recipe = recipe;  
    }  
  
    public Recipe(){  
    }  
  
    @Override  
    public String toString(){  
        return pindex+", "+name+", "+difficulty+", "+worktime+", "+cookingtime+", "+ingredients+", "+recipe;  
    }  
}
```

Abbildung 22 Java Code Recipe Klasse

4.5 MyCursorAdapter Klasse

Die „MyCursorAdapter“ Klasse wird für die ListView benötigt. Ein Adapter wird benötigt, um Daten von einem Cursor für eine ListView verfügbar zu machen. Die „MyCursorAdapter“ Klasse wird von der „CursorAdapter“ Klasse erweitert. Im Konstruktor müssen der Context, der Cursor und der Boolean-Wert autoRequery geholt werden. Die Methode newView(Context context, Cursor cursor, ViewGroup parent) wird überschrieben und gibt das Row-Layout als neues Layout an.

Die Methode bindView(View view, Context context, Cursor cursor) wird ebenfalls überschrieben. Die TextViews werden per Id vom Row-Layout geholt. Dann wird der Inhalt des Cursors geholt, also die einzelnen Zellen pro Zeilen.

```
import android.content.Context;
import android.database.Cursor;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.CursorAdapter;
import android.widget.TextView;

public class MyCursorAdapter extends CursorAdapter {
    public MyCursorAdapter(Context context, Cursor c, boolean autoRequery) {
        super(context, c, autoRequery);
    }

    @Override
    public View newView(Context context, Cursor cursor, ViewGroup parent) {
        return LayoutInflater.from(context).inflate(R.layout.row_layout, parent, attachToRoot false);
    }

    @Override
    public void bindView(View view, Context context, Cursor cursor) {
        TextView tvpindex = view.findViewById(R.id.tvpindex);
        TextView tvname = view.findViewById(R.id.tvname);
        TextView tvworktime = view.findViewById(R.id.tvworktime);
        TextView tvcookingtime = view.findViewById(R.id.tvcookingtime);
        TextView tvdifficulty = view.findViewById(R.id.tvdifficulty);

        // Inhalt des Cursors holen, also die einzelnen Zellen PRO Zeilen
        int id = cursor.getInt(cursor.getColumnIndexOrThrow(DBSQueries.ID));
        String name = cursor.getString(cursor.getColumnIndexOrThrow(DBSQueries.NAME));
        String worktime = cursor.getString(cursor.getColumnIndexOrThrow(DBSQueries.WORKTIME));
        String cookingtime = cursor.getString(cursor.getColumnIndexOrThrow(DBSQueries.COOKINGTIME));
        String difficulty = cursor.getString(cursor.getColumnIndexOrThrow(DBSQueries.DIFFICULTY));

        tvpindex.setText(Integer.toString(id));
        tvname.setText((name));
        tvworktime.setText((worktime));
        tvcookingtime.setText((cookingtime));
        tvdifficulty.setText((difficulty));
    }
}
```

Abbildung 23 Java Code MyCursorAdapter

4.6 DBSQueries Klasse

Die DBSQueries Klasse ist für die Datenbankbefehle und die Werte, sowie die „getRecipeCursor()“ und „insertRecipe(Recipe recipe)“ Methode verantwortlich.

Durch die DBSQueries Klasse werden der Datenbankname und die Datenbankversion festgelegt.

```
private static final String DATABASE_NAME = "database1.db";
private static final int DATABASE_VERSION = 2;
```

Mit der Methode „getCreateTables()“ wird der Befehl zurückgegeben der die Tabelle in der Datenbank erstellt. Der Tabellename und die Tabellenwerte wurden in statischen und finalen Variablen ausgelagert. Dadurch ist die Übersichtlichkeit besser gegeben.

```
public static String getCreateTables() {
    String s = "CREATE TABLE if not exists "+TABLE1+" (" +
        ID+" INTEGER PRIMARY KEY AUTOINCREMENT, " +
        NAME+" VARCHAR(30) DEFAULT '', " +
        DIFFICULTY+" VARCHAR(30) DEFAULT '', " +
        WORKTIME+" VARCHAR(6) DEFAULT '', " +
        COOKINGTIME+" VARCHAR(6) DEFAULT '', " +
        INGREDIENTS+" VARCHAR(400) DEFAULT '', " +
        RECIPE+" VARCHAR(1500) DEFAULT '' " +
        " );";
    Log.i(TAG, msg: "*****\n"+s);
    return s;
}
```

Abbildung 24 getCreateTables() Methode

In der Methode „insertRecipe(Recipe recipe)“ wird eine neue Instance vom DBSOpenHelper geholt und in der dbsOpenHelper Variablen gespeichert. Des Weiteren werden die Eingabewerte aus den UI-Elementen geholt und in die Datenbank, mit der Methode insertSQL(values, TABLE1) eingefügt.

```
public static long insertRecipe(Recipe recipe) {
    DBSOpenHelper dbsOpenHelper = DBSOpenHelper.getInstance();
    ContentValues values = new ContentValues();
    values.put(NAME, recipe.name);
    values.put(DIFFICULTY, recipe.difficulty);
    values.put(WORKTIME, recipe.worktime);
    values.put(COOKINGTIME, recipe.cookingtime);
    values.put(INGREDIENTS, recipe.ingredients);
    values.put(RECIPE, recipe.recipe);
    return dbsOpenHelper.insertSQL(values, TABLE1);
}
```

Abbildung 25 insertRecipe() Methode

Mit der Methode „getRecipeCursor()“ kann man sich die Werte des Cursors holen.

```

public static Cursor getRecipeCursor()    {
    String[] tableColumns = {ID, NAME, DIFFICULTY, WORKTIME, COOKINGTIME, INGREDIENTS, RECIPE};
    DBSOpenHelper dbsOpenHelper = DBSOpenHelper.getInstance();
    Cursor cursor = dbsOpenHelper.getQuery(TABLE1, tableColumns, whereClause: null, orderBy: null);
    if (cursor==null) {
        Log.e(TAG, msg: "Der cursor (getRecipeCursor) ist null");
        return null;
    }
    else {
        int count = cursor.getCount();
        Log.e(TAG, msg: "(getRecipeCursor) Anzahl der Zeilen: "+count);
        return cursor;
    }
}

```

Abbildung 26 getRecipeCursor() Methode

Als Erweiterung des Projekts könnten die Methoden quoted(), updateSQL() und deleteSQL() in das Projekt integriert werden. Dann könnten Daten (Rezepte) bearbeitet oder auch gelöscht werden.

4.7 DBSOpenHelper Klasse

Die DBSOpenHelper Klasse wird von der SQLiteOpenHelper erweitert. Die DBSOpenHelper Klasse ist eine Helfer-Klasse, die dabei hilft die SQLite Datenbank zum Laufen zu bringen und verwaltet die Datenbankerstellung und das Versionsmanagement. Mit den Methoden „onCreate(SQLiteDatabase)“ und „onUpgrade(SQLiteDatabase, int, int)“ kann die Klasse eine neue Datenbank erstellen, wenn diese noch nicht existiert. Zudem kann diese Klasse mit der „onUpgrade“ Methode die Datenbank upgraden, wenn es nötig ist. Transaktionen werden verwendet, um sicherzustellen, dass sich die Datenbank immer in einem vernünftigen Zustand befindet.

```

@Override
public void onCreate(SQLiteDatabase db) {
    String sql= DBSQueries.getCreateTables();
    Log.i(TAG, msg: "***** String getCreateTables");
    try {
        db.execSQL(sql);
        Log.i(TAG, msg: "***** database are created (1)");
    } catch (SQLException e) {
        Log.i(TAG, msg: "***** no database are created", e);
    } finally {
        Log.i(TAG, msg: "***** in finally onCreate");
    }
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    Log.i(TAG, msg: "Upgrade of the database from version "
        + oldVersion + " to "
        + newVersion + "; all datas are deleted");
    String sql= DBSQueries.getCreateTables();
    db.execSQL(sql);
    onCreate(db);
}

```

Abbildung 27 onCreate() und onUpgrade() Methode

5. Zusammenfassung

Die Android-App listet mehrere Rezepte auf, die mit EditText oder Spinner verarbeitet und in eine Datenbank aufgenommen werden. Durch Toasts wird dem Anwender mitgeteilt, ob die Daten nicht korrekt eingetragen wurden. Durch SQLite ist eine Datenbankanbindung gegeben, die die Rezepte speichert.

5.1 Erweiterungen

- Filter (Suche nach Namen, Schwierigkeitsgrad oder Zeit)
- Bearbeiten und Löschen von Datensätzen (Rezepten)
- Einfacheres Löschen von mehreren Einträgen.