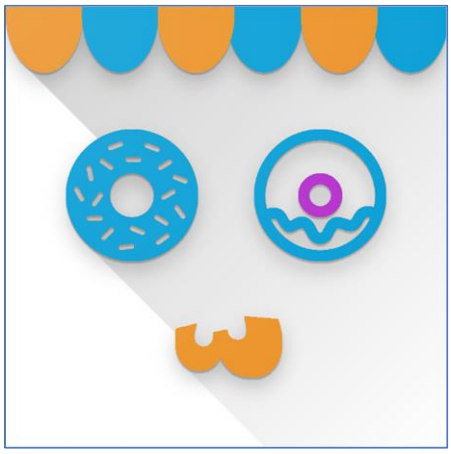
Munchie
Die Rezept-Verwaltung



Programmierprojekt

Vertiefung Mobile Solutions

Name: Leonard Nickels

Matrikelnummer: 5115084

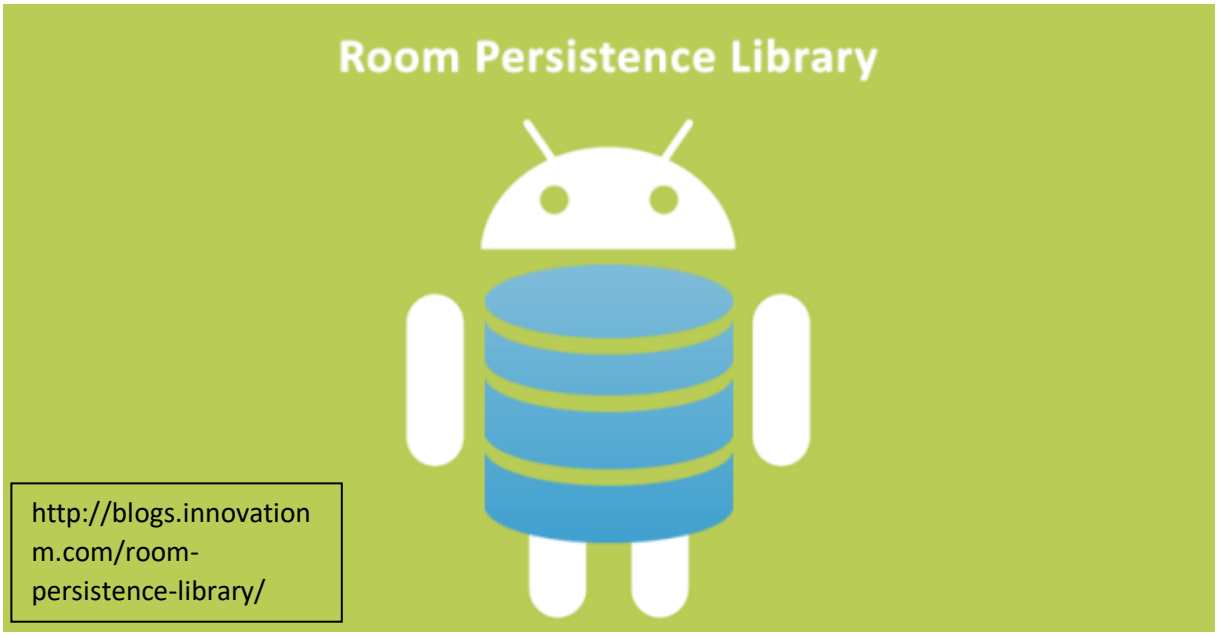
Einleitung

Die App Munchie ist für Android entwickelt und dient zur Verwaltung von Rezepten und Zutaten. Das Ziel ist es, die Errechnung von Rezepten durch die Auswahl eigener Zutatenpreise zu vereinfachen und bei Bedarf leicht ändern zu können. Dabei dient Munchie nicht nur zur Errechnung, sondern auch um Rezepte zu speichern und diese zu teilen. Dabei ist es möglich nicht nur den Preis einer Zutat, sondern auch die Zutat selbst als Produkt zu speichern und schnell Online zu bestellen.

Die Grundidee von Munchie entstammt aus dem Problem Motivtorten für Kunden zu backen. Die immer wiederkehrende Rechnung neuer Zusammensetzungen und Maßeinheiten ist viel Aufwand und soll durch eine handliche App gelöst werden. So ist der Aufwand nur noch Anfangs groß und Veränderungen können leicht eingepflegt werden.

Architektur

Die Architektur von Munchie besteht aus zwei Komponenten, einmal Android als Zielplattform und der Room Persistence Library.



Die Room Persistence Library ist eine Datenbank, die ausschließlich lokal auf dem Endgerät läuft, sie bietet eine Abstraktionsschicht über SQLite und ist leicht zu implementieren, hat aber trotzdem den vollen Umfang von SQLite. Die Room Persistence Library wurde gewählt, da für eine Speicherung der Rezepte und Zutaten eine kleine Datenbank benötigt wird, sie vereinfacht das Speichern der Daten auf dem Android Smartphone und ist ohne Internet Verbindung verwendbar. (mehr Infos: Room Persistence Library)

Das Betriebssystem Android wurde gewählt da der Entwickler, die meiste Erfahrung mit diesem hat. Die Dokumentation in Android ist gut ausgeführt und die Möglichkeiten zu entwickeln sind groß, zudem ist die Entwicklungsumgebung Android Studio sehr hilfreich und unkompliziert bei der Entwicklung von Apps für Android. Die App muss auch ohne Internet-Verbindung aufrufbar sein. Dies alles sind Gründe für eine native Android Applikation.

Projekt: Aufbau

Entwickelt wurde die App in mit Hilfe der Entwicklungsumgebung Android Studio. Die Android-App besteht dabei aus drei Hauptteilen:

- Startseite mit Rezepten und Zutaten (Activity)
- Hinzufügen und Bearbeiten von Rezepten oder Zutaten (Activities)
- Datenbank für Rezept und Zutaten

Eine Activtiy beschreibt in Android eine Klasse, die zu einem Layout zugeordnet wird. Dem Nutzer wird maximal eine Activity gezeigt. Wird eine App bedient, so ist alles was man sieht an eine Activity gebunden. Die Activity besitzt einen eigenen Lebenszyklus, welcher bestimmt was passiert, wenn diese in den Hintergrund rückt (in andere App wechseln), oder wenn man in der App zurück geht.

Die Activities

Die App besitzt drei Activities, die *MainActivity* ist für das Anzeigen aller Rezept und Zutaten zuständig. Die *AddAndEditIngredientActivity* ist für das Bearbeiten und Hinzufügen von Zutaten und die *AddAndEditRecipeActivity* ist für das Bearbeiten und Hinzufügen von Rezepten zuständig.

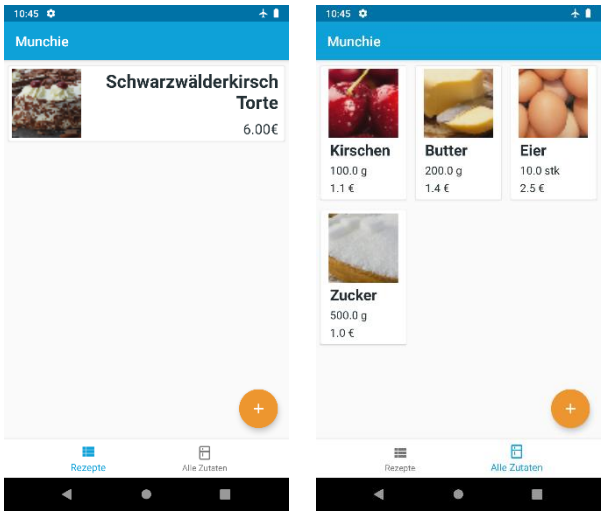


Abbildung 1: MainActivity Rezepte und Zutaten

Die *MainActivity* ist in zwei Tabs aufgeteilt. Klickt man auf eines der Rezepte oder Zutaten oder auf den *FloatingActionButton* mit dem „+“-Symbol unten rechts im jeweiligen Tab, so startet man die Activity zum Bearbeiten oder Erstellen von Rezepten und Zutaten (siehe Abbildung 2).

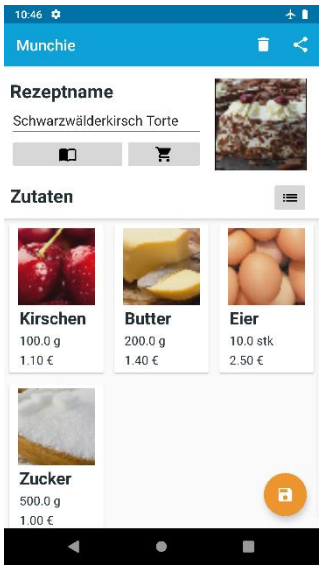


Abbildung 2: Rezept Bearbeiten

Die Klassen

Die Klassen *Recipe* und *Ingredient* beschreiben die Rezepte und Zutaten durch Objektklassen. Sie besitzen alle Attribute wie Name und ID, um diese eindeutig zu beschreiben.

Die Klassen *IngredientAdapter* und *RecipeAdapter* sind für die Listen der Rezepte und Zutaten wichtig. Sie Füllen die *RecylcerViews* mit dem Inhalt der einzelnen Rezepte und Zutaten. Dafür gibt es separat einen Adapter für die Rezepte (*RecipeAdapter*) und einen für die Zutaten (*IngredientAdapter*) da sich die Listenelemente zwischen Zutaten und Rezepten unterscheiden. Zudem definieren die Adapter was passiert, wenn ein Listenelement ausgewählt wird.

Die Klassen *IngredientPriceConverter* und *RecipeHashMapConverter* sind wichtig für die Umwandlung der Preisdaten einer Zutat und die Liste aller Zutaten eines Rezeptes in Form einer *LinkedHashMap* aus der Datenbank. Die Datenbank selbst speichert diese in einem String-Format ein und die Klassen *Ingredient* und *Recipe* besitzen komplexere Attribute, welche die Room Datenbank nicht einfach speichern kann.

Die Klasse *MediaLoader* ist dafür da die Bilder, welche durch Munchie aufgenommen wurden und im Android Gerät gespeichert wurden, zu laden.

Die Klasse *RecipeDatabase* ist zum Erstellen der Datenbank mit den verwendeten Klassen und das Interface *DaoAccess* beschreibt alle Interaktionen und Query-Anfragen für die Datenbank in Funktionen.

Die Klasse *Manager* ist für die temporäre Haltung aller Rezepte und Zutaten während die App läuft. Nur über diese Klasse wird mit der Datenbank interagiert.

Projekt: Implementierung

Es folgt die Implementierung einiger der oben genannten Klassen. Es werden nicht alles ausführlich erklärt, da die meisten Klassen selbsterklärend oder in Android bzw. Java Standard sind. Die Beschriebenen Java-Klassen sind in dem Pfad „ RezeptVerwaltung/app/src/main/java/wizardofba/rezeptverwaltung/“. Die Bild , Layout, sowie Textdaten und andere Ressourcen sind im Pfad „ RezeptVerwaltung/app/src/main/res/“ zu finden.

MainActivity:

Die *MainActivity* (siehe Abbildung 1) ist das erste was der Nutzer sieht. Sie ist wie der Name sagt die Haupt-Activity und von hier aus kommt man auf die Rezepte, die Zutaten und die Einstellungen (in Screens ausgeblendet). Das User Interface der *MainActivity* ist *activity_main.xml*, dieses enthält einen *RecyclerView* für die Listen der Rezepte und Zutaten, einen *FloatingActionButton* und eine *BottomNavigationView*, welche zum Steuern zwischen Rezepten und Zutaten verwendet wird.

```
38      @Override
39      protected void onCreate(Bundle savedInstanceState) {
40          super.onCreate(savedInstanceState);
41          setContentView(R.layout.activity_main);
42
43          setAppIcon();
44
45          manager = Manager.getInstance(this);
46
47          recipeAdapter = new RecipeAdapter();
48          ingredientAdapter = new IngredientAdapter();
49
50          recyclerView = (RecyclerView) findViewById(R.id.recycler_view);
51
52          linearLayoutManager = new LinearLayoutManager( context: this);
53          gridLayoutManager = new GridLayoutManager( context: this,  spanCount: 3);
54
55          recyclerView.setLayoutManager(linearLayoutManager);
56          recyclerView.setAdapter(recipeAdapter);
```

Abbildung 3: OnCreate MainActivity

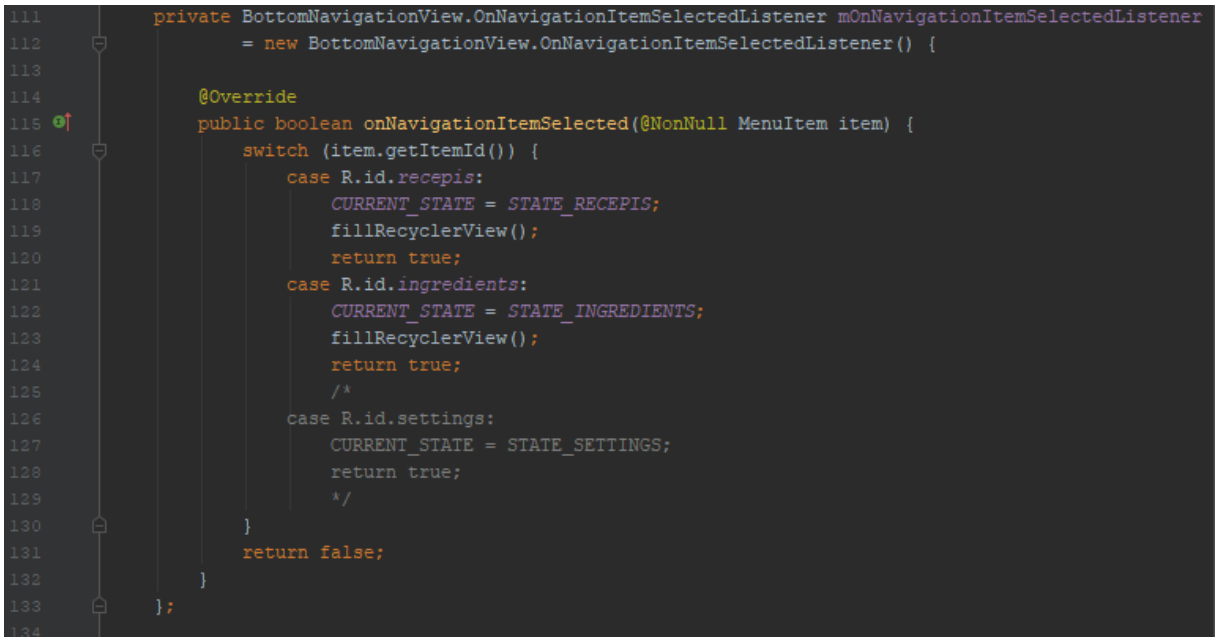
In der *onCreate(…)* Methode, welche als erstes im Activity Lebenszyklus aufgerufen wird, werden nötige Parameter und Variablen gesetzt und mit dem User-Interface verknüpft. In Abbildung 3 ist zu sehen, wie einige Variablen initiiert werden. Das User Interface, also die View wird in Zeile 41 gesetzt. Die Funktion *setAppIcon()* (Zeile 43) setzt das App-Icon in dem App Verlauf des mobilen Endgeräts. Eine Instanz der Klasse *Manager* wird der Variablen *manager* (Zeile 45) zugewiesen. Es handelt sich hierbei um eine Singleton - Klasse, die nur eine laufende Instanz der Klasse zulässt. Die Adapter für die Zutaten und Rezepte werden erstellt (Zeile 47-48). Der *RecyclerView* wird in

Zeile 50 zugeordnet. Die *LayoutManager* für Rezepte (Linear) und Zutaten (Grid) werden erstellt (Zeile 52-53) und folgend der *LayoutManager* und *RecipeAdapter* für Rezepte ausgewählt, da dies der erste Tab ist.

Abbildung 4: MainActivity onCreate, FAB

In Abbildung 4 wird der *FloatingActionButton (FAB)* in Zeile 58 zugeordnet und dann in Zeile 60 der *OnClickListener* für diesen implementiert, welche je nach der Variablen *CURRENT_STATE* eine Aktion ausführt wenn der *FAB* gedrückt wird. Ist die *MainActivity* im *STATE_RECIPES* so wird die *AddAndEditRecipeActivity* gestartet, ist der aktuelle Status *STATE_INGREDIENTS*, so wird die *AddAndEditIngredientActivity* gestartet (Zeile 68-80). Die States werden durch die *BottomNavigationView* (Zeile 84-85) gesetzt und je nachdem, ob man in Rezepte oder Zutaten ist, gewählt (siehe Abbildung 5). Die Methode *fillRecyclerView()* (Zeile 119, 123) setzt den *LayoutManager* und *Adapter* des *RecyclerViews* für den jeweiligen Status.

Wird eine der Activities *AddAndEditRecipeActivity* oder *AddAndEditIngredientActivity* erfolgreich beendet, so werden die Adapter der Zutaten und Rezepte darüber durch die Methode *notifyUpdate()* informiert und diese aktualisieren somit ihre Liste. Die Methode *notifyUpdate()* wird dabei durch *onActivityResult(…)* aufgerufen.



```
111     private BottomNavigationView.OnNavigationItemSelectedListener mOnNavigationItemSelectedListener
112             = new BottomNavigationView.OnNavigationItemSelectedListener() {
113
114         @Override
115         public boolean onNavigationItemSelected(@NonNull MenuItem item) {
116             switch (item.getItemId()) {
117                 case R.id.recepis:
118                     CURRENT_STATE = STATE_RECEPIS;
119                     fillRecyclerView();
120                     return true;
121                 case R.id.ingredients:
122                     CURRENT_STATE = STATE_INGREDIENTS;
123                     fillRecyclerView();
124                     return true;
125                     /*
126                 case R.id.settings:
127                     CURRENT_STATE = STATE_SETTINGS;
128                     return true;
129                     */
130             }
131             return false;
132         }
133     };
```

Abbildung 5: BottomNavigationView SelectedListener

Zuletzt besitzt die *MainAcitivity* eine Funktion die die statische Variabel des Managers zurück gibt (*getManager()*) um auf diese zugreifen zu können.

AddAndEditRecipeActivity und AddAndEditIngredientActivity:

Dieses zwei Activities werden durch die *MainActivity* gestartet und sind ähnlich aufgebaut, sie haben einen ähnlichen Zweck. Die Activities sind dazu da, Zutaten oder Rezepte zu erstellen oder schon erstellte zu bearbeiten. Bei Start wird in der Funktion *onCreate(…)*, wie auch in der *MainActivity*, die UI mit der Klasse verbunden und die entsprechenden Funktionen auf diese zugeordnet. Die Zutaten-Ansicht besteht aus einem *EditText* für den Namen (siehe Abbildung 6), einem *ImageView* für das Bild, ein *EditText* für den Preis und ein *EditText* für die Menge, sowie einem *ImageButton* mit dem Amazon-Logo, einem FAB mit einem Disketten-Symbol zum Speichern der Zutat, einem *Spinner* für die Auswahl der Einheit der Menge und ein Symbol zum Löschen und eines zum Teilen in der oberen Toolbar (Zutat hat kein Teilsymbol). Das Rezept unterscheidet sich hierbei damit, dass es keinen Amazon-Button gibt, aber einen Button für die Beschreibung des Rezeptes und zum Bestellen, sowie eine Liste der enthaltenen Zutaten mit den entsprechenden Mengenangaben. Zusätzlich gibt es noch ein Button mit Listensymbol, um die Zutaten des Rezeptes zu ändern oder andere hinzuzufügen (Wählen aus Liste aller Zutaten).

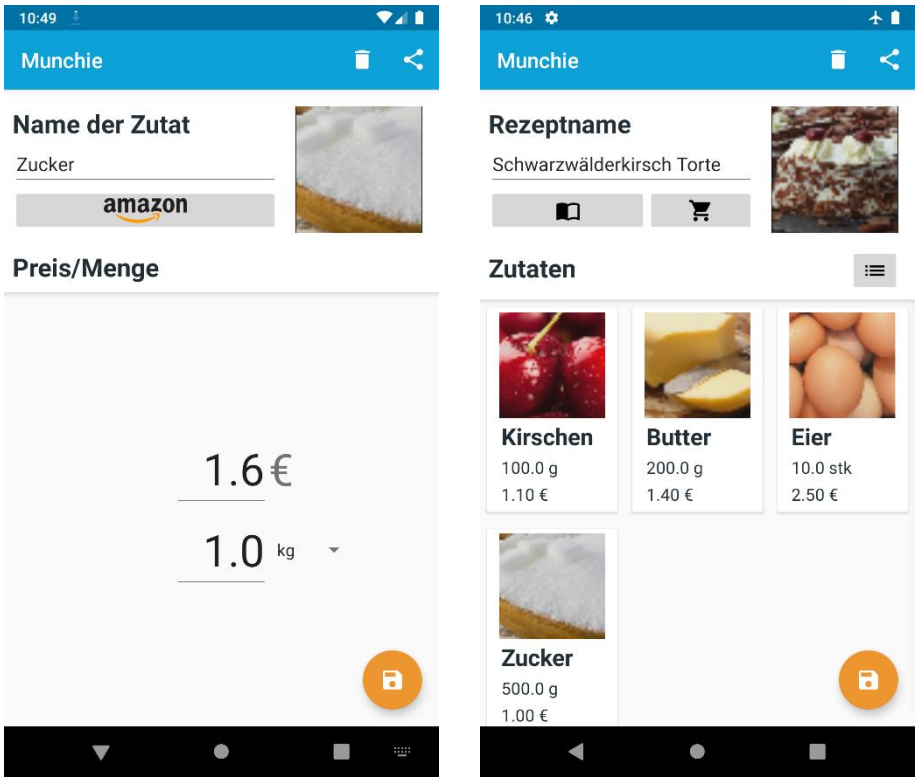


Abbildung 6: AddAndEditRecipeActivity und AddAndEditIngredientActivity

Für die Liste der enthaltenen Zutaten eines Rezepts wird derselbe *IngredientAdapter* wie auch für die *MainActivity* verwendet, nur der Aufruf unterscheidet sich hier. Bei Aufruf der *AddAndEditIngredientActivity* durch die Liste in der *MainActivity*, also mit dem Ziel eine vorhandene Zutat zu betrachten oder zu bearbeiten, wird im *IngredientAdapter* die Information über die ausgewählte Zutat (bei Recipe genauso) der Activity im *Intent* übermittelt und in der *onCreate()* Methode der *AddAndEditIngredientActivity* abgefragt (siehe Abbildung 7 Zeilen 99-131). Die Variable *mIngredient* (Zeile 104, 128) ist hierbei das temporär erzeugte *Ingredient*-Objekt, das von der Activity bis zum Speichern in der Datenbank oder Verlassen der Activity gehalten wird. Die Information, die der Activity bei einer vorhandenen Zutat übermittelt wird, ist die ID der Zutat. Die IDs sind eindeutig und nicht doppelt, es wird hierbei eine UUID beim Erstellen generiert. Durch die ID kann man mit dem *Manager* die gewünschte Zutat aus der Datenbank ziehen (Abbildung 7 Zeile 104). Beim Bearbeiten werden die Views mit den Informationen der geladenen Zutat gefüllt.

```
99          Intent intent = getIntent();
100         String id = intent.getStringExtra( name: "id");
101         if(id != null && !id.equals("")) {
102             CURRENT_STATE = UPDATE_STATE;
103
104             mIngredient = MainActivity.getManager().getIngredientPerUUID(id);
105             nameEditText.setText(mIngredient != null ? mIngredient.getName() : "");
106             amount.setText(mIngredient != null ? mIngredient.getPrice().first.toString() : "");
107             priceEditText.setText(mIngredient != null ? mIngredient.getPrice().second.toString() : "");
108
109             int tempPosition = intent.getIntExtra( name: "position",  defaultValue: -1);
110             Bitmap tempBitmap = MainActivity.getManager().getAllIngredientImgs().get(tempPosition);
111             if(tempBitmap != null) {
112                 imageView.setImageBitmap(tempBitmap);
113             } else {
114                 imageView.setImageDrawable(getDrawable(R.drawable.ic_harvest));
115             }
116             imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
117
118             int i = 0;
119             for(; i < tempStrArray.length; i++) {
120                 if(mIngredient.getUnit().equals(tempStrArray[i])) {
121                     spinner.setSelection(i,  animate: true);
122                     i = tempStrArray.length;
123                 }
124             }
125
126         } else {
127             CURRENT_STATE = NEW_STATE;
128             mIngredient = new Ingredient();
129             description = "";
130             spinner.setSelection( position: 2,  animate: true);
131         }
```

Abbildung 7: onCreate(...) in AddAndEditIngredientActivity

Der FAB zum Speichern der Zutat oder des Rezeptes, zieht sich die Informationen aus den Views der Activity und übergibt diese dem *Manager*, der diese dann neu der Datenbank hinzufügt, oder aktualisiert (siehe Abbildung 8).

```
178                 switch (CURRENT_STATE) {
179
180                     case NEW_STATE:
181                         MainActivity.getManager().addIngredient(mIngredient);
182                         break;
183                     case UPDATE_STATE:
184                         MainActivity.getManager().updateIngredient(mIngredient);
185                         break;
186                 }
```

Abbildung 8: Zutat hinzufügen oder Updaten, FAB OnClickListener (AddAndEditIngredientActivity)

Der *ImageView* ist mit einem *OnClickListener* belegt und will die Kamera-App starten, um einem Rezept oder einer Zutat ein Bild hinzuzufügen. Es wird nach der Erlaubnis des Nutzers für die Kamera und die Datenspeicherung gefragt. Das Bild wird bei erfolgreichem fotografieren und bestätigen in einem Ordner der App auf dem mobilen Endgerät gespeichert, das *Ingredient* oder *Recipe* Objekt hält nur den Pfad dieses Bildes in der Datenbank. Für das Anfragen der Erlaubnis des Nutzers und das Fotografieren werden die Funktionen *onRequestPermissionsResult()*, *onActivityResult()* und

dispatchTakePictureIntent() verwendet. Zudem müssen im *AndroidManifest* die gewünschten Zugriffe eingetragen werden.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.CAMERA"/>
```

Abbildung 9: AndroidManifest Permissions

Die Methode *onRequestPermissionsResult()* wird nachdem der *ImageView* betätigt wurde und die Erlaubnis noch nicht erteilt wurde aufgerufen. Wird die Erlaubnis vom Nutzer erteilt so wird die Methode *dispatchTakePictureIntent()* aufgerufen, welche dann die Kamera-App startet und bei erfolgreichem Aufnehmen und Bestätigen eines Bildes die Methode *onActivityResult()* aufruft. Diese übergibt das Bild dann dem *ImageView* und speichert den Pfad des Bildes in einer Variable.

Das Löschen-Symbol in der Toolbar funktioniert über einen *AlertDialog*, der nachfragt, ob das Rezept oder die Zutat wirklich gelöscht werden soll und bei Bestätigung über den Manager dieses Objekt löscht und die *Activity* schließt.

Der Amazon-Button der Zutaten-Activity öffnet genauso einen *AlertDialog*, aber in diesem wird ein *WebView* angezeigt (siehe Abbildung 10 (3) & (4)), in dem Amazon mit dem eingegebenen Namen der Zutat geladen wird. Die Idee ist es hierbei, ein Produkt der Zutat in Amazon zu finden und dies zu speichern.

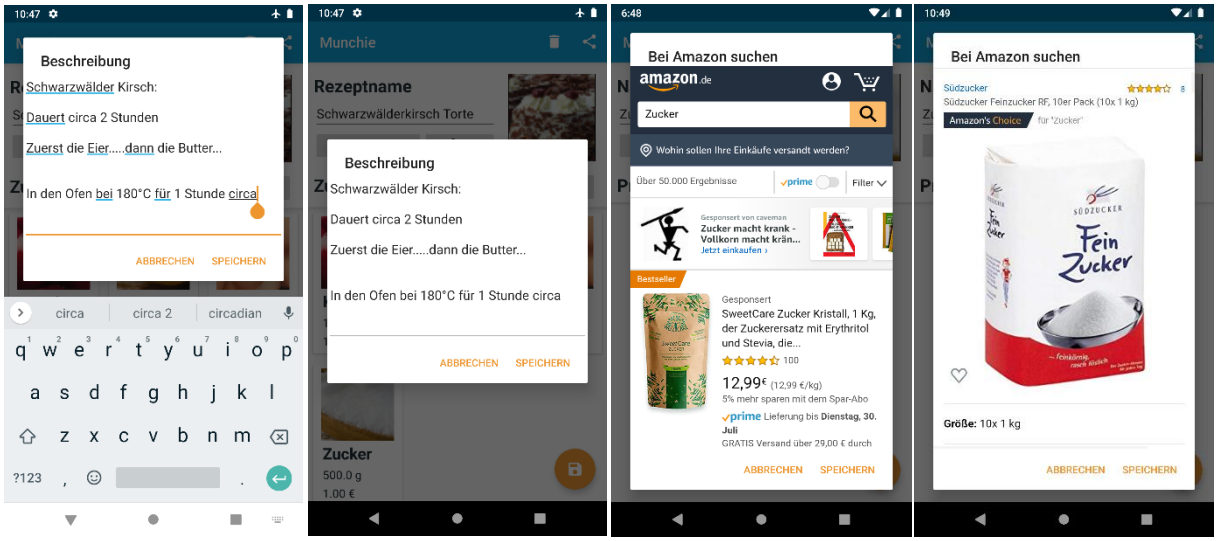


Abbildung 10: (1) & (2) Beschreibung Rezept, (3) & (4) Suchen und Speichern bei Amazon

Der Beschreibungs-Button der Rezept-Activity öffnet auch einen *AlertDialog* mit einem *EditText* für die Beschreibung des Rezeptes (siehe Abbildung 10 (1) & (2)). In Abbildung 12 ist zu sehen, wenn der Listen-Button der Rezept-Activity gedrückt wird. Hier werden auch in einem *AlertDialog* alle Zutaten, welche die App gespeichert hat, in Listenform angezeigt und diese sind durch Multiple-Choice an- und abwählbar.

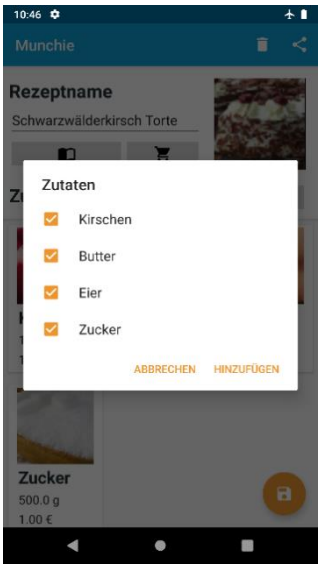


Abbildung 11: Rezept, Zutaten verwalten

Der Einkaufsbutton der Rezept-Activity benutzt die API von Amazon und setzt aus den in den Zutaten gespeicherten Links, eine URL zusammen, welche die Zutaten in einer Liste bereitstellt. Diese Liste schlägt Amazon dann dem Kunden vor, die Zutaten dem Warenkorb hinzuzufügen. Die Liste kann vorher in Amazon bearbeitet werden (siehe Abbildung 11).

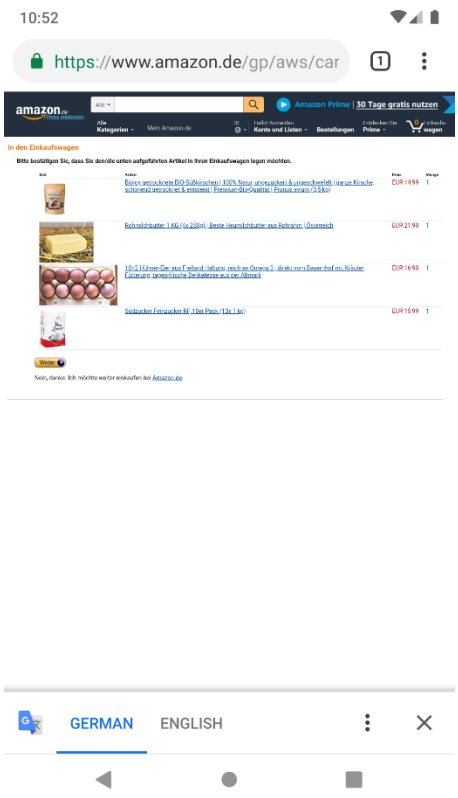


Abbildung 12: Amazon Warenkorb-Liste

Manager:

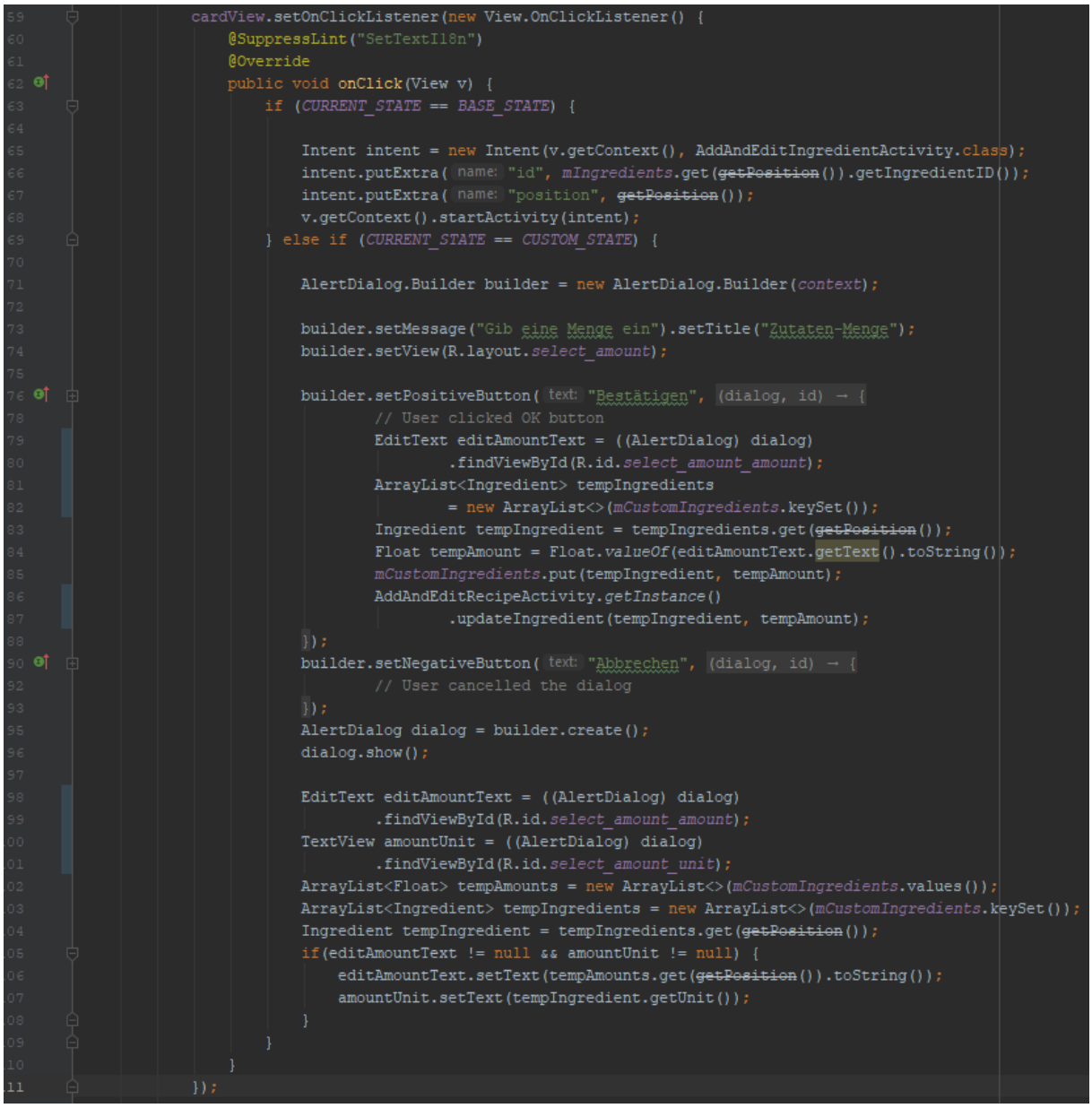
Der *Manager* ist die Verbindung der App zu den gespeicherten Daten der Datenbank (*RecipeDatabase*). Der *Manager* ist ein Singleton, es kann nur eine Instanz existieren. Die Klasse initiiert jeweils Listen für die Zutaten und deren Bilder, sowie Listen für die Rezepte und deren Bilder (siehe Abbildung 13 Zeilen 36-39). Durch den *Manager* können Rezepte/Zutaten hinzugefügt, entfernt oder aktualisiert werden. Hierfür verwendet der Manager eine Instanz der *RecipeDatabase* Klasse. Zudem besitzt die Klasse einige Methoden um bestimmte Zutaten oder Rezepte als Objekt zu erhalten, oder Umwandlungen von Listen durchzuführen.

```
22  public class Manager {
23
24      private static final String DATABASE_NAME = "recipes_db";
25      private static Manager instance;
26
27      private RecipeDatabase recipeDatabase;
28      private List<Recipe> allRecipes;
29      private List<Ingredient> allIngredients;
30      private List<Bitmap> allRecipeImgs;
31      private List<Bitmap> allIngredientImgs;
32      private Context mainActivity;
33
34      private Manager(Context context) {
35
36          allRecipes = new ArrayList<>();
37          allIngredients = new ArrayList<>();
38          allRecipeImgs = new ArrayList<>();
39          allIngredientImgs = new ArrayList<>();
40          mainActivity = context;
41
42          recipeDatabase = Room.databaseBuilder(context,
43                  RecipeDatabase.class, DATABASE_NAME)
44                  .fallbackToDestructiveMigration()
45                  .build();
46
47          loadDatabase();
48      }
```

Abbildung 13: Manager

Ingredient und Recipe Adapter:

Der *Ingredient-* und der *RecipeAdapter* sind dafür da, die Listen der Rezepte oder Zutaten in *CardViews* zu füllen und zu verwalten. Für die Rezepte wird ein *LinearLayout* verwendet und für die Zutaten ein *GridLayout*, da damit gerechnet wird, dass es mehr Zutaten als Rezepte gibt und das *GridLayout* mehr Platz bietet (siehe Abbildung 1).



```
cardView.setOnClickListener(new View.OnClickListener() {
    @SuppressLint("SetTextI18n")
    @Override
    public void onClick(View v) {
        if (CURRENT_STATE == BASE_STATE) {

            Intent intent = new Intent(v.getContext(), AddAndEditIngredientActivity.class);
            intent.putExtra( name: "id", mIngredients.get(getPosition()).getIngredientID());
            intent.putExtra( name: "position", getPosition());
            v.getContext().startActivity(intent);
        } else if (CURRENT_STATE == CUSTOM_STATE) {

            AlertDialog.Builder builder = new AlertDialog.Builder(context);

            builder.setMessage("Gib eine Menge ein").setTitle("Zutaten-Menge");
            builder.setView(R.layout.select_amount);

            builder.setPositiveButton( text: "Bestätigen", (dialog, id) -> {
                    // User clicked OK button
                    EditText editAmountText = ((AlertDialog) dialog)
                            .findViewById(R.id.select_amount_amount);
                    ArrayList<Ingredient> tempIngredients
                            = new ArrayList<>(mCustomIngredients.keySet());
                    Ingredient tempIngredient = tempIngredients.get(getPosition());
                    Float tempAmount = Float.valueOf(editAmountText.getText().toString());
                    mCustomIngredients.put(tempIngredient, tempAmount);
                    AddAndEditRecipeActivity.getInstance()
                            .updateIngredient(tempIngredient, tempAmount);
            });
            builder.setNegativeButton( text: "Abbrechen", (dialog, id) -> {
                    // User cancelled the dialog
            });
            AlertDialog dialog = builder.create();
            dialog.show();

            EditText editAmountText = ((AlertDialog) dialog)
                    .findViewById(R.id.select_amount_amount);
            TextView amountUnit = ((AlertDialog) dialog)
                    .findViewById(R.id.select_amount_unit);
            ArrayList<Float> tempAmounts = new ArrayList<>(mCustomIngredients.values());
            ArrayList<Ingredient> tempIngredients = new ArrayList<>(mCustomIngredients.keySet());
            Ingredient tempIngredient = tempIngredients.get(getPosition());
            if(editAmountText != null && amountUnit != null) {
                editAmountText.setText(tempAmounts.get(getPosition()).toString());
                amountUnit.setText(tempIngredient.getUnit());
            }
        }
    }
});
```

Abbildung 14: IngredientAdapter

Die Adapter sind für Zutaten und Rezepte sind ziemlich ähnlich, weshalb auch hier nur der Zutaten-Adapter angesprochen wird. In Abbildung 14 sieht man den *OnClickListener* des *CardViews* der eine Zutat darstellt. Auch hier hat der Adapter zwei verschiedene Zustände. Einmal der Zustand der neuen Erstellung einer Zutat und dann das Auswählen einer vorhandenen Zutat. Der State wird

beim Erstellen des Adapters durch den Konstruktor mitgegeben. Beim Erstellen einer neuen Zutat, hier in der *MainActivity* bei drücken des *FABs,* wird die *AddAndEditIngredientActivity* gestartet, um eine neue Zutat zu erstellen. Wird eine Zutat in einem Rezept ausgewählt, so öffnet sich ein *AlertDialog*, welcher dem Nutzer ermöglicht die Menge der Zutat nur für das Rezept spezifisch zu ändern. Das ist möglich, da die Klasse *Recipe* eine Liste aller Zutaten mit der Mengenangabe speichert.

```
141         cardView.setOnLongClickListener(new View.OnLongClickListener() {
142             @Override
143             public boolean onLongClick(View v) {
144
145                 if(CURRENT_STATE != BASE_STATE) {
146                     ArrayList<Ingredient> tempIngredients = new ArrayList<>(mCustomIngredients.keySet());
147                     if (tempIngredients.size() > 0) {
148                         ingredientToDelete = tempIngredients.get(getPosition());
149                         AlertDialog.Builder builder = new AlertDialog.Builder(v.getContext());
150                         builder.setMessage("Willst du die Zutat aus dem Rezept nehmen?")
151                                 .setPositiveButton( text: "Ja", dialogClickListener)
152                                 .setNegativeButton( text: "Nein", dialogClickListener).show();
153                     }
154                 }
155                 return false;
156             }
157         });
```

Abbildung 15: IngredientAdapter onLongClick

In Abbildung 15 sieht man den *onLongClickListener* des *CardViews* welcher in Rezepten dafür verwendet wird, diese auf dem Rezept zu entfernen. Der *ViewHolder* hält die *Views* des *UserInterfaces* und ordnet die Daten den Views zu. Für die unterschiedlichen Status gibt es auch unterschiedliche interne Listen, die der Adapter hält.

Ausblick

Die App hat zum jetzigen Stand die Grundfunktionen für eine Rezeptverwaltung. Dennoch gibt es einige Funktionen, die so noch nicht vollständig sind oder einige die geändert werden sollten.

Ein weiterer Schritt wäre der Wechsel von Amazon auf eine Supermarkt-Kette wie Rewe, die in Sachen Lebensmittel viel mehr Auswahl haben und somit das Besorgen der Zutaten erheblich erleichtern würde.

Rezepte können noch nicht anderen Rezept hinzugefügt werden. Ein Beispiel hierfür, ist wenn man ein Rezept für Tortenböden eingespeichert hat und eine Torte diese Tortenböden mehrfach braucht, man es einfach dem Rezept hinzufügt. Diese Funktion ist zurzeit noch nicht vollständig implementiert, aber die Voraussetzungen hierfür sind schon gegeben, nur ist die Frage wie man Rezepte von Zutaten visuell unterscheidet.

Ein weiterer Gedanke, war die Zeit für Rezept einzupflegen, um so Termine besser planen zu können. Soll ein Rezept zu einem gewissen Zeitpunkt fertig sein, so ist es wichtig zu Planen wann man beginnt dieses zuzubereiten. Dies könnte auch in Etappen der verschachtelten Rezepte geschehen um die einzelnen Komponenten, um den Tag herum, flexibler zu gestalten.

Die momentane Teilfunktion der Rezepte, sendet einen einfachen Text mit der Beschreibung, den Zutaten und deren Menge. Hier könnte noch das Foto als ein zu teilendes Objekt hinzugefügt werden, in den heutigen Zeiten mit Instagram sicher eine brauchbare Funktion.

Die Synchronisation mit einem Server und somit der Online Speicherung wäre ein weiteres Feature, welches die App vervollständigen würde. So könnte man unabhängig vom Gerät auf all seine Rezepte und Zutaten zugreifen.

Das importieren und Exportieren von Rezepten und Zutaten (oder Zutatenlisten) für andere Nutzer wäre eine weitere nützliche Funktion. Früher hat man ja auch immer die Rezepte seiner Freunde und Familie abgeschrieben oder kopiert und so sein Sortiment erweitert.

Die Implementierung von Einstellungen, welche in dieser Fassung momentan ausgeblendet sind, würde dem Nutzer mehr Möglichkeiten geben seine App zu individualisieren. Ideen hierfür wären die Farbtonwahl der App, die Schriftgröße und Größe der Listenelemente, oder vielleicht sogar die Auswahl zwischen Verschiedenen Shops zum Einkaufen (REWE, Amazon, etc.)