Informatik
Hauptcampus

HOCHSCHULE TRIER

Foodtruck Finder

Tobias Klevenz

Master-Abschlussarbeit

Betreuer: Prof. Dr.-Ing. Georg J. Schneider

Ort, 27.10.2017

Abstract

This document is part of a master’s thesis, which explores the creation of a mobile application that is used to list entries of locations within a radius of a device, developed for the Android operating system. The application described uses food trucks, which are mobile kitchens with changing locations. It describes how it efficiently receives the location data from an API, and bringing relevant data to the user, focusing on usability by applying recommended design pattern.

Table of contents

Foodtruck Finder ..... **I**

1 Introduction **1**

2 Motivation **2**

3 Related Work **3**

4 Material and Methods **8**
4.1 Android 8
4.1.1 Android Component 8
4.1.2 Using android components 11
4.1.3 Storing application data 11
4.1.4 Android Libraries 11
4.2 App Engine 13
4.2.1 Java Servlet 13
4.2.2 Foodtrucks Deutschland API 13

5 Application Design **15**
5.1 Android App 16
5.1.1 User Interface 16
5.1.2 Requesting Location access 18
5.1.3 Service 18
5.1.4 Data Storage 19
5.2 App Engine Web App 20

6 Implementation **21**
6.1 Initial launch and setup 21
6.2 Location Strategy 22
6.3 Geofences 23
6.4 User Interface 25
6.4.1 WelcomeActivity 25
6.4.2 MainActivity 27
6.4.3 DetailActivity 31
6.4.4 MapActivity 33
6.5 Service 35
6.6 Data Storage 36
6.7 App Engine Web App 38

7 Using Foodtruck Finder **40**

8 User Feedback 44

9 Summary 46

10 Outlook 47

Table of figures

Figure 1 Screenshot of existing Foodtrucks app with highlighted weak points................ 3
Figure 2 Navigating to a food trucks details page.. 4
Figure 3 Search view of existing Foodtrucks app 5
Figure 4 Active and inactive truck location in the Foodtrucks app................................... 6
Figure 5 Lifecycle of an activity in android 9
Figure 6 JSON Syntax in EBNF 14
Figure 7 JSON data providing location information for specific regions 14
Figure 8 Interaction between android app and API... 15
Figure 9 Algorithm used for grouping food trucks list 17
Figure 10 Entity-Relationship diagram for Foodtruck Finder's database........................ 19
Figure 11 Sequence of events when first launching Foodtruck Finder............................ 21
Figure 12 Geofence Events 24
Figure 13 Slides used for introducing a user to Foodtruck Finder.................................. 26
Figure 14 Custom ViewPager class, line 18-20 shows the additional method 26
Figure 15 Layout of MainActivity 27
Figure 16 Views used for RecyclerView 28
Figure 17 ViewHolder for a divider item in MainActivity 29
Figure 18 FoodtruckAdapter 29
Figure 19 Abstract class FoodTruckListItem.. 30
Figure 20 Search result on device 31
Figure 21 Layout of DetailActivity 32
Figure 22 Abstract class used for the items in DetailActivity... 33
Figure 23 Layout of MapActivity 34
Figure 24 Creating a PeriodicTask 35
Figure 25 Java class used to create table.. 37
Figure 26 Java class used to create ConrtentProvider... 37
Figure 27 Sequence of events connecting to API through App Engine........................... 39
Figure 28 Foodtruck Finder showing four active trucks.. 40
Figure 29 Details for selected food truck 41
Figure 30 Integrated map 42
Figure 31 Selecting a custom location in the settings 42
Figure 32 Results user survey, ease of use.. 44
Figure 33 Results user survey, user friendlyness 44
Figure 34 Results user survey, design.. 45

1 Introduction

Street Food was originally described as just a Snack sold on the Street prepared by a mobile Kitchen. Today this snack has grown into a whole Industry that prides itself on providing quality food which they view as their own craft. Away from just being fast food, many providers of Street Food want to provide a gourmet like kitchen on the street that attracts food enthusiasts. In their home country, the USA, the food truck industry has been booming for many years, showing steady growth with a total annual revenue of 1.2 billion dollars in 2015 according to market research published by [MobileCuisine]. While the European market surely is still far away from those numbers, Foodtrucks Deutschland as the biggest food truck platform in europe is as of the writing of this tracking around 600 trucks in Germany, Austria and Switzerland, with plans to expand to further countries in the future.

Foodtrucks Deutschland build their business on providing premium services to food truck owners and providing users a directory of available trucks. In collaboration with Foodtrucks Deutschland I developed an Android App that makes use of their API and informs users about available Foodtrucks in their area.

This project has been realized in close collaboration with Foodtrucks Deutschland, the development of the included android application has influenced the feature set of their API, just as the data their API provides has influenced the resulting android application. As the title of this paper, the android application developed is named Foodtruck Finder, highlighting its core function already in its name, enabling users to find food trucks.

2 Motivation

In 2017 where the penetration of smartphones in Europe and the US is getting close to 70% according to [Newzoo], the importance of smartphones and the popularity of apps does not have to be discussed anymore, by now those are deeply integrated into the daily lives of millions. In fact the market is so saturated that in recent years the industry is talking about so-called App Fatigue, which as nicely described by Ben Schippers at [TechCrunch] is comparable to a gold rush that came to an end.

According to a study by [StatistaApps], there are currently 2.8 million apps available in the Google Play Store which seems to confirm the now infamous quote from Steve Jobs: *“There is an app for that”*, however as a study from may 2017 [StatistaPublishers] show, more than 70% of time spent in apps goes to the apps of big corporations, with Google and Facebook sharing a 50% of the total. Furthermore looking at monthly app downloads [StatistaAppDownloads] found that 49% of smartphone users do not install any apps at all while around 40% of users install between 1 and 4 apps per month. These statistics highlight the difficulty of publishing a mobile application and show that providing value to users is crucial for an app’s success.

As part of this thesis, an android application called Foodtruck Finder, that finds food trucks in a radius to a device’s location, was developed from ground up. The application was implemented using common design patterns by closely following Google’s recommendations for Android development, to deliver a familiar user experience. At its basic level Foodtruck Finder has to fulfill the following requirements:

- Acquire a user's location
- Connect to Foodtrucks Deutschlands API and retrieve a dataset of available food truck locations
- Convert and save the retrieved data on the device
- Display the dataset in a perimeter to the acquired location

3 Related Work

While on the one hand there are lots of applications that deliver information about where to eat to a user, including ones run by big companies like Yelp, Foursquare or even Google Maps with its nearby functionality, those apps assume a fixed location for their listings which a food truck with regularly changing locations does not have. At the same time, when serving a niche audience like food truck enthusiasts, being listed in those apps would make food trucks hard to find between millions of regular restaurants and thus rendering those applications useless as a way of letting users discover food trucks.

On the other hand, some food trucks have their own apps, these, however, fulfill a different use case as they are targeted at a specific truck's customers instead of providing information about multiple available food trucks. Considering the statistics mentioned in the previous section [StatistaAppDownloads], which highlights the reluctance smartphone users have when installing apps, the benefit of having this kind of highly specialized app, that only serve a single purpose can be called into question. Furthermore, the data provided through these apps generally consists of the same information that can be found on a truck's website or social media pages, like food menu, contact information, and schedule, which can be accessed by their customers without having to install any additional apps.

The existing app that uses the API provided by Foodtrucks Deutschland, is simply called Foodtrucks. **Figure 1** shows the app's main user interface and highlights some key points in its implementation.

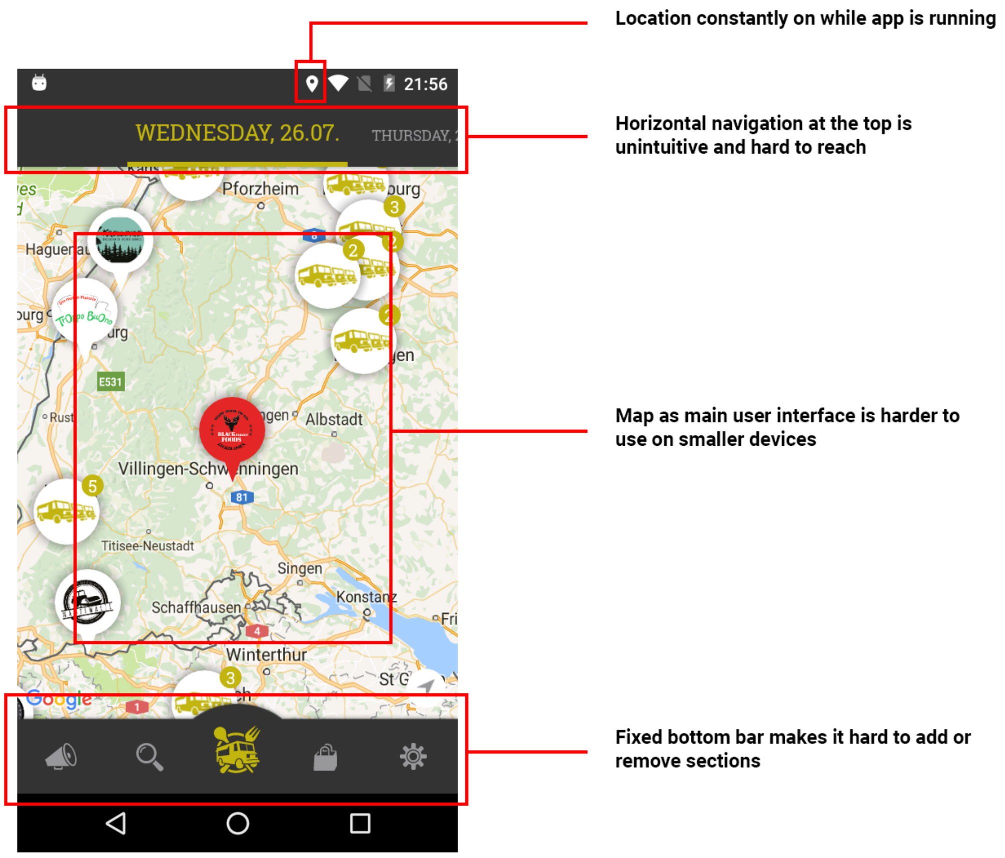

Location constantly on while app is running
Horizontal navigation at the top is unintuitive and hard to reach
Map as main user interface is harder to use on smaller devices
Fixed bottom bar makes it hard to add or remove sections
21:56
WEDNESDAY, 26.07.
THURSDAY,
Pforzheim
Haguenau
Albstadt
Villingen-Schwenningen
Titisee-Neustadt
Singen
Schaffhausen
Konstanz
Winterthur
Google

Figure 1 Screenshot of existing Foodtrucks app with highlighted weak points

Whenever Foodtrucks is running in the foreground, it will constantly poll the device’s location. Since the app does not show any live updates of location data to the user or provide any visible feature that would require constant updates to the user’s location, implementing location access like this is not required. This will drain the device's battery, which is not only bad for the user but also bad for the app itself, as Andrew Ahn, Product Manager for Google Play recently described at [AndroidDevBlog], Google will automatically demote apps in their listings in the Play Store that show this kind of behaviour, and will instead promote high-quality apps which are implemented the right way. Which means the app will not be ranked prominently for related search results in the store, which leads to a lower acquisition of new users. Furthermore, Foodtrucks offers a setting for the users to be notified about nearby food trucks, which when activated will cause the device to permanently poll for location updates, even if the app is not actively running.

Upon launching the app it presents itself to the user with an embedded Google Map, centered on the user’s location. The map is the main user interface with which the user is supposed to find trucks that are shown as markers on the map. While this is helpful for getting an overview of all available food trucks, choosing this as the main interface to get information about a truck involves a lot of steps as the following example will show. To find a truck in a certain area the user has to swipe to that area, zoom into that area either with the pinch to zoom or the double tap gesture, which in many cases has to be executed multiple times in areas where more than one food truck is available. When the marker for the truck the user is looking for is visible he has to tab that marker once, to expand the information window, and again to finally get the information about that truck. As can be seen in **Figure 2**, when having Hamburg set as hometown, it takes a total of 10 steps to get to a specific food trucks information. This issue is amplified on smaller devices where a map as main way of navigation gets even harder to use due to the constraint size of the device.

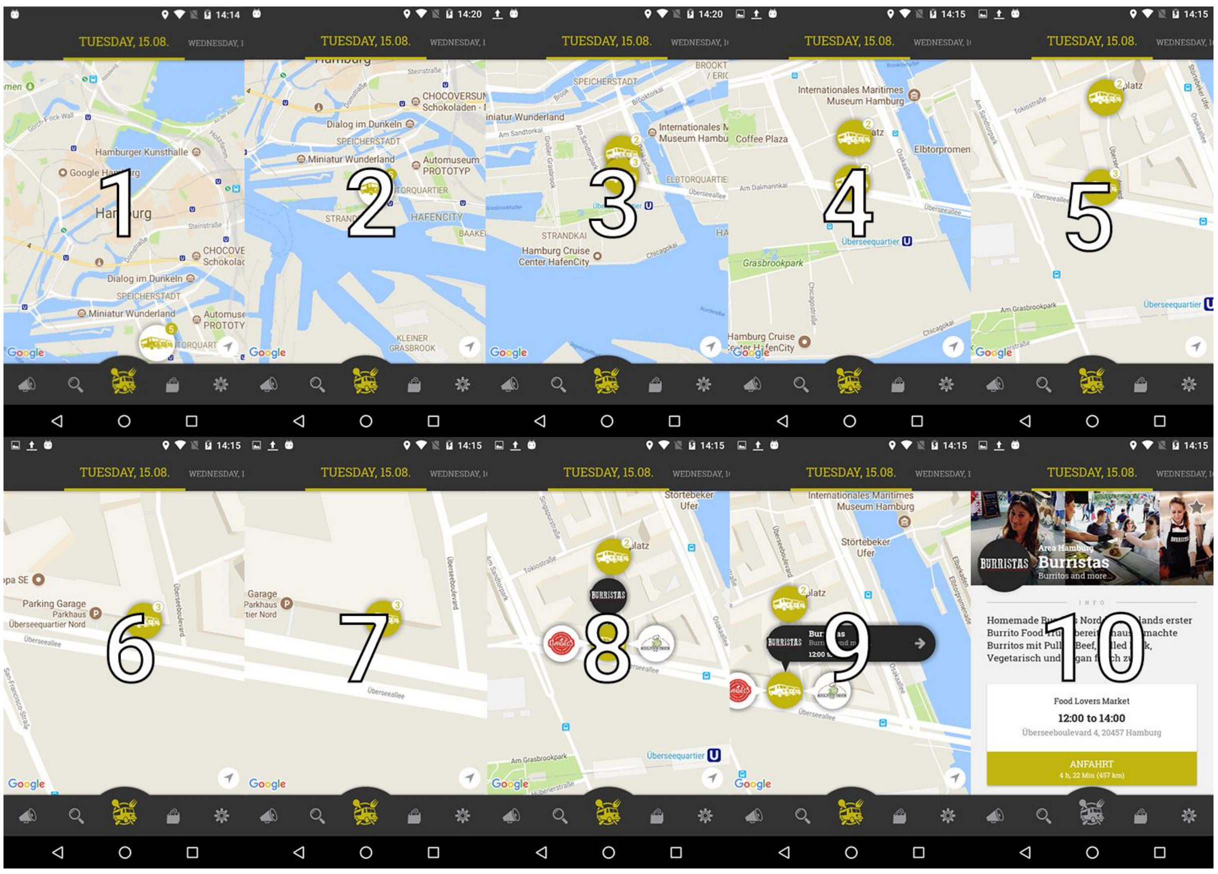
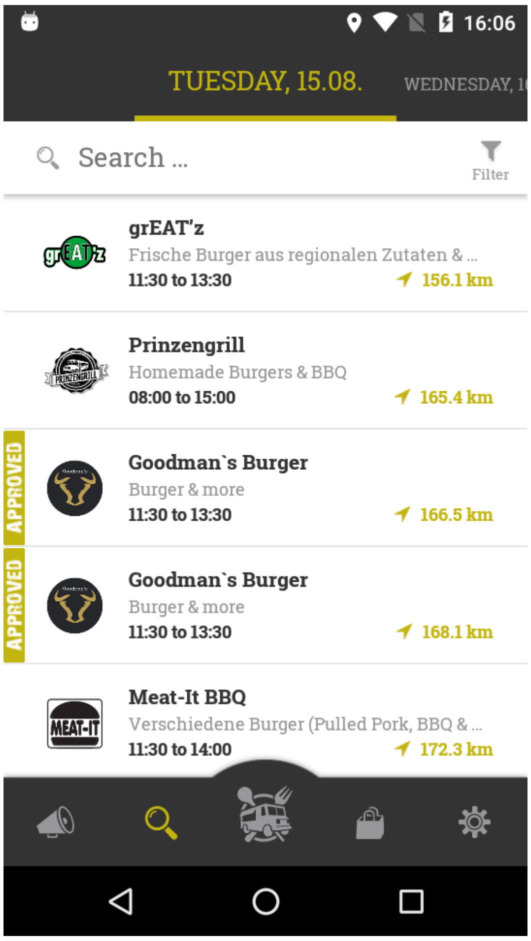


Figure 2 Navigating to a food trucks details page using the map in the existing Foodtrucks app

As a secondary option, the app offers a search function, which by default, meaning no search term is entered, lists the food trucks according to proximity to the device. Although there are some flaws in this view's implementation, for example as can be seen in **Figure 3**, it is 4 pm on the device, however the trucks shown had already closed before that, this is a much better representation of the data set, where a user can get to the information about certain trucks around him with as little as 3 or less interactions such as search, scroll, click. Using a similar view as the main interface for the app, meaning a basic list of items, sorted by the distance to the user, would provide a faster way to deliver the most relevant data to the user.



grEAT'z
Frische Burger aus regionalen Zutaten & ...
11:30 to 13:30 156.1 km

Prinzengrill
Homemade Burgers & BBQ
08:00 to 15:00 165.4 km

Goodman`s Burger
Burger & more
11:30 to 13:30 166.5 km

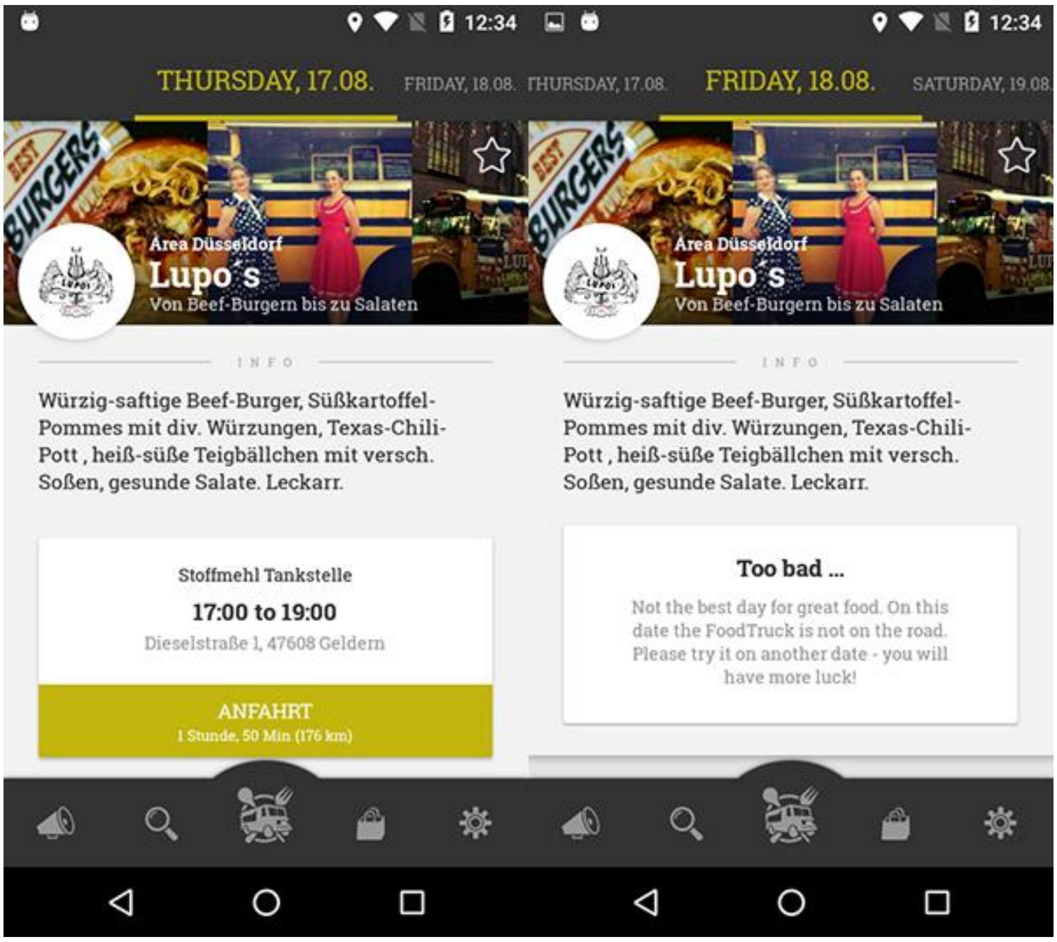
Goodman`s Burger
Burger & more
11:30 to 13:30 168.1 km

Meat-It BBQ
Verschiedene Burger (Pulled Pork, BBQ & ...
11:30 to 14:00 172.3 km

Figure 3 Search view of existing Foodtrucks app

To navigate through the app, Foodtrucks relies on a tab bar at the top, which allows the user to switch between dates, and a navigation bar at the bottom to switch between different sections of the app. The bar at the bottom gives access to a news feed, the search functionality, the main map view, a shopping cart for preorders, and the settings of the app. The developers of the app, highlighted the main view by extending the space above the icon as can be seen in figure 3. However, this implementation limits the bottom bar to a fixed uneven number of items, disregarding the padding between items listed here this design could work for three, five or seven items, even though, depending on the size of the device, using seven items could reduce the clickable area for each item too much. This implementation clearly favors form over function, which is illustrated by the fact that the shopping cart for preorders is still present in the app, even though the preorder functionality has never been fully implemented and currently serves no function at all, selecting the shopping cart icon, will take the user to an empty cart that can not be filled in any way. Using this custom build bottom bar, removing the item from the bottom bar would require redesigning a large portion of the app, while in a standard bottom bar it would've been possible to remove an item without breaking the design. Furthermore the implementation of the bottom bar goes against Google's material design guidelines [MaterialGuidelines], which state that the bottom navigation should be used for *"three to five top-level destinations of similar importance"*, meaning each of the entries listed in a bottom bar, should provide a function that is equally important to the user. This is clearly not the case here, as for example the settings of the application will be visited much less frequently than the map or the search section.

The tab bar at the top is used to switch between days of the ongoing week and is present on the map, the search view and the view showing the food trucks details, however not on any of the other views, which are static and do not change per day, which again shows the difference in importance of the items listed in the bottom bar. The tab bar switches linear to the next day, for the upcoming days across all sections of the app that use the tab bar, for the map and the search results this works quite well, as the results change to the ones of the selected day, however when showing a food trucks information as can be seen in figure 4, if the truck is not active on a day it will just display a message informing the user of its unavailability. If in the worst case, for the ongoing week the truck would not be active apart from today, a user looking for the trucks next date switches from day to day without any changes on the shown page. [MaterialGuidelines] warns about the mixing of tab bars and bottom bars, *"as the combination may cause confusion when navigating an app"*.



Würzig-saftige Beef-Burger, Süßkartoffel-Pommes mit div. Würzungen, Texas-Chili-Pott , heiß-süße Teigbällchen mit versch. Soßen, gesunde Salate. Leckarr.

Not the best day for great food. On this date the FoodTruck is not on the road. Please try it on another date - you will have more luck!

Figure 4 Active and inactive truck location in the Foodtrucks app

As shown, Foodtrucks makes some crucial errors in user interface design and implementation. As stated, using the map as the main interface is questionable, even though the map view itself is well implemented, browsing the map using gestures works fast and fluid, even when many items are shown on the map. When zooming in and out of the map, markers on the map automatically are grouped together at certain zoom levels which makes viewing the map easier. As demonstrated later in this paper, Foodtruck Finder is implemented using a list based main interface, similar to the view showing search results in the existing app, and provides a map similar to this one as a secondary option.

4 Material and Methods

4.1 Android

Android is a Linux based open source operating system currently developed by Google. Initially used for smartphones only, Android today is used by many different devices, such as tablets, TVs, Game Consoles or even cars. Android is the most popular mobile operating system with a market share of 73% according to [StatcounterMobile] and recently, with the growing ubiquity of smartphones, even became the world's most used operating system, including desktop, beating windows with a market share of 42% according to [StatcounterOS].
While Foodtrucks Deutschland’s users are equally split between around 10000 users each on IOS and Android, choosing between developing for Android instead or IOS comes down to personal preference with Android being open source and Google offering its development environment on all platforms.

The main programming language used for developing android apps is Java for which Google provides its own runtime and implementation and does not rely on Oracle’s JDK/JRE. Android supports all Java 7 language features and some Java 8 features depending on the version targeted by the app.

The current Android version as of the writing of this document is 7.1, with version 8 being released late 2017. Android has a long lasting problem of fragmentation, with a lot of devices still using older versions of Android, with about 25% of devices active today using a version that is around 4 years old or even older as can be seen at [AndroidDashboard]. When developing an Android app a developer has to decide on a minimum version that the app should support, with Foodtruckfinder I chose version 4.1 as the last supported version covering 99% of devices today. To bring newer APIs and functionality to older devices Google offers multiple support libraries that developers can use in their applications, to help create an app that behaves and more or less looks the same on all devices, with the exception of some operating system specific features like for example animations or the support for a transparent status bar under which the app can draw.

With Android 5.0 Google introduced a new design language, called material design, with which it released a set of user interface guidelines to promote a consistent user experience across apps. Foodtruckfinder adheres as much as possible to those guidelines and applies recommended design patterns to create a coherent app experience.

4.1.1 Android Component

This section will give a brief overview of the basic structure of Android apps to explain the concepts used later on in this document. As outlined in [AndroidFundamentals], Android apps consist of four types of components, which are activities, services, broadcast receivers, and content providers. Each component type serves its own purpose and follows a certain lifecycle that defines how the component is created and destroyed.

Activity

According to [ProAndroid5, C.1], “an activity is a UI concept that usually represents a single screen in your application” that generally hold one or more user interface components. Activities can be compared to windows in a desktop operating system, on a mobile device, however, most of the time they take over the whole screen. [AndroidDevActivity] also describes an activity as “a single, focused thing that the user can do” meaning it is an application component that generally is meant for the user to interact with.

Every activity that is started, used and exited always goes through the same activity lifecycle which essentially consists of four states. When an activity is actively used it is running, when a new activity gets drawn over the current activity or the user presses the home button, the current activity gets paused as long as it is still visible, when the new activity completely hides the activity or it is not visible anymore after pressing the home button, it is stopped until it is exited and the activity is destroyed, or it is resumed and enters the running state again.

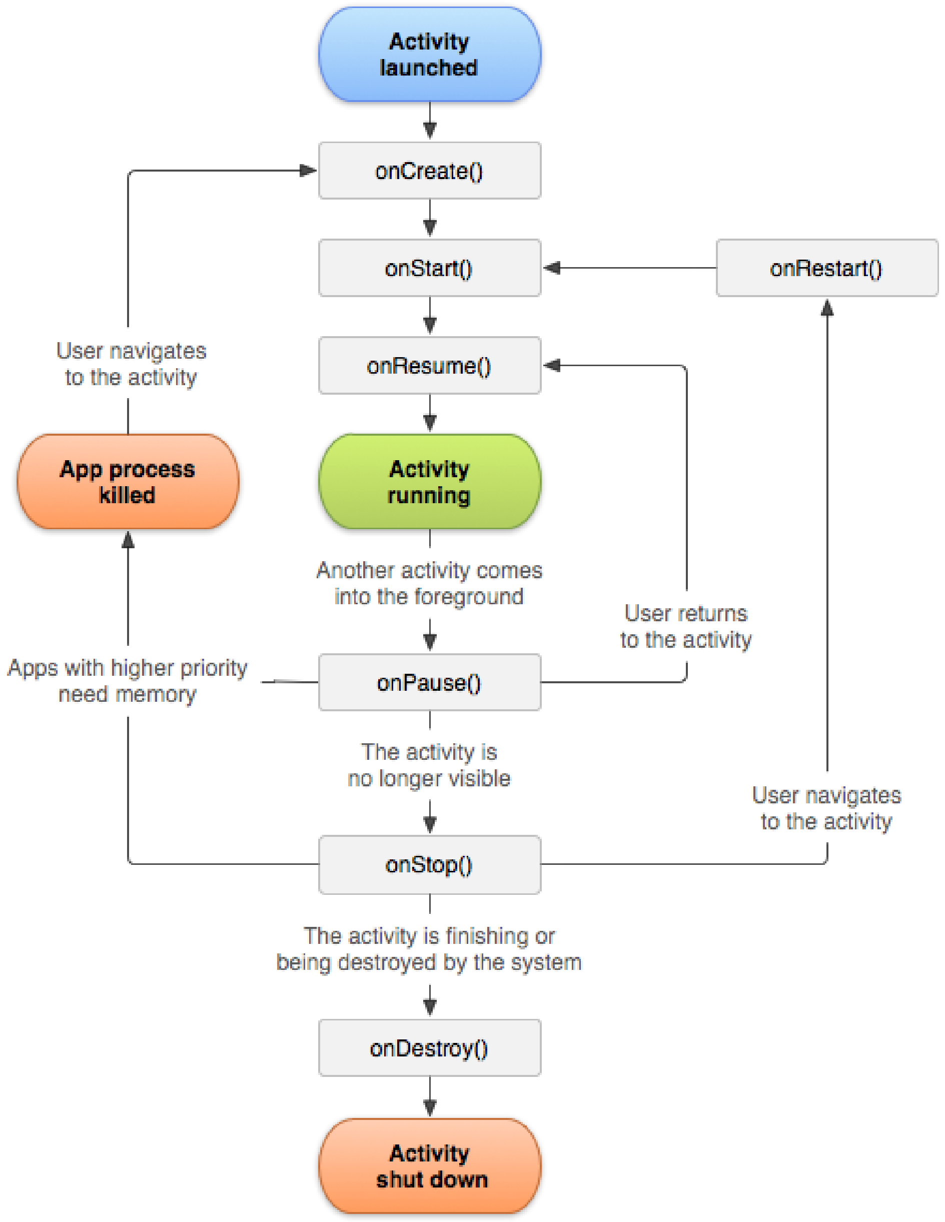

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 5 Lifecycle of an activity in android

To react to the different states inside an app, Android offers callback methods that allow the developer to take action depending on the state change. **Figure 5** shows the states described and the corresponding callback methods. There are some common practices for how to use the callback methods, for example setting up UI components and generally everything that has to be done before the app is visible, should be implemented in `onCreate`, or clean-up work like freeing up used resources or unregistering components should be done in `onDestroy`.

To support more flexible and dynamic layouts, Android makes use of fragments. As described by [ProAndroid5, C.1], a fragment can be considered like a sub-activity, that runs inside an activity. An activity can host multiple fragments that can basically act the same as an activity on its own. A fragment has a similar lifecycle as the one for the activity, described above, however since a fragment cannot stand on its own, it is always bound to an activity, it will adhere to the activity's lifecycle as well. Fragments were originally introduced to support devices with larger screens, such as tablets or TVs, however, are also commonly used in smartphone apps.

An activity draws the user interface components, which in android are called views, onto the device’s screen. Views in Android are the basic building blocks of all user interface components and have two major subclasses, Widgets and ViewGroups. Widgets represent all interactive UI elements like buttons or text fields, and ViewGroups all layout components, which as described in [AndroidDevView] *“are invisible containers that hold other Views (or other ViewGroups) and define their layout properties”*. Android uses XML files for its layouts. Usually there are one or more layout files for each activity, in which a combination of views defines the layout for the activity. The layout files are referenced in the activity and converted to Java objects during an app’s build process, additionally, views can also be created from code.

Service

Services in android are processes that an application is running in the background, that are invisible to the user. Services are for example used to asynchronously run longer lasting tasks that an app wants to perform without blocking the user interface or to work on a remote process.

Broadcast Receiver

A broadcast receiver allows an app to respond to system-wide broadcasts, even when an app is not currently running. This allows apps to execute code outside of their regular lifecycle, for example using a broadcast receiver an app can schedule to show a notification to the user when an event occurs. A broadcast receiver can also be very useful in combination with a service that is running asynchronously, the service can send a message to the broadcast receiver upon its completion and trigger the app to react according to the results of the service.

Content Provider

Content providers provide an interface to access data stored in a database without having to be aware of the underlying databases schema, by providing a URI based model for accessing data. In general, a ContentProvider is used to share data between applications, for example giving that appropriate permissions are granted, an application can access a user’s contacts on a device, using a ContentProvider that is made available by the operating system. As described by [Meier], ContentProviders *“allow you to decouple the application components that consume data from their underlying data sources”*, which brings with it a certain set of benefits and convenience, like using tools that Android provides to load data from ContentProviders, which makes using a ContentProvider useful, even without sharing data with other applications.

To access the data made available by a content provider, Android uses a so-called ContentResolver, which allows an app to acquire a cursor object that provides access to read and write to a database. A cursor object can also be obtained by querying a database directly. It is used to iterate over the database rows returned by the query and retrieve their content.

4.1.2 Using android components

AndroidManifest

The Android manifest is an XML file that declares application components and holds meta information that the application requires to run. All components that an app uses must be declared in the manifest, which is used to let the system know what components an app contains. Furthermore, permissions that the app needs to run have to be defined in the manifest as well.

Intent

While as mentioned in the previous section, a content provider is used using a content resolver, the other three android components, activities, services and broadcast receivers, are accessed using intents. An intent in android describes an operation that is used to communicate between different components. If for example, one activity wants to launch another activity it does so using an intent object. This is not bound to just a single application, intent can also be sent and received between apps or with the operating system itself. To receive incoming intents an app has to register itself with the operating system as being able to handle a specific intent. It does so by defining an intent filter for an activity or a service in the AndroidManifest that the app uses to handle the incoming intent.

Android uses two different type of intents, explicit intents, and implicit intents. An explicit intent specifies the exact class that should be run by executing the operation and is used for example for starting a specific activity or a service. An implicit intent on the other hand just describes the general action that should be performed and lets the operating system choose how the intent should be handled, for example sending an intent with a URL would trigger the operating system to perform the action with an app that has registered itself as being able to handle that type of intent, which in case of a URL pointing to a website could be a browser or if it was a URL pointing to Google Maps, the Google Maps Android app itself would be used to handle the incoming intent.

4.1.3 Storing application data

Android offers a variety of ways to store persistent application data, next to using a content provider or an SQLite database, apps can also store data on the device's internal or external storage. Files saved by an app are stored in the app's user directory, and can usually only be accessed by the app itself. Any app-specific settings, that can be configured by the user or the app, are stored using SharedPreferences, which are also stored in the app's user directory. SharedPreferences are used to store a small set of key-value pairs. As they can only hold simple data types like for example boolean, integer or strings they are commonly only used to set flags or quick information that the application needs on runtime.

4.1.4 Android Libraries

When developing for Android the use of libraries to speed up the development process and not having to reinvent the wheel the whole time is encouraged, with Android Studio offering an easy way to add those libraries to a project using Gradle, the build tool that Android Studio relies on. The following section will describe the libraries used for this project.

Google Play Services

With Google Play Services, Google offers an easy way to connect to its own services/APIs like maps, analytics, ads and much more. Any android device that has access to the Google Play store will automatically have the Google Play services app running as a service in the background to which any app that implements the Google Play services client library has access too. Over the last few years, Google has been moving many services out of the operating system and into Google Play Services so updates to APIs can be provided without the need to update the operating system. The following services were used to implement Foodtruckfinder, their implementation will be further explained in section 5 (link to section)

App Indexing

The App Indexing service allows an app to be indexed on the devices, which makes its data available to be searched within the Google Search on the device.

Google Cloud Messaging

Cloud Messaging provides a way to send and receive messages from the cloud. It offers various tools including push messages and a task scheduler called GCMTaskService that Foodtruckfinder uses to schedule background updates to its data. Cloud Messaging has recently been moved into Firebase, a mobile/cloud development platform that Google acquired in 2014. Foodtruckfinder currently still uses the old GCM API, in a future version this will be fully migrated to its FCM (Firebase Cloud Messaging) counterpart.

Location

Foodtruck Finder uses the location service to access the fused location provider, an API that intelligently combines different signals to provide location information [GoogleDevLocation] in a way that is more efficient than for example directly using a device's GPS sensor, as well as the Geofence API which is used to provide notifications to the user about nearby trucks, triggered when a user enters a certain area.

Maps

The Maps service allows an app to embed and display a customized map or parts of it within an app. Additionally, I made use of an API called maps utilities, that is also provided by Google. It offers a more streamlined way to customize a map with custom markers and marker clusters that automatically group markers together if there is not enough room to show all the markers in a certain area.

Places

With the Places API, Google provides access to the same data that is used by Google Maps and Google+ Local. With the PlacePicker widget, it provides a UI dialog and a list of nearby places, which Foodtruckfinder leverages to let the user pick a custom location instead of the device's location.

OkHttp

OkHttp is an HTTP client developed and maintained by Square that provides a more convenient way of making HTTP requests than implementing the tools provided with the Android framework. As described on their website [OkHttp]: *"OkHttp perseveres when the network is troublesome: it will silently recover from common connection problems",* meaning it already includes features and error handling that would've needed to be implemented by hand instead.

Google Maps Android API Utility Library

This library provides advanced features for using maps in android applications, such as tools for adding custom markers to a map or clustering markers together if a number of markers at a certain zoom level is too much to clearly distinguish them.

Schematic

Schematic is a library that allows for the automatic generation of a ContentProvider that is backed by an SQLite database. A ContentProvider in Android is a tool used to share the data of an app primarily with other apps but can also be used within segments of an app itself. It uses a URL based scheme to provide access to the underlying SQLite database.

Glide

Glide is an image loading library that automatically loads and caches images either to memory or to disk. It provides an interface that as mentioned on their GitHub repository [Glide]: "supports fetching, decoding, and displaying video stills, images, and animated GIFs". Foodtruckfinder uses Glide primarily to display the logos of food trucks in its main screen, where it provides smooth scrolling of a large list as it was designed to work with androids list-based views like RecyclerView or ListView.

4.2 App Engine

Under the name App Engine, Google offers a platform as a service solution that allows for rapid development of scalable web applications and API backends. App engine applications don't run on a single server but instead run in a distributed environment free of capacity management, server maintenance, and load balancing. While a number of requests sent to the web application increases, app engine automatically assigns additional resources as needed without the developer having to change a line of code, as described by [CloudComp]. App Engine offers a runtime for various languages such as Java, Python, PHP and Go. In this project, Java was used to create a Java Servlet that utilizes app engine's Memcache feature to act as a buffer between the app and Foodtrucks Deutschland's API.

4.2.1 Java Servlet

As perfectly summed up by [JavaEE7Tutorial, C.17], "a servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed by means of a request-response programming model", meaning a server component that receives a request, processes that request on the server it is running and sends a response to the sender. In the case of App Engine, that server is represented by Google's cloud that dynamically scales to the users demand. Applications running on App Engine run in a restricted environment, they do not have access to the file system they are running on and are not allowed to make any system calls, which alleviates a lot of security problems.

4.2.2 Foodtrucks Deutschland API

The API provided by Foodtrucks Deutschland is accessed using HTTP post request, which will return data sets in the form of JSON files. According to [JSON], "JSON (JavaScript Object Notation) is a lightweight data-interchange format", that aims to be easy to read for humans and machines. Based on a subset of JavaScript it is most commonly used in web development.

```
JSONDocument := '{' JSONElements '}'
JSONElements := JSONObject|JSONArray|KeyValuePair
JSONObject := '{' (JSONElements ',')* '}'
JSONArray := '[' (JSONElements|Value ',')* ']'
KeyValuePair := '"key":' Value
Value := String|Integer|Boolean|null
```

Figure 6 JSON Syntax in EBNF

The basic syntax of a JSON document is outlined in **Figure 6**. A JSON document contains one or more JSON elements, which can either be an object, an array, or a key-value pair. A JSON object can contain all JSON elements while a JSON array can also contain simple values. The key-value pairs utilize a string as key and can hold values of the types, string, integer, boolean or null. To structure the document JSON uses curly brackets for objects and square brackets for arrays. **Figure 7** shows a JSON example that contains location information for certain regions. In line 2 an array called regions is declared, that has two entries in line 3 to 9 and line 10 to 16. This file is used to provide location coordinates for regions, without having to have an exact address.

```
 1 {
 2   "regions": [
 3     {
 4       "id": "5ce37ae3db1a88c258388d98bb83bf75",
 5       "name": "Aachen",
 6       "latitude": "50.773376",
 7       "longitude": "6.086938",
 8       "state": "NW"
 9     },
10     {
11       "id": "157507ba44e9b075b848f82730db8fb2",
12       "name": "Berlin",
13       "latitude": "52.517976",
14       "longitude": "13.404108",
15       "state": "BE"
16     }
17   ]
18 }
```

Figure 7 JSON data providing location information for specific regions

Foodtrucks Deutschland’s API was developed to be consumed by their own website, as well as the existing mobile applications for IOS and Android. The development of Foodtruck Finder happened in close collaboration with Foodtrucks Deutschland to extend the API to the needs of Foodtruck Finder. The API currently offers three major endpoints, to return information about currently active locations, registered food truck operators as well as specific detailed information about single operators.

5 Application Design

Next to the main component, the Android App, with the realization of this project, a web app running on Google App Engine that acts as a buffer between Foodtruck Finder and Foodtruck Deutschland’s API has been created. The web app utilizes App Engine's ability to store data in memory to reduce the load on the actual API.

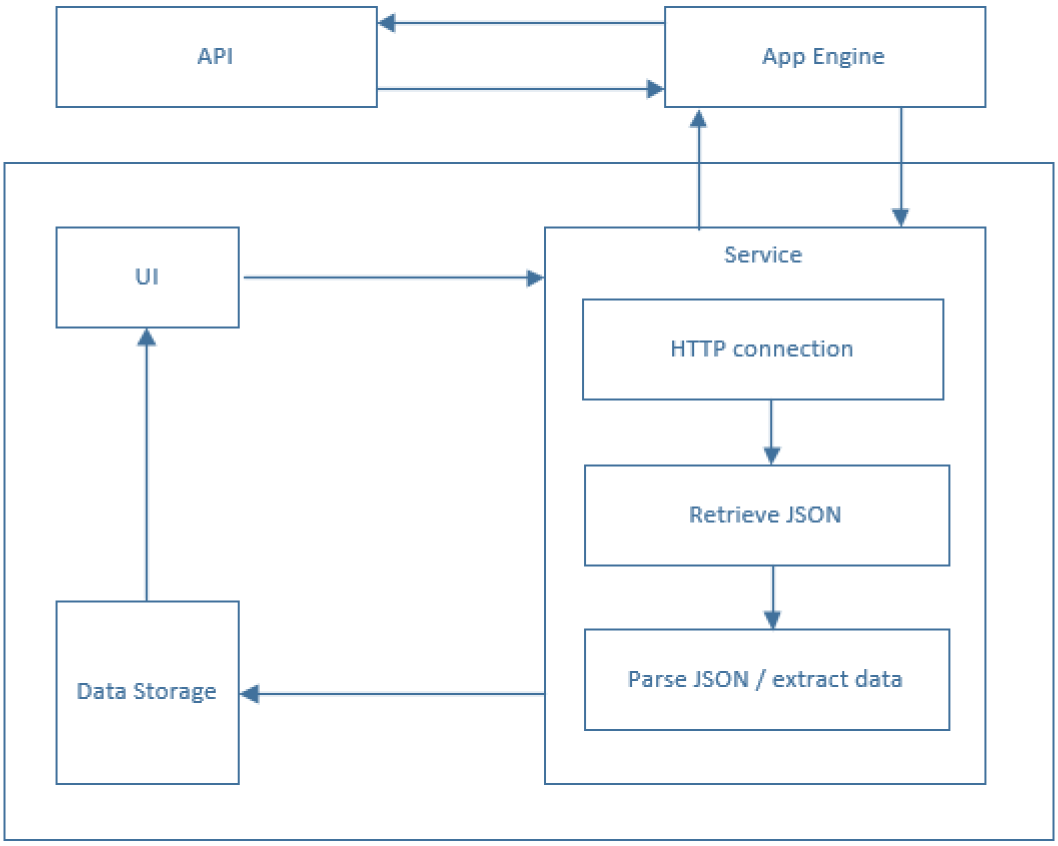


Figure 8 Interaction between android app and API

Figure 8 describes the basic interaction between Android App, App Engine, and the API. The UI component of the Android app will start a service that will send a request to the servlet running on app engine, the servlet will then either respond with the data stored in its memory from a previous request or send a new request to the API, store the response in memory and send it back to the service running in the android app. The service will parse the retrieved JSON file, extract the data needed, and store it in the data storage, which takes the form, of a ContentProvider backed by an SQLite database. After successfully storing the data the service will notify the UI about the new available data, which will trigger the UI to refresh its views and display the new data.

5.1 Android App

The composition of an android app typically can be split up into two major categories, which are the UI components on one hand, basically everything a user interacts with when using the app, and on the other hand are background services that run asynchronously in a separate thread, typically used for loading and handling of data used by the application. It is crucial that these parts run in separate threads, so long lasting operations on the services side don't block the user interface and result in the application appearing to freeze for the user.

5.1.1 User Interface

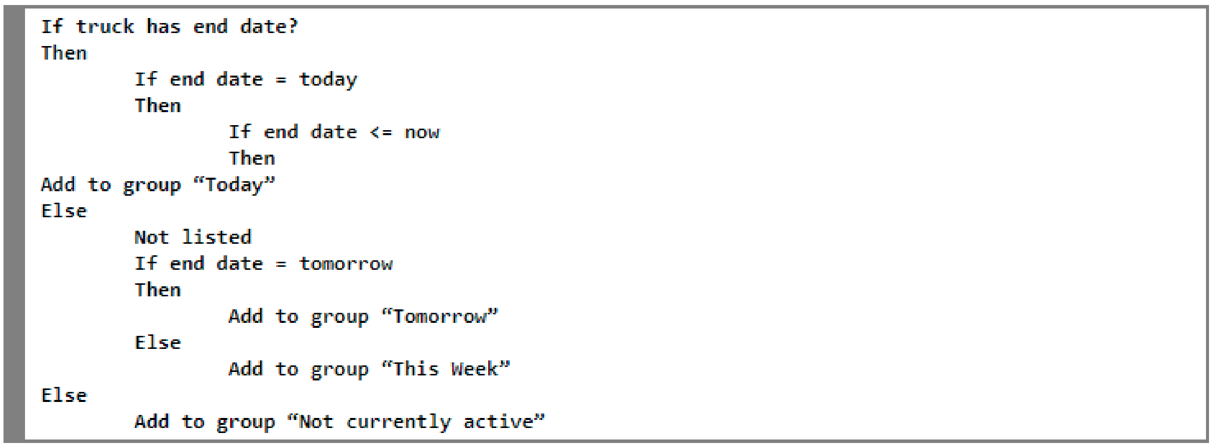
Foodtruck Finder's user interface consists of three major components. MainActivity is the main way of navigating the app and finding food trucks, it displays a grouped list of trucks sorted by the distance to user's location. DetailActivity is launched upon clicking on one of the entries in the list and will display information about the selected food truck such as its location, schedule, and information about the truck itself. MapActivity displays a full-screen map with markers showing the trucks' locations.

Additionally, there are WelcomeActivity which will be displayed once when the application is first installed, as well as SettingsActivity used to modify certain parameters of the app.

MainActivity

When using the app, the first screen a user sees is MainActivity. This activity is designed to be intuitive and bring the most essential data to the user. A user opening the app, in general, has a single intention and that is finding a food truck nearby that serves something that a user is interested in. To achieve this, MainActivity will display a list of food trucks, sorted by the distance to the user. Each item in the list will show the name of the truck, its logo, its offer text, which is a short description provided by the trucks themselves, as well as the location and the distance to the user.

The dataset used by Foodtruck Finder consists of a week's worth of location information for the trucks, as well as the information about all registered food trucks. Displaying this in a list based format created certain challenges, when sorting the list. Sorting the list by distance is obvious as the closer a truck is to a user the more likely the user is to go there. Additionally sorting by date, which means listing trucks that are currently open first, is also obvious for the same reason. Using this sorting model would create a list where all trucks that are currently open are listed first, sorted by the distance to the users, then all trucks that are open on the next day, sorted by the distance to the user, and the same for the rest of the week. Additionally at the end of the list this would put all the trucks that are currently not open. When the fact that some food truck owners have multiple trucks at different locations at the same time is taken into consideration, this would result in a long list of items. To solve this issue, Foodtruck Finder groups the list by availability. The list is split up into four groups, **today**, **tomorrow**, **this week** and **not currently available**. Trucks that have multiple locations are grouped together, with the closest location to the user being shown in the list. **Today** will list all trucks that are open at the moment the user uses the app, as well as trucks that will be open later in the day, however will not display trucks that were open earlier but had already closed. **Tomorrow** will list all trucks that are available tomorrow, **this week** all that are available throughout the week and all other trucks that are currently not available will be listed under **not currently available**. **Figure 9** shows the algorithm used to determine the grouping of the list, with end date being the time and date until which a truck is available at a certain day.



```
If truck has end date?
Then
        If end date = today
        Then
                If end date <= now
                Then
Add to group "Today"
Else
        Not listed
        If end date = tomorrow
        Then
                Add to group "Tomorrow"
        Else
                Add to group "This Week"
Else
        Add to group "Not currently active"
```

Figure 9 Algorithm used for grouping food trucks list

Furthermore, it is essential that the list is searchable and can be filtered by common categories. Since searching for food trucks is the predominant feature of the app, the search has been given a position which Google designed to promote the primary action of an app, with a so-called floating action button at the bottom right of the app. According to [MaterialGuidelines], *"Floating action buttons are used for a promoted action. They are distinguished by a circled icon floating above the UI"*. To keep consistency within the app, the same pattern has also been implemented in MapActivity.

To filter the list by categories, a drawer that can be shown by swiping from right to left at the edge of the screen or by pushing the filter button in the toolbar is used. The drawer overlays MainActivity and shows a list of popular tags, that can be used to filter the list. It allows for multiple entries to be selected which will result in filtering the list to show any trucks that match any of the selected categories.

DetailActivity

DetailActivity is accessed by clicking on one of the entries in MainActivity. DetailActivity is there to provide the most important information about a single food truck, which is information about the truck, exact location and its schedule for the upcoming week. For trucks that are registered with Foodtruck Deutschland's premium service contact information like website, email, phone number and social media links are also shown here.

To display the location DetailActivity features an embedded map view, that shows the area surrounding the truck with a map marker pointing at the truck's location. If there are multiple locations for a truck, the one closest to the device will be displayed.

The schedule of the truck is shown in a list sorted chronologically as well as by distance to the user. Each entry in the schedule contains the date, time and location as well as a link that will allow the user to open an app for navigating to the truck's location.

MapActivity

Upon clicking the map view in DetailActivity the user is taken to MapActivity, which will display a fullscreen map, similar to Google Maps, with map markers for all available food trucks. The map offers the same user interaction patterns that are common for apps like Google Maps, such as double-tap or pinch to zoom. To keep familiarity between screens, MapActivity can be filtered by the same availability groups that MainActivity is grouped by, which are today, tomorrow and this week. To switch between the different categories MapActivity uses a bottom bar which is designed to *"make it easy to explore and switch between top-level views in a single tap"* [MaterialGuidelines]. Upon tapping on a bottom navigation entry, MapActivity seamlessly reloads the map markers for the given availability period.

As mentioned previously, MapActivity also puts its search functionality prominently in a floating action button at the bottom right corner of the screen, which upon a tap expands the search bar in the activities toolbar at the top, where a search term can be entered. Upon selecting a result MapActivity moves the map to the results location.

WelcomeActivity

The first time a user launches FoodtruckFinder he will be greeted with WelcomeActivity, which will display a set of slides introducing the user to the features of the app. This serves as an onboarding process to let the user know what to expect from using the app, as well as for the initial setup of the app, as the user is asked for permission to use the location of the device, and set some important settings like the radius used to look for food trucks and to enable notifications for trucks in the user's area. Clicking and reading through the slides of the WelcomeActivity will take a user from 5 to 20 seconds. Foodtruck Finder utilizes this time to connect to the API, get the user's location and setup the data to display once WelcomeActivity is finished.

5.1.2 Requesting Location access

Foodtruck Finder’s core functionality, displaying trucks based on a user's location, requires permission to access the device's location. On older Android versions permissions to device features were given upon installation of the app, and could not be revoked by the user after that. In an effort to raise user’s awareness and provide more transparency about app’s permissions, starting with Android version 6, Google introduced a new permissions model where permissions are requested at runtime and can be revoked by the user at any time. Due to this fact it is important to make the users of Foodtruck Finder aware that permission to request location access is constantly required as the app will not function properly without it. Therefore during the onboarding process, WelcomeActivity will display a slide, explaining why Foodtruck Finder needs access to the device‘s location and stating the fact that it can’t function without it. The slide features a button that can be used to grant access to location services, which upon click will display a dialog with which the user can allow or deny access to the device‘s location. WelcomeActivity will only allow the user to go to the next slide if location access is granted. To make sure that the permission has not been revoked by the user later on, MainActivity will check for the permission every time it launches and will offer the same dialog again if it has been revoked.

5.1.3 Service

To retrieve the data provided by Foodtrucks Deutschland, Foodtruck Finder uses a service that is running in the background. As mentioned previously, Foodtrucks Deutschland’s API provides its data via various endpoints. It provides an endpoint for locations, operators and operator details. Locations can either be a list of food trucks that are active on the current day or the next seven days. Operators contains a list of all food truck operators that are registered with their network. Operator details provides specific information for a single food truck operator. Foodtruck Finder uses the same service for all of the API’s endpoints. The job of the service is to retrieve the data from one of the endpoints, extract the information that Foodtruck Finder requires and store it locally on the device.

To keep the data that is stored locally up to date and reduce the load time when opening the app, the service is scheduled to run in the background at a certain period, without the app having to be running. The locations data is scheduled to update daily, and the operator data is scheduled to update weekly.

5.1.4 Data Storage

Foodtruck Finder stores its data in an SQLite database that is accessed through a private ContentProvider. Usually, a ContentProvider in Android is used to share data between applications, in Foodtruck Finder however it has been found beneficial to use a ContentProvider even though no data is shared outside of the app itself. Using a ContentProvider provides an abstraction layer to the database which eliminates the need to manage opening and closing the database, and instead uses self-defined URIs to access the data. Using a library like schematic furthermore, streamlines creating database tables and automatically creates the ContentProvider. How this works is explained in more detail in the following section.

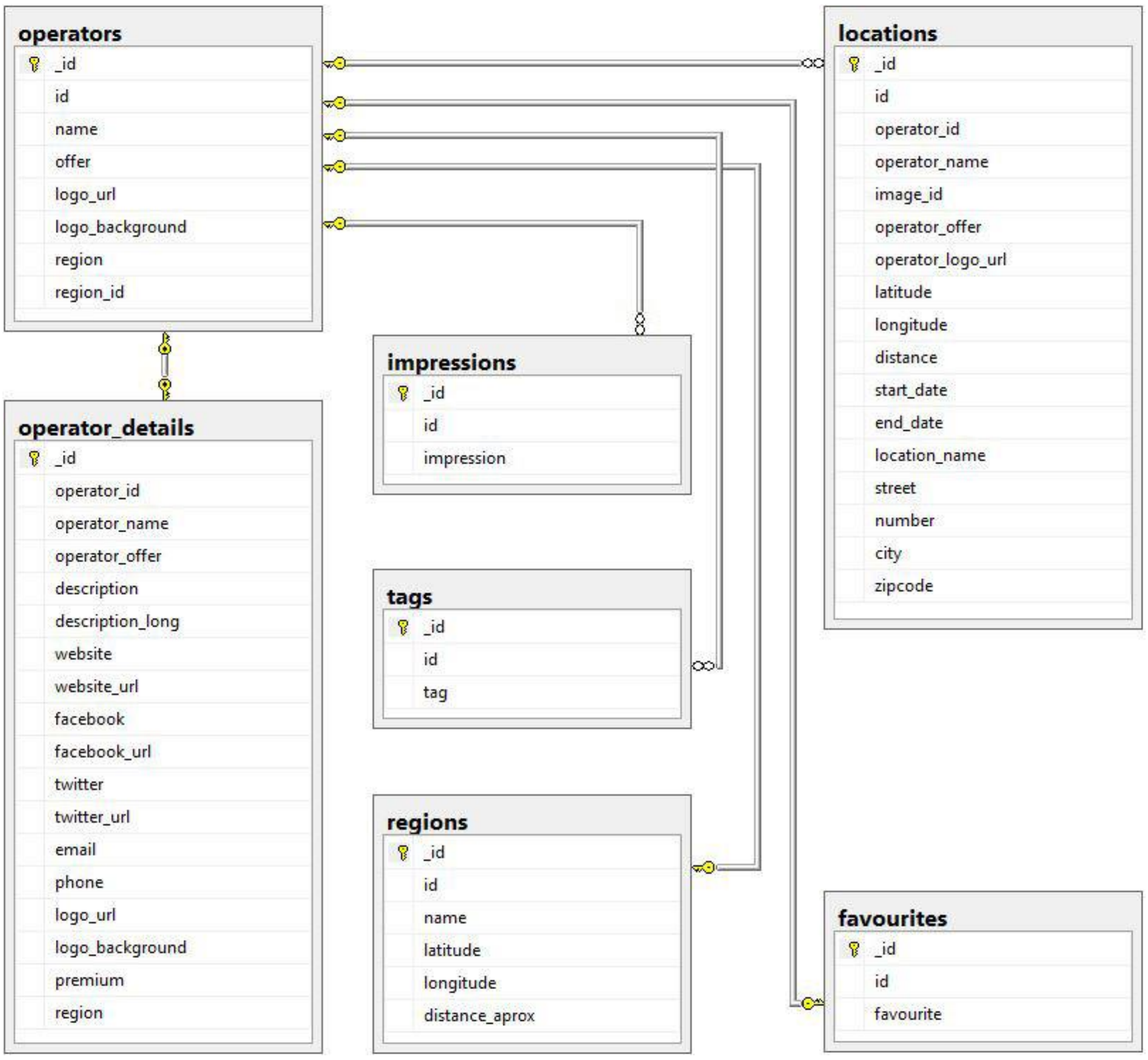


Figure 10 Entity-Relationship diagram for Foodtruck Finder’s database

The database has seven tables, with operators being the main starting point for most queries, as can be seen in **Figure 10**. operators holds all trucks that are currently registered with Foodtrucks Deutschland, this is a mostly static table which as mentioned in the previous section will be automatically updated once per week. locations has all locations and dates on which an operator is active for the upcoming week, this is the most important table as the data in here is used to fill the list in MainActivity as well as the schedule in DetailActivity. Each operator can have an entry in operator_details, which the app creates when accessing the DetailActivity for a specific truck, here information about an operator's offer and his contact information is stored. The other tables hold additional information needed. regions provides longitude and latitude to an operator that currently is not active and has no location, so the map view in DetailActivity can show the region the truck usually operates. tags hold a list of food categories that match the operator, impressions holds a set of URLs that link to images displayed in the toolbar of DetailActivity and favorites holds an entry for each truck that has been saved as a favorite.

5.2 App Engine Web App

To reduce server load on Foodtrucks Deutschland‘s API, as well as load time for users, within the scope of this project a Java Servlet running on Google’s App Engine has been developed. The servlet acts as a buffer between Foodtruck Finder and the API. Upon receiving a request, the servlet will fetch the corresponding data from Foodtruck Deutschland’s API and store it temporarily in memory and return it to Foodtruck Finder. If the same data is requested again, instead of fetching the data from Foodtrucks Deutschland, it will be returned from memory, as long as App Engine keeps the data in memory. In its current implementation, the application utilizes App Engine’s distributed in-memory data cache that is shared with other App Engine apps. Data stored this way is not persistent and will be cleared by App Engine if another application requires it.

Using App Engine, which is a distributed system across many regions, compared to accessing Foodtruck Deutschland’s server running the API, which is in a fixed location in Germany, data transfer rates for users are more consistent and depend less on the user’s location.

6 Implementation

6.1 Initial launch and setup

Upon first launch of the app, certain setup and initialization steps have to be performed. When an android app is launched by clicking the launcher icon on the home screen, the first activity that will be started is always the one that has been designated as MAIN in the android manifest. As shown in **Figure 11**, if the user launches Foodtruck Finder for the first time MainActivity will be started, which will immediately start WelcomeActivity to start the onboarding process, while still processing additional tasks launched in its `onCreate` method in the background. WelcomeActivity not only acts as an introduction to the app but also as a way to keep the user busy for a few seconds, so actual wait time after finishing the setup in WelcomeActivity is reduced or even eliminated.

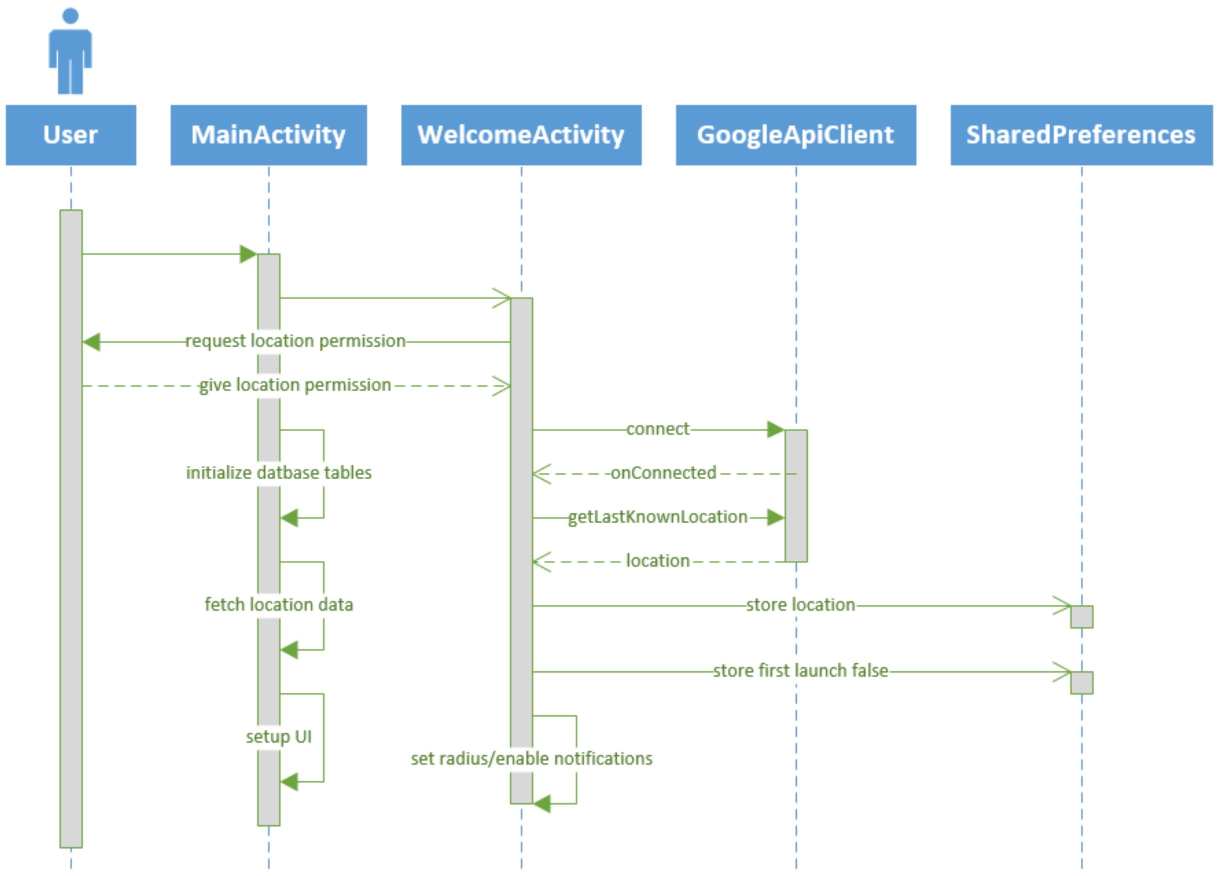


Figure 11 Sequence of events when first launching Foodtruck Finder after it was installed on a device

While the user is busy with WelcomeActivity, MainActivity will initialize database tables with offline data that is shipped with the app in the form of JSON files. The data shipped with the app includes a recent version of a dataset containing all food trucks registered with Foodtrucks

Deutschland, as well as some location data about the regions they operate in. This location data is used to map trucks that are not currently active to a certain region, so they can still be displayed in order of distance to the user, in the “not currently on the road” section, without having an active location. Next to this, the actual location data of trucks active for the upcoming week is requested from the API. Shipping a partial amount of the data required to run with the app like this, has shown to be a great improvement on initial launch speed of the app.

WelcomeActivity consists of five pages that the user has to go through before the app can be used. This is implemented using a custom ViewPager. A ViewPager is as described in [AndroidDevViewPager] a *“Layout manager that allows the user to flip left and right through pages of data”*. Foodtruck Finder uses a custom ViewPager with the added ability to enable or disable paging, meaning the ability to move to the next or previous page. The first page shows a welcome message to the user without any required user input. The second page asks the user to give permission to access the device’s location and explains to the user that it cannot function without this permission. As described in the previous section, when the second page is active the paging feature of the ViewPager will be disabled until the user agrees to give permission to access the device's location. After permission is given, WelcomeActivity connects to Google’s API to retrieve the device's last known location and calls a utility method that will store the location data as longitude and latitude in SharedPreferences and start a task calculating the distance for each location stored in the database. This process assumes that the task fetching the location data, that has been started in MainActivity has completed, and location data that can be updated is present. During all tests with users, this has been the case, however, there is the possibility that the user is on a slow internet connection and location data might not be present at the time, in that case, MainActivity will start the task to update the distance values again when location data has successfully been retrieved.

The following pages allow the user to set the radius that is used for displaying food trucks, and to enable notifications for nearby trucks, which will activate the use of geofences. The last page will display a message thanking the user for using the app. When the user finishes the process by pressing the OK button, a flag is set in the app's SharedPreferences preventing WelcomeActivity to be launched again when starting the app. When WelcomeActivity exits, the user is taken back to MainActivity.

6.2 Location Strategy

When developing an app that needs to be aware of a device's location it is important to be considerate how to acquire the location and how accurate that location needs to be. Android offers multiple ways to acquire a device’s location such as directly accessing a device's GPS sensor or its network location provider which utilizes WiFi and network triangulation. Foodtruck Finder requires a user's location because the list of trucks shown to the user is ordered by distance and limited to a certain radius that can be changed in its settings. Showing a list of trucks without these location-aware features would not be useful, as there would be no point of reference for the user and trucks shown could be far away. For the currently average available amount of food trucks on a user's location, it has been determined that a default radius of 50 kilometers is sufficient to deliver enough relevant data to the users and all food trucks found will be in driving distance. With a radius of this size, the accuracy of the location required by the app is not very important, so there is no need to acquire pinpoint accurate GPS location, instead Foodtruck Finder makes use of Google’s Fused Location Provider API to acquire the device's last known location, which is a location that may have been stored on the device previously by another application, and therefore has a lower accuracy. The Fused Location Provider API is integrated into Google Play Services which is available on any device that has access to the Google Play Store. As there are currently no plans to distribute the app on another platform, since the Play Store covers the biggest available user base, depending on Play Services is not an issue, and the benefits of using Google’s API that is designed to be as battery efficient as possible over other methods like for example actively polling for a device's location are apparent.

Every time MainActivity is launched it uses the fused location client to send a request to the API to get the device's last known location using its getLastKnownLocation method. If the client was able to successfully connect to the API, it will either return a Location object or it will return null when no last location is found. getLastKnownLocation will not actively poll for a new location, it will simply return the location that was last stored on the device, which means if for example the location services on the device have been disabled and re-enabled just before calling getLastKnownLocation, there will be no stored last location and null is returned instead. To handle this case, MainActivity will trigger the location client to start looking for location updates, using its requestLocationUpdates method. This will start actively polling for a new location using whatever provider is available at the time, be it GPS or network. The request is set to stop looking for updates after the first location has been received, as this is all the app requires.

As previously mentioned, if a location has previously been acquired, it has been stored as latitude and longitude in the app’s SharedPreferences. If the distance between a newly acquired location and the previously stored location is bigger than one thousand meters, the SharedPreferences will be updated with the new location, and the task to recalculate the distances to currently active trucks is launched. To ensure that the device is able to acquire a location, Foodtruck Finder has to check if location services are enabled on the device and that permission to access the device’s location is still granted, as any device running Android 6 and later has the ability to revoke the permission as mentioned earlier. When either of these checks fails, a dialog is shown to the user, allowing them to enable location services or give permission to access the location. Both checks are performed every time MainActivity is launched.

6.3 Geofences

To provide information about nearby food trucks to users, Foodtruck Finder makes use of the Geofences API, which also comes integrated into Play Services. A geofence is created by defining a location using its latitude and longitude and an area around it that is set using a radius. If a device enters the defined area, it will be triggered to take action on this event, such as showing notifications to the user. If location services on the device are enabled Play Services will detect if the device enters one of the defined geofences and will send a callback, in the form of an intent, that is received by a service. The service obtains a geofencing event from the received intent and process it according to the type of the retrieved intent. Foodtruck Finder defines an IntentService that is named GeofenceTransitionsIntentService to handle incoming events from the Geofencing API.

As can be seen in figure 12, there are three event types that can be used to trigger a geofence, which are enter and leave, triggered when a device enters or leaves the geofence, and dwell which is triggered when the device stays inside of the geofence for a specified amount of time, that can be set when creating the geofence.

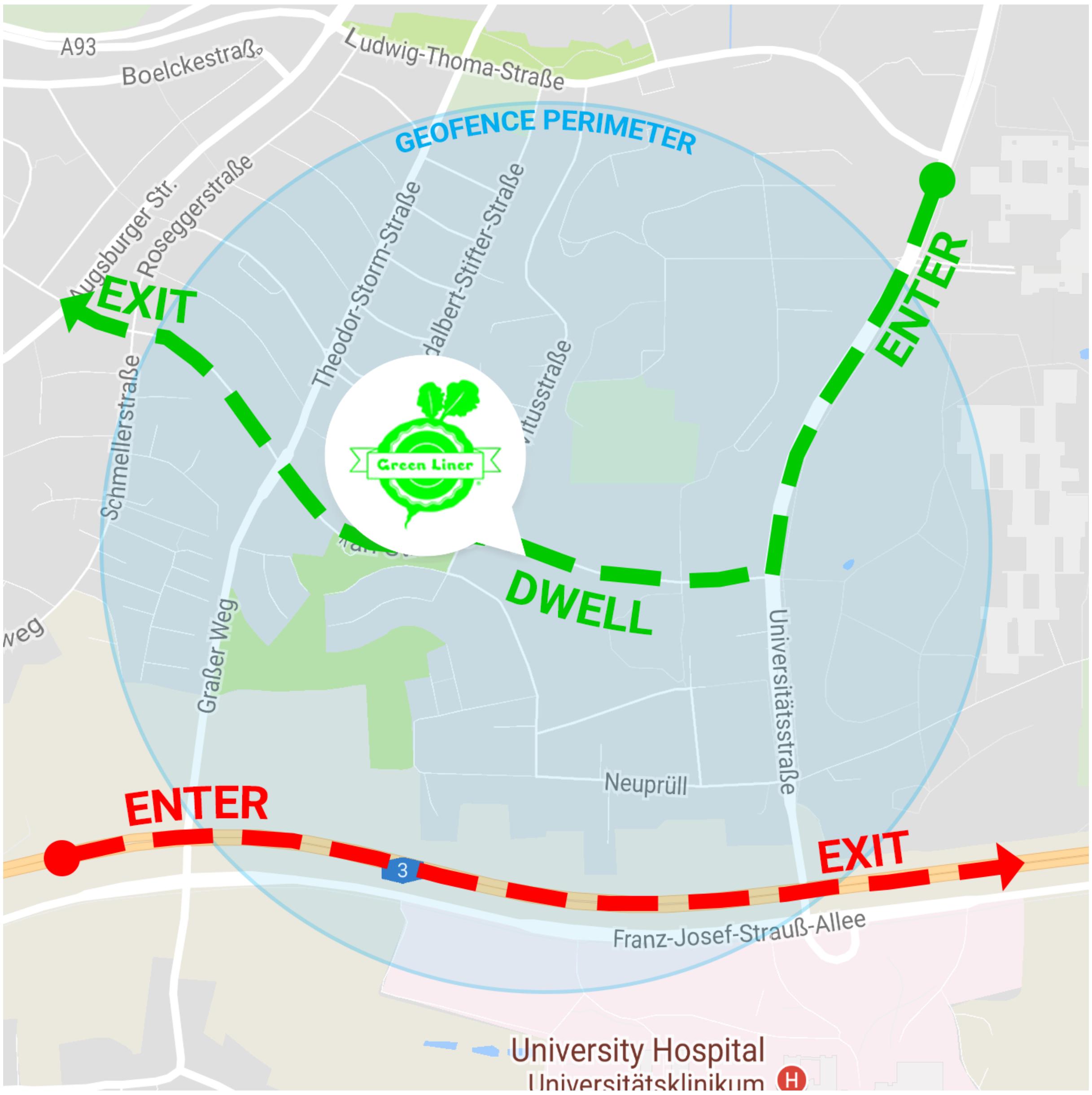


Figure 12 Geofence Events

To highlight the differences between the event types, Figure 12 shows two lines that represent routes a car could take through the geofence. In this example, both the green line and the red line would trigger the enter and exit events, however only the green line would trigger the dwell event. The car on the red line is driving on the highway, which means it will drive too fast to spend enough time in the geofence to trigger the dwell event. On the other hand, the car on the green line is driving through the city at a much slower speed, which means it will spend enough time inside the geofence to also trigger the dwell event. As of the writing of this document Foodtruck Finder uses the enter event to trigger the geofences, in which case both cars in this example would trigger the geofence and be notified about the nearby food truck.

To create the geofences, a query is sent to the content provider, returning a set of food truck locations that are currently active, in order of distance to the user. This set is limited to 100 items, as this is the maximum amount of geofences that can be set per device per user. For each item in the set, a geofence object is created, using the location's coordinates, an identifier unique to each location entry and a radius of 2000 meters. The Geofencing API provides a client that is used to send the created geofences to the API. To do this the geofencing client expects a

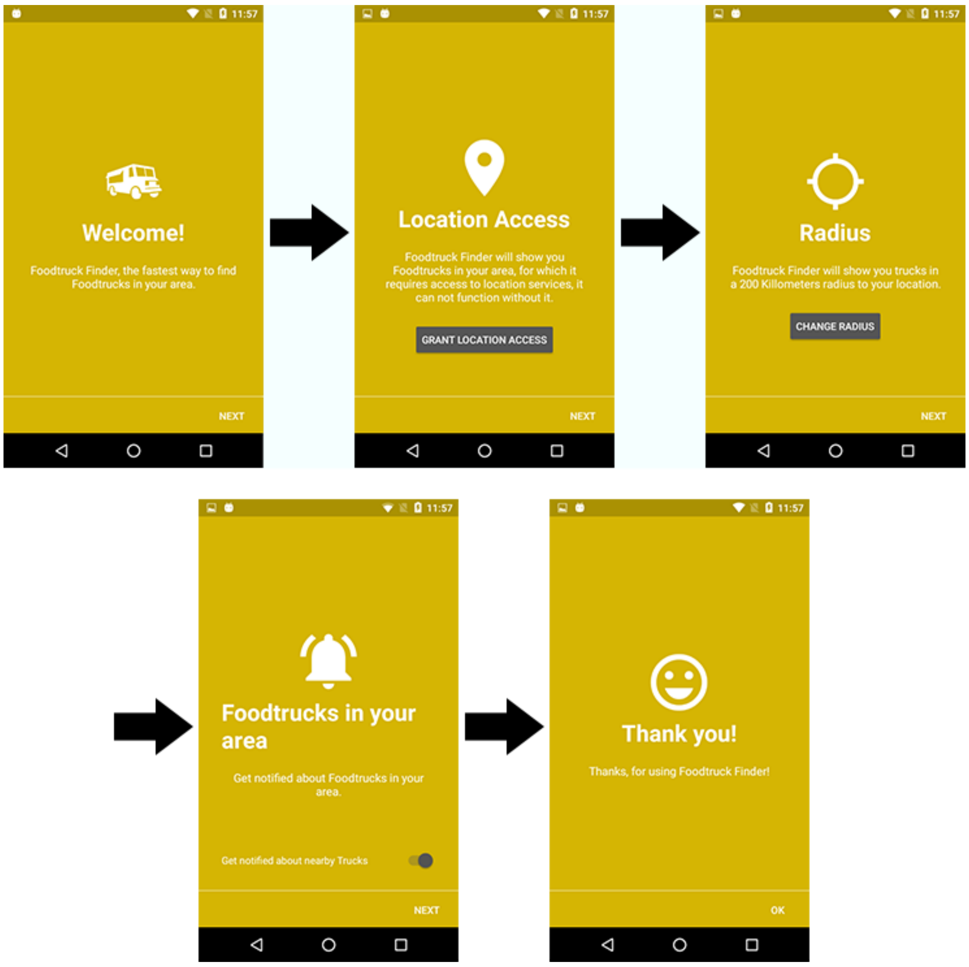
GeofencingRequest and a PendingIntent. The GeofencingRequest specifies the list of geofences that should be monitored and sets a property that will determine how the geofences are initially triggered. If the device is already inside the geofence when it is added, the property is set to trigger the geofence enter event. The PendingIntent is used to send back the Intent notifying the GeofenceTransitionsIntentService about the geofence event. Thanks to the use of a PendingIntent the app itself does not need to be running to receive geofencing events, instead, the service will silently receive the intent and extract the geofence event from it.

When the service receives an intent, it will extract a GeofencingEvent object from the intent that can contain one or more geofences that have been triggered. Using the identifiers set when creating the geofences, the service will query the content provider to get the corresponding truck locations. Since the geofences could have been set before the truck was active at the retrieved location, or the event could have been received after the truck had already closed, the service will check each of the returned entries if the corresponding truck is currently active. If one or more active trucks are found, a notification is created on the device. If multiple trucks are found they are listed in a single notification which upon clicking will take the user to Foodtruck Finder's MainActivity if only a single truck is found a click on the notification will take the user directly to the DetailActivity for that specific truck.

6.4 User Interface

6.4.1 WelcomeActivity

As mentioned in the previous chapter, WelcomeActivity is used as an introduction to the app, letting the user know what can be expected. It will only be launched once after Foodtruck Finder is installed unless the app's data is manually deleted by the user. The slides shown in **Figure 13** are implemented using a ViewPager, which is as described by [AndroidDevViewPager] a *"Layout manager that allows the user to flip left and right through pages of data"*, meaning it enables the user to browse the content using a paging behaviour, which lets the user go from page to page by swiping left or right.



Welcome!

Foodtruck Finder, the fastest way to find Foodtrucks in your area.

NEXT

Location Access

Foodtruck Finder will show you Foodtrucks in your area, for which it requires access to location services, it can not function without it.

GRANT LOCATION ACCESS

NEXT

Radius

Foodtruck Finder will show you trucks in a 200 Killometers radius to your location.

CHANGE RADIUS

NEXT

Foodtrucks in your area

Get notified about Foodtrucks in your area.

Get notified about nearby Trucks

NEXT

Thank you!

Thanks, for using Foodtruck Finder!

OK

Figure 13 Slides used for introducing a user to Foodtruck Finder

WelcomeActivity uses a custom ViewPager, that has the added ability to disable or enable the paging behavior. This is achieved by extending the ViewPager class and adding a simple method that will set a boolean that will be checked when a swipe event occurs. **Figure 14** shows the custom class.

```
 1 public class WelcomeSlideViewPager extends ViewPager {
 2   private boolean mPagingEnabled = true;
 3
 4   public WelcomeSlideViewPager(Context context, AttributeSet attrs) {
 5     super(context, attrs);
 6   }
 7
 8   @Override
 9   public boolean onTouchEvent(MotionEvent ev) {
10     return mPagingEnabled && super.onTouchEvent(ev);
11   }
12
13   @Override
14   public boolean onInterceptTouchEvent(MotionEvent ev) {
15     return mPagingEnabled && super.onInterceptTouchEvent(ev);
16   }
17
18   public void setPagingEnabled(boolean enabled) {
19     mPagingEnabled = enabled;
20   }
21 }
```

Figure 14 Custom ViewPager class, line 18-20 shows the additional method

ViewPager has two callback methods that are called when a touch event occurs, `onTouchEvent` (figure 14: l. 9-10) and `onInterceptTouchEvent` (figure 14: l. 14-15), that both will check for the added boolean value. If the value has been set to false, before the touch event occurs, the callback methods will return false without executing the code of its superclass.

The second page in WelcomeActivity will ask the user for permission to access the device's location, which is essential for Foodtruck Finder to run. To make sure the user cannot just skip through WelcomeActivity, which would lead to a not fully functioning app, paging on this page is disabled until the user grants access to the location permission.

The following pages allow the user to change the default radius that is used when looking for food trucks and enable notifications for nearby food trucks, which will enable the creation of the geofences.

It can be assumed that going through WelcomeActivity will take a user somewhere between 5 and 20 seconds, which according to testing the app on various networks should give enough time for MainActivity to be ready for use, after WelcomeActivity is exited, resulting in a seamless experience for the user when using the app for the first time.

6.4.2 MainActivity

MainActivity is the main user interface that Foodtruck Finder provides. As outlined in the previous chapter, it is made up of a scrollable list of items. By employing this design pattern most users will be instantly familiar with the app, as this is one of the most used design patterns in mobile apps. When looking at the, according to [ComScore], ten most used apps in 2017, nine of those use a similar pattern as their main way of navigation, the exceptions here being Google Maps, which by its nature focuses on the map first, however recent version of Google Maps on mobile, additionally offer a list based navigation to browse nearby points of interest.

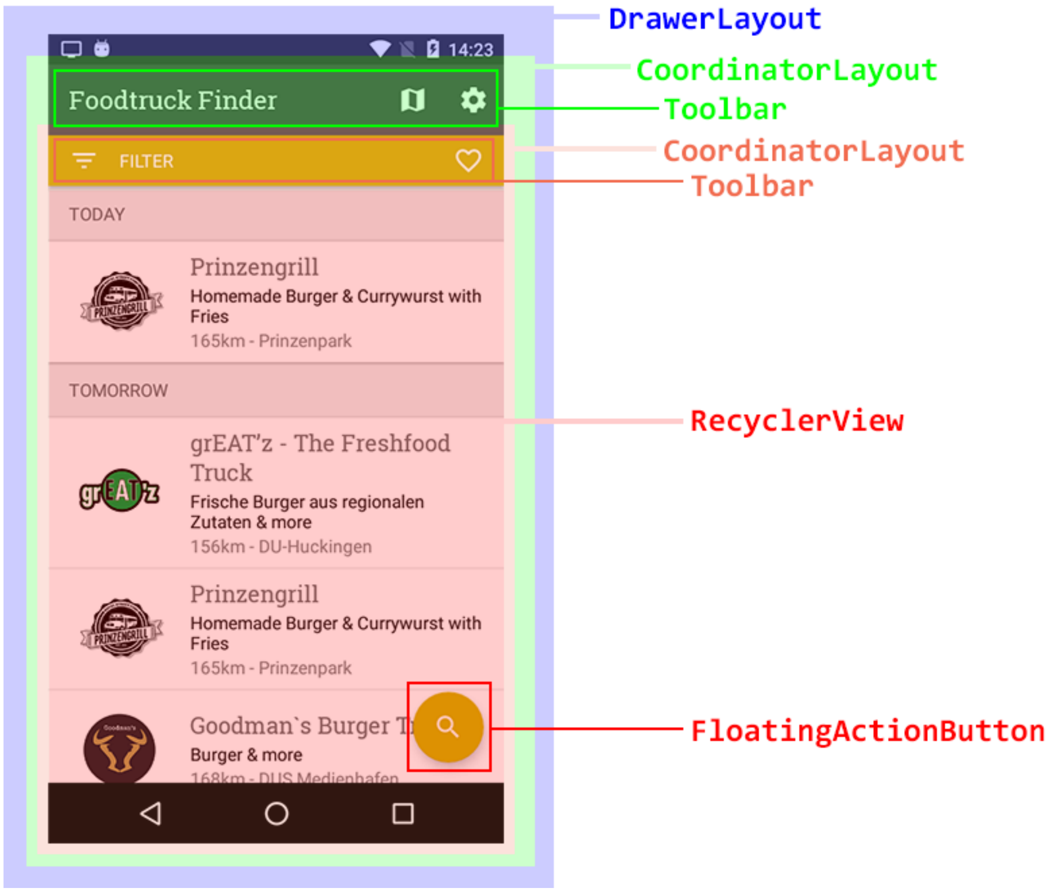


Figure 15 Layout of MainActivity

As illustrated in **Figure 15**, MainActivity consists of a set of nested layout elements surrounding a RecyclerView that holds the main content. The outermost level is a DrawerLayout, which describes a container that can hold multiple drawers that can be swiped in from either side of the screen. Foodtruck Finder uses a drawer that can be swiped in from the right side of the screen, to hold the list of tags that can be used to filter food trucks by category. Inside of the DrawerLayout, there are two nested CoordinatorLayouts, each with their own toolbar. A CoordinatorLayout in android is used to provide certain behaviours to their child views, in Foodtruck Finder's case each of the toolbars reacts differently to the scrolling of the content, with the main toolbar sticking to the top and the secondary toolbar collapsing when the content is scrolled upwards and snapping back into view when scrolled downwards, to provide additional room when browsing the main content.

The main content of the app is displayed using a RecyclerView, which Google introduced with Android 5.0 as a more flexible alternative to the then popular ListView. RecyclerView decouples the representation of the items in a list from the component and allows the use of different layout managers to display the content in various ways, as for example in a grid or a list. Furthermore, RecyclerView will only create as much views as it needs to display what is currently on screen, and reuse or recycle a view that is scrolled off screen to display what is going to be scrolled on screen next. In the example shown in **Figure 16**, even though the list of items could virtually be unlimited, in the case of Foodtruck Finder it is usually somewhere between 20 and 50 items, only seven views will be needed to display them and assure smooth scrolling.

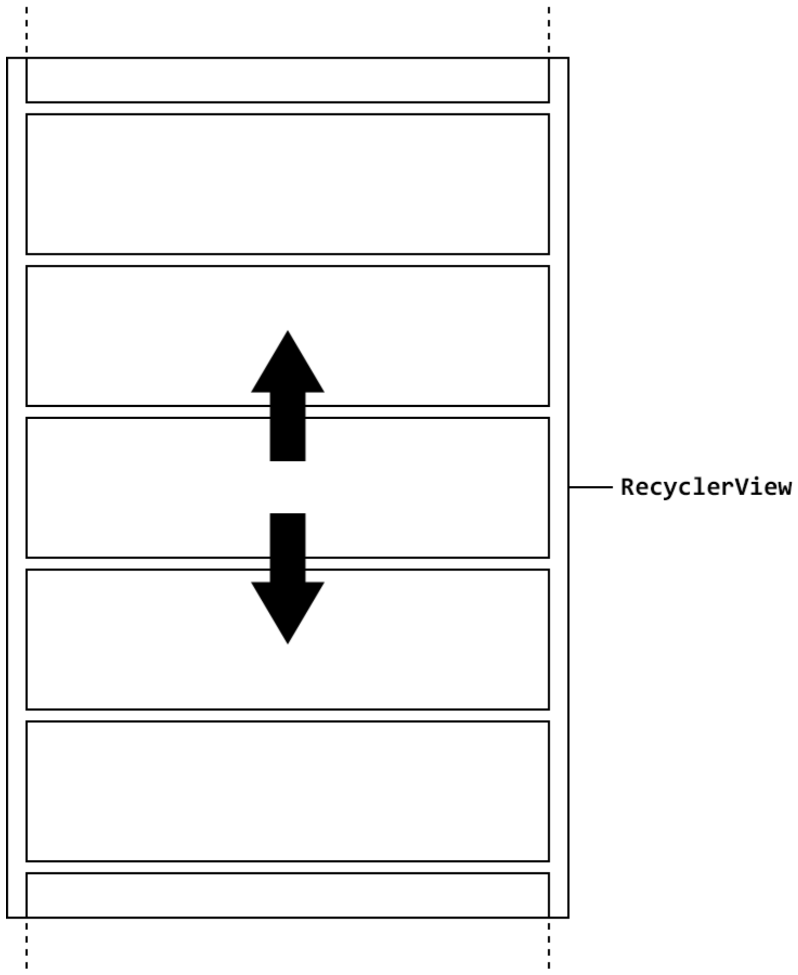


Figure 16 Views used for RecyclerView

To implement this behavior, RecyclerView enforces the use of an Adapter that uses the ViewHolder pattern. An adapter in android is a controller object that connects a view to a data source, which in this case is Foodtruck Finder's ContentProvider. A ViewHolder is a relatively simple class that holds references to the views used to display the contents of an item.

```
1 class DividerItemViewHolder extends RecyclerView.ViewHolder {
2     TextView itemHeader;
3
4     DividerItemViewHolder(View itemView) {
5       super(itemView);
6       itemHeader = (TextView) itemView.findViewById(R.id.list_item_header);
7     }
8 }
```

Figure 17 ViewHolder for a divider item in MainActivity

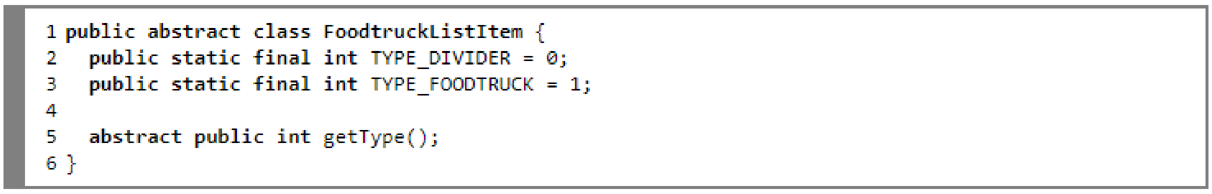
The code example shown in **Figure 17** shows the ViewHolder class used for the items that divide the food trucks list into different groups based on their availability. It holds a single TextView called itemHeader in a field (l. 2), which is assigned using an id defined in the views layout file (l.6). The adapter is responsible for creating the ViewHolder and binding it to the relevant data. When implementing the adapter, the two main methods to override are onCreateViewHolder and onBindViewHolder. As shown in **Figure 18**, In onCreateViewHolder a reference to the view is created using its layout file, which then is used to create a new ViewHolder object that is returned by the method. In onBindViewHolder the adapter then accesses the field defined in the ViewHolder and modifies the values of the associated view, in this example, it sets the text of the TextView to TODAY.

```
 1 public class FoodtruckAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder> {
 2
 3   ...
 4
 5   @Override
 6   public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
 7     View view = LayoutInflater.from(parent.getContext())
 8                             .inflate(R.layout.item_foodtruck_divider, parent, false);
 9     return new DividerItemViewHolder(view);
10   }
11
12   @Override
13   public void onBindViewHolder(RecyclerView.ViewHolder viewHolder, int position) {
14     ...
15     holder.itemHeader.setText("TODAY");
16   }
17
18   ...
19
20 }
```

Figure 18 FoodtruckAdapter Line 6 - 9 ViewHolder is created from layout file, line 15 the view held by the ViewHolder is modified

Every time a view is scrolled off screen, RecyclerView will recycle that view and use it again for a view that is being scrolled on screen, at the other end of the display, by calling the adapter's `onBindViewHolder` method. This results in a smooth scrolling experience, even for a large number of items.

As previously mentioned, the RecyclerView in MainActivity uses groups to separate items into categories based on their availability, like available today or tomorrow. To do this Foodtruck Finder uses different item types that are displayed in the same list, one to display the food trucks and one to display the dividers between the groups. This is achieved by creating an abstract model class FoodtruckListItem, that can be seen in **Figure 19**, and two classes extending this class, FoodtruckItem and DividerItem. The extending classes implement the `getType` method from the parent class and return their respective type, used to identify them later on. Furthermore each class has a set of fields to hold the information needed to display each item, for the divider items that is a single field for the date text, for example today or tomorrow, for the food truck items this holds data like the name of the truck, its distance to the user or the URL to the trucks logo. When a cursor object is passed to the adapter, it will parse the data from the cursor according to the algorithm shown in the previous section (figure 9) and create a list of FoodtruckListItems holding both the DividerItems and the FoodTruckItems. This list is then accessed by the adapters `onBindViewHolder` method when associating the data with their respective views.



```
public abstract class FoodtruckListItem {
  public static final int TYPE_DIVIDER = 0;
  public static final int TYPE_FOODTRUCK = 1;

  abstract public int getType();
}
```

Figure 19 Abstract class FoodTruckListItem

To connect to the data source, Android offers so-called loaders. Loaders are used to asynchronously fetch data from a data source, without the need to implement code to handle concurrency, as they will automatically run on a separate thread and notify an activity over callback methods. Foodtruck Finder uses a CursorLoader, which is specifically designed to connect to a content provider. The CursorLoader will return a cursor object pointing at the relevant data, that will be given to the adapter to process. The CursorLoader automatically registers an observer that will trigger a reload of the data when a change is detected. This is one of the biggest benefits of using a content provider instead of connecting to the database directly, as in the latter case a loader would've had to be implemented to handle this functionality, while the cursor loader provides this functionality out of the box. To use a CursorLoader an activity has to implement the LoaderCallbacks interface and implement its callback methods, `onCreateLoader`, `onLoadFinished` and `onLoaderReset`. In `onCreateLoader` a query has to be created to instantiate one or more loaders. When the loader returns the queried data, `onLoadFinished` is called. Here the processing of the returned data is implemented, in the case of MainActivity it will pass the returned data to the adapter connected to the RecyclerView and trigger it to reload its content. `onLoaderReset` is called when a loader is reset and its data is made unavailable. In this case, MainActivity passes a value of null to the adapter.

Foodtruck Finder's search functionality is triggered using the floating action button in the bottom right of the app, it then hides the content of the main toolbar and replaces it with a search field. Upon typing into the search field the RecyclerView instantly refreshes its view with the corresponding responses. This happens through the SearchViewListener attached to the search view, whose callback method `onQueryTextChange` will return a new cursor object from the ContentProvider after every entered character and pass it to the adapter, which parses the data as described above.

The drawer that can be swiped in from the right of the screen uses the same pattern, a RecyclerView with adapter and ViewHolder, to display the categories used to filter the food truck list. Selecting a filter works analogous to executing a search, and will also trigger the RecyclerView to instantly reload its content.

When an entry in the food truck list is clicked, two intents are created and started. The first intent will start a service, that will fetch the details for the selected operator from Foodtruck

Deutschland's API, the second intent will launch DetailActivity, passing an identifier for the selected operator with the intent.

6.4.3 DetailActivity

There are multiple ways that DetailActivity can be launched, the most obvious is as just mentioned, clicking an entry in MainActivity. A second way to open DetailActivity is from within MapActivity. Trucks are marked by map markers within MapActivity that when clicked on will show an info window above the marker, displaying a truck's schedule, a click on this info window will launch DetailActivity showing the details of the truck that was marked on the map. A third option to open a trucks details is provided using the app indexing API provided by Play Services. Every time a DetailActivity for a certain operator is opened, a call to the app indexing API is made that registers the current operator with the API, making the current operator searchable with the Google Search app on the device.

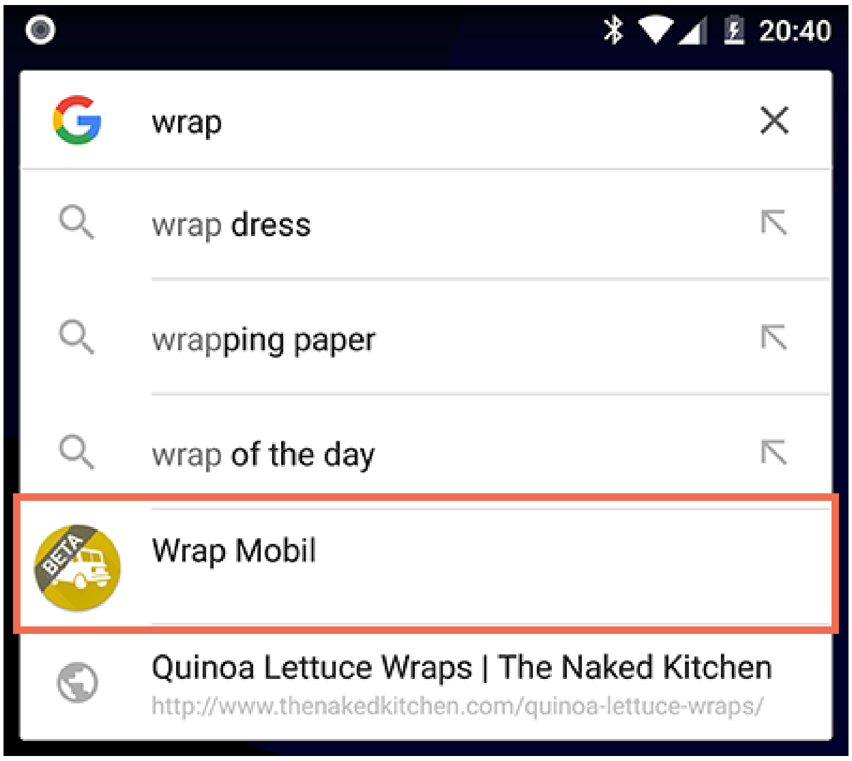

20:40
wrap
wrap dress
wrapping paper
wrap of the day
BETA
Wrap Mobil
Quinoa Lettuce Wraps | The Naked Kitchen
http://www.thenakedkitchen.com/quinoa-lettuce-wraps/

Figure 20 Search result on device, after operator has been registered with app indexing

As can be seen in **Figure 20**, the food truck that has been registered with app indexing can now be searched for on the user's device. This works by defining a unique URL for each operator, that will be given to the app indexing API, that Foodtruck Finder has registered for being able to handle on the device, using an intent filter. When the entry is launched from the search results, the operating system will send an intent, that will trigger Foodtruck Finder to launch DetailActivity.

For DetailActivity to know which operator to launch the identifier of an operator has to be passed with the intent that was started to launch DetailActivity. When launched from within the app, from MainActivity or MapActivity that ID is passed using so-called intent extras that are given to the intent before it is launched. If the activity is launched from the search result, the id of the operator is part of the URL and will be extracted from there. Using the passed operator id, DetailActivity will call the ContentProvider to access details information of the operator, and if the truck is currently active, location and schedule information.

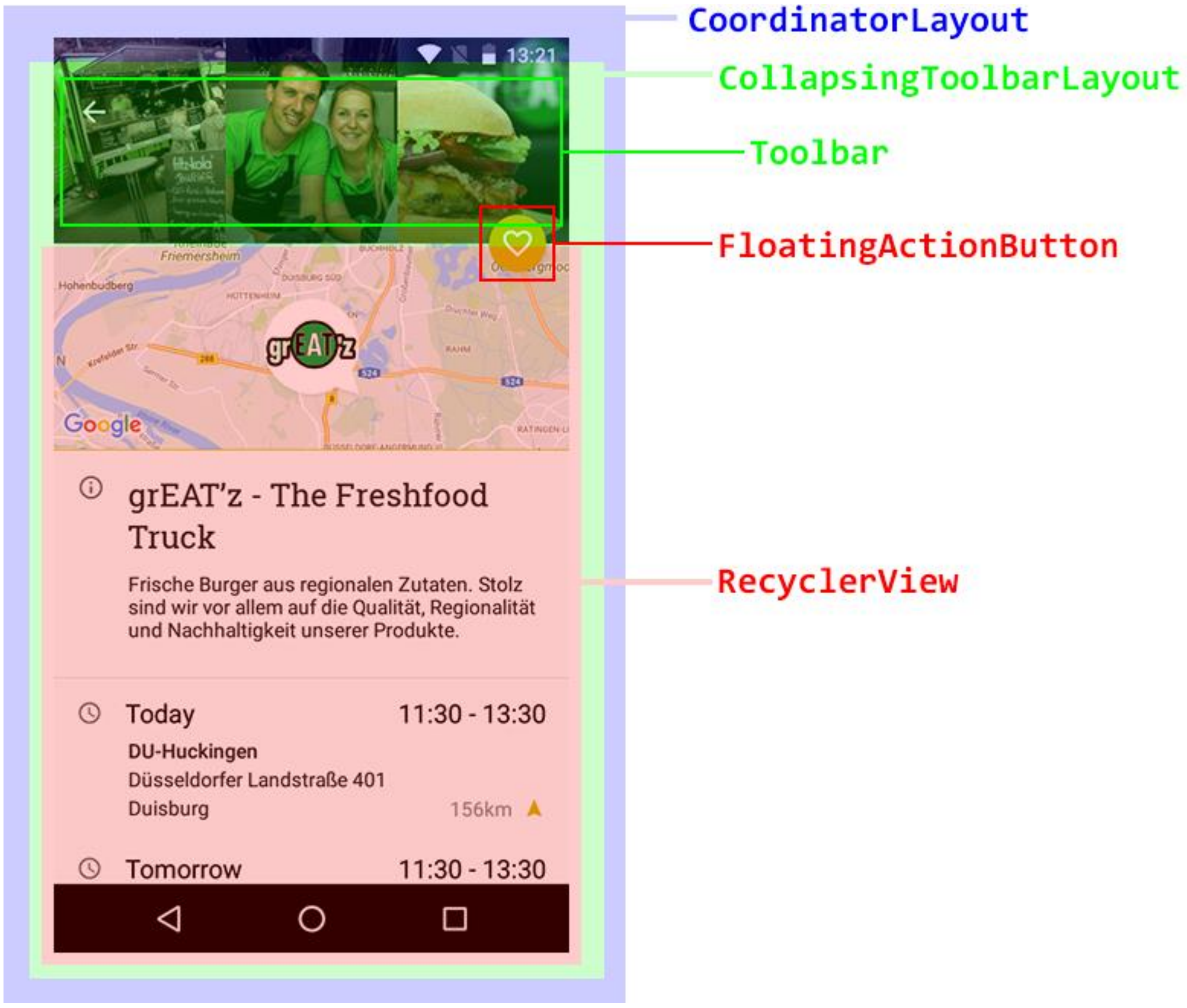

CoordinatorLayout
CollapsingToolbarLayout
Toolbar
FloatingActionButton
RecyclerView
13:21
grEAT'z - The Freshfood Truck
Frische Burger aus regionalen Zutaten. Stolz sind wir vor allem auf die Qualität, Regionalität und Nachhaltigkeit unserer Produkte.
Today
11:30 - 13:30
DU-Huckingen
Düsseldorfer Landstraße 401
Duisburg
156km
Tomorrow
11:30 - 13:30

Figure 21 Layout of DetailActivity

Similar to MainActivity, DetailActivity uses nested layout views for its layout, as can be seen in **Figure 21**, it uses a CollapsingToolbarLayout inside of a CoordinatorLayout. The CollapsingToolbarLayout allows it to have a bigger toolbar that shrinks in size when the layouts inner content is scrolled upwards. Using a set of flags that can be set to influence the layouts scrolling behavior, the toolbar is set to reappear as a regular sized toolbar when scrolled back down, exposing the back button to the user without having to scroll all the way down.

Anchored to the toolbar is a FloatingActionButton that allows the user to save the current food truck to favorites. At its current implementation, the favorite functionality allows the user to view only favorites in MainActivity by using the corresponding button in the filter toolbar.

DetailActivity's main content is displayed in a RecyclerView, using the same concept for listing different item types, that was described in the previous section. The RecyclerView in DetailActivity lists five different item types that are all derived from the same abstract class that can be seen in **Figure 22**.

```java
1 public abstract class DetailsItem {
2   public static final int TYPE_MAPVIEW = 0;
3   public static final int TYPE_OPERATOR_DETAILS = 1;
4   public static final int TYPE_SCHEDULE_ITEM_DATE = 2;
5   public static final int TYPE_SCHEDULE_ITEM_LOCATION = 3;
6   public static final int TYPE_DIVIDER = 4;
7
8   abstract public int getType();
9 }
```

Figure 22 Abstract class used for the items in DetailActivity, line 2-6 defines identifiers for the different classes that extend the abstract class. Each item will return its corresponding identifier by implementing the getType method in line 8.

The RecyclerView will list the MapView, the operator's details, and the schedule in this order using the divider views to group the sections.

The MapView is part of the Google Maps API that is used via Google Play Services. As described in [GoogleDevMapView] it provides *"a View which displays a map (with data obtained from the Google Maps service)"*. A MapView can be used like a fully functioning map, that can be panned and zoomed in, just like Google Maps itself, however, DetailActivity hides all UI controls for the MapView and loads the map using an option called lite mode. With lite mode enabled, MapView will load a static image of the location specified, instead of a fully interactive map. Test results during development showed that this marginally improved performance when opening DetailActivity, the main reason behind using lite mode, however, was the constrained size of the MapView, where interacting with the map would simply not make much sense. Furthermore being able to interact with the map would interfere with the scrolling behavior of its layout container. Upon clicking on the MapView, MapActivity will be launched, taking the user to a full-screen map, pointing at the location that was displayed in the MapView. If the truck being displayed is not currently on the road, the map will be displayed in a grayscale theme, that can be set via the GoogleMapOptions. Additionally clicking the map will be disabled, as MapActivity would launch pointing at a location without an active food truck.

As previously mentioned the operator's details by default will show its name and a descriptive text about what it has to offer, and only if an operator is registered with Foodtrucks Deutschland's premium service its contact links like email, phone, website or social media.

The entries in the schedule are divided into an item for the date and an item for the location to allow grouping multiple locations with the same date and time together, in case an operator has multiple trucks active in the same timeslot. The location sections include the distance of a truck to the user, that is marked with a navigation icon, which will launch an intent to open a navigation app when clicked, providing the user a point to point navigation to the selected location.

6.4.4 MapActivity

MapActivity provides a full-screen map to the user, that marks food truck's locations using a set of custom markers. It makes use of the Google Maps Android API Utility Library to manage the large amount of markers required for displaying all active food trucks. As of the writing of this, around 150 food trucks are active per day and around 800 throughout the week, loading that amount of images every time MapActivity launches would be quite resource intensive. Using the library markers are automatically only loaded when they come into view, which significantly improves performance when first launching MapActivity or zooming out to see a wider area. Furthermore, since displaying a lot of markers at a certain zoom level would make distinguishing markers impossible, MapActivity makes use of so-called MarkerClusters that are also provided by the maps utility library. Instead of displaying a marker pointing to a specific truck, when clustered a marker will point to an area where multiple trucks are located and show the number of trucks located in that area on the marker.

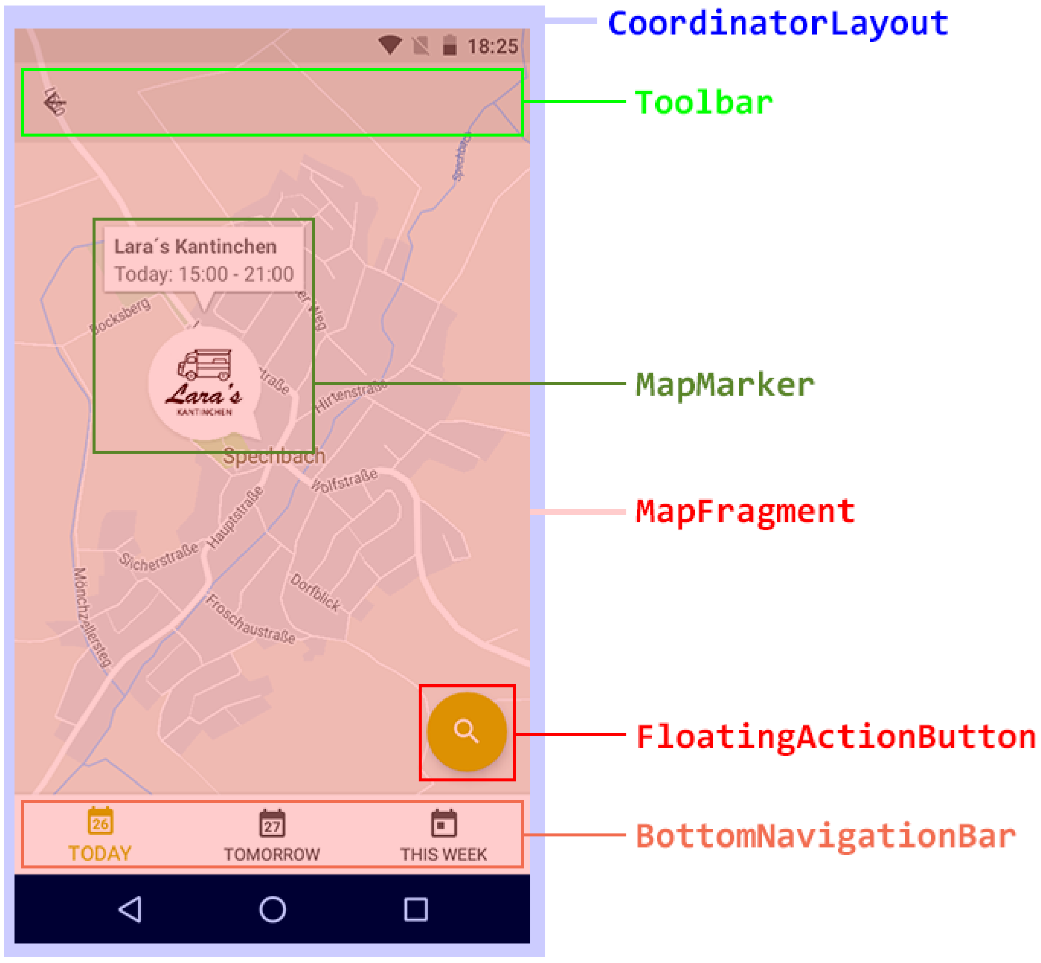


Figure 23 Layout of MapActivity

As can be seen in **Figure 23**, the layout of MapActivity consists of a CoordinatorLayout that holds a MapFragment and some UI elements. As described in [GoogleDevMapFragment] a MapFragment is a MapView, wrapped in a fragment that automatically handles the lifecycle of the map. The MapFragment is set to fill the screen completely, providing a more immersive experience for the user. This is emphasized by using a transparent toolbar at the top, that changes to a search field when the search is triggered using the FloatingActionButton in the bottom right corner. This intentionally uses the same design pattern as MainActivity does, to introduce familiarity within the app.

To allow the user to switch between the availabilities of the locations, MapActivity features a BottomNavigationBar that can be used to apply the same grouping categories that are used in MainActivity, today, tomorrow or this week. Upon selecting an entry in the BottomNavigationBar, MapActivity will seamlessly reload the MapMarkers for the selected availability range.

Loading location data to place the markers in MapActivity works similar to loading food truck entries in MainActivity, by using a CursorLoader that connects to the ContentProvider. When the availability range is changed using the BottomNavigationBar, the selection made in onCreateLoader is changed and the loader is restarted. MapActivity creates and sets up the required markers in onLoadFinished, so whenever the loader is returned it will automatically create a new set of markers and replace the ones that had previously been created.

The data queried to create a marker consist of the locations coordinates, identifiers to match the images used for the markers to the location, and information about the truck, like its name and the schedule for the currently selected date range. While the images used for the cluster markers are created during runtime, the images used for the markers are packaged with the application and only created during runtime if a location was not found within the packaged images. This has been done to further improve the performance of MapActivity, as it is crucial that the map feels smooth and fluid when panning or zooming in.

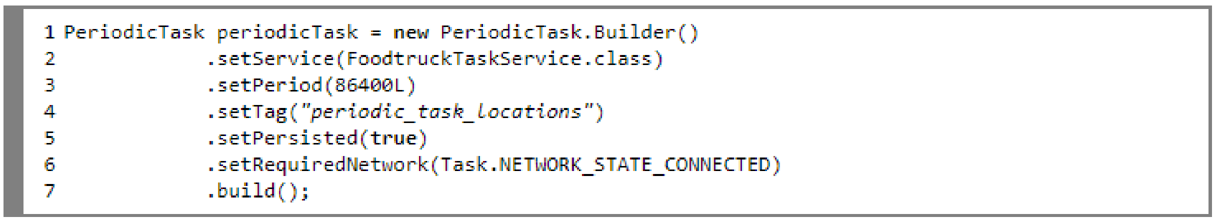
The schedule information for each location is placed in the info window of the marker as can be seen in **Figure 23**. Furthermore as previously noted, a click on a marker’s info window will launch the corresponding DetailActivity.

MapActivity can either be launched from clicking on the MapView in DetailActivity or from selecting the map icon in the toolbar of MainActivity. Depending on from where and how it is launched it behaves slightly different. When launched from DetailActivity, the intent created to launch MapActivity will carry the longitude and latitude of the location that was shown in DetailActivity, as well as the date range that the location is active in. This will trigger MapActivity to center and zoom on the given location and select the date range that the truck is active in the BottomNavigationBar. Additionally, the marker of the location that came with the intent will be set to show its info window automatically when MapActivity is opened. When launched from MainActivity the latitude and longitude given to the intent will point to the users last known location, that was saved in the SharedPreferences. There will be no date range given and the selection for the query will default to today, so will the selected entry in the BottomNavigationBar.

When searching in MapActivity, search results are returned in order of distance to the user, selecting a search result will trigger the map to pan to the selected location. The search uses the same implementation as MainActivity and will return results matching either a food truck's name or offer.

6.5 Service

As described in the previous chapter, Foodtruck Finder uses a background service to fetch data from Foodtrucks Deutschland. The service is implemented using a GcmTaskService, which is used to schedule periodic tasks using Google Play Services. GcmTaskService connects to the GcmNetworkManager an API that according to [GoogleDevGcm] is used to “schedule network-oriented tasks, and let Google Play services batch network operations across the system”, which means the tasks created will be run by Play Services in a certain time frame, letting the system decide when to execute the tasks. Using the GcmNetworkManager Foodtruck Finder does not need to provide logic that would reschedule a task in case of failure, as Play Services will automatically take care of that. Foodtruck Finder schedules two tasks when MainActivity is launched. One is set to update the truck locations every 24 hours, the other is set to update the truck operators every seven days.



```
PeriodicTask periodicTask = new PeriodicTask.Builder()
            .setService(FoodtruckTaskService.class)
            .setPeriod(86400L)
            .setTag("periodic_task_locations")
            .setPersisted(true)
            .setRequiredNetwork(Task.NETWORK_STATE_CONNECTED)
            .build();
```

Figure 24 Creating a PeriodicTask

Figure 24 shows the creation of the PeriodicTask to update the location data every 24 hours., The service responsible for responding to the scheduled task is FoodtruckTaskService, which is set using `setService` ([PeriodicTask] l.2). The period in which the task should repeat is set in seconds using `setPeriod` ([PeriodicTask] l.3). The tag set using `setTag` ([PeriodicTask] l.4), is used by GcmNetworkManager to identify the task. GcmNetworkManager will only keep one task with the same tag active at a time, which is important because the method to create the task is called every time MainActivity is launched, however, it will not add a new task if one already exists. By setting `setPersisted(true)` ([PeriodicTask] l.5) it is made sure that the task does not get deleted on a device reboot. The final option used is `setRequiredNetwork` ([PeriodicTask] l.6), which is set to use any network available, Foodtruck Finder's required data is usually in the range of kilobytes, and any network should be sufficient, however, if in case larger amounts of data was required it would make sense to set this option to only run the task on a WiFi connection.

When GcmNetworkManager sends out a message to execute a scheduled task, FoodtruckTaskService will respond by running its onRunTask callback method. Here a connection to Foodtrucks Deutschland's API is created using OkHttp. Using OkHttp here reduce the amount of boilerplate code that needs to be written from managing HTTP connections. Additionally, OkHttp has the benefit that it automatically *"silently recovers from the most common connection problem"*, as described on their website [OkHttp].

Upon successful connection to Foodtrucks Deutschland, the returned JSON data is parsed and added to the database via the ContentProvider.

One downside of using GcmNetworkManager to execute tasks is that there is no guarantee that a task would be executed immediately, however, Foodtruck Finder requires that the location data is refreshed without delay when MainActivity is launched. To handle this problem, Foodtruck Finder uses an IntentService that manually instantiates a FoodtruckTaskService object, and calls its onRunTask method directly, which can be launched from any activity by creating an intent and calling the startService method.

6.6 Data Storage

As previously mentioned, using a ContentProvider brings certain benefits when working with a RecyclerView and loaders. However, usually the downside of using a ContentProvider is having to implement a ContentProvider, which can be a task just as time-consuming as implementing a custom loader. Implementing a ContentProvider is done by creating a class that extends the abstract class ContentProvider, implement an SQLiteOpenHelper class that handles the creation of the underlying database as well as one or more classes in the form of a Contract, that defines the names of tables and columns of the database. The ContentProvider class has to implement methods that handle querying and updating the database, like query, insert, update or delete, and defines URIs for the endpoints used to access the database. It needs to take care of matching the various database update methods to the URIs of the endpoints. The URIs used by a ContentProvider are in the form: content://<CONTENT AUTHORITY>/<ENDPOINT>, with content authority being used to uniquely identify the ContentProvider on the device. When an app consuming data from this ContentProvider uses the URI to query for the data, the ContentProvider class has to match the URI to its corresponding table and provide code used to query the database for each request.

To reduce the code needed to implement the ContentProvider, Foodtruck Finder makes use of Schematic, a library that automatically generates a ContentProvider and sets up the database tables. To set up Schematic, it requires three types of Java classes that have to be implemented. First, to create the tables for the databases schematic uses Java classes that define a set of static fields to define the table column names, with Java annotations that set the column properties like key definitions and data types, as can be seen in **Figure 25**. Secondly, to define the database itself, it uses a similar structured class that references the classes for the columns to set up the tables. Schematic will use this to generate classes that make up the contract part of the ContentProvider and instantiate the database.

```
1 public class FavouritesColumns {
2   @DataType(DataType.Type.INTEGER) @PrimaryKey @AutoIncrement
3   public static final String _ID = "_id";
4   @DataType(DataType.Type.TEXT) @NotNull @Unique
5   public static final String ID = "id";
6   @DataType(DataType.Type.INTEGER)
7   public static final String FAVOURITE = "favourite";
8 }
```

Figure 25 Java class used to create a table, Line 3, 5 and 7 define column names, line 2, 4 and 6 set the annotations that set the datatype for the columns.

The third class that needs to be provided is used to define the endpoints for the ContentProvider and create the ContentProvider class implementation. **Figure 26** shows an extract of Foodtruck Finder's ContentProvider that is implemented using Schematic.

```
 1 @ContentProvider(authority = FoodtruckProvider.AUTHORITY, database = FoodtruckDatabase.class)
 2 public class FoodtruckProvider {
 3   public static final String AUTHORITY = "co.pugo.apps.foodtruckfinder.data.FoodtruckProvider";
 4   static final Uri BASE_CONTENT_URI = Uri.parse("content://" + AUTHORITY);
 5
 6   ...
 7
 8   @TableEndpoint(table = FoodtruckDatabase.FAVOURITES)
 9   public static class Favourites {
10     @ContentUri(
11             path = Path.FAVOURITES,
12             type = "vnd.android.cursor.item/dir"
13     )
14     public static final Uri CONTENT_URI = Uri.parse(BASE_CONTENT_URI + "/favs");
15   }
16   ...
17 }
```

Figure 26 Java class used to create ConrtentProvider, Line 3 defines the content authority, line 4 defines the base URI that is used for all endpoints, line 8-15 defines an endpoint for the favorites table

Using the `@ContentUri` annotation, schematic defines the URIs used to access the endpoints for specific database tables. Additionally, to what can be seen in **Figure 26** (l. 11-12) for the path and the return type, schematic supports adding SQL specific functions like join, group by and order by to the annotations. The given type, here `vnd.android.cursor.item/dir`, defines the type that the ContentProvider returns for the given endpoint, which in this case will return a list of items, meaning multiple rows are returned by the underlying database query. To only return a specific set of rows, schematic uses the `@InexactContentUri` annotation, that basically works the same way, except defining a where clause that will be used in conjunction with a parameter given to the URI.

6.7 App Engine Web App

As mentioned earlier, the web app running on Google App Engine is created using an implementation of a Java Servlet. The servlet is accessed by sending an HTTP get request to the servlet's URL, which is in the form `http://<project_id>.appspot.com/ftd`. At a minimum, it requires one parameter called fetch to be sent with the request, that defines which request should be fetched from Foodtruck Deutschland's API. For example adding `fetch=getLocations.json` will request the location information from Foodtrucks Deutschland. Furthermore, any number of parameters can be added to the request, that will be passed to Foodtruck Deutschland's API, for example, getLocations can have a parameter date that will determine if a single day or a whole week of location data is returned.

When a request is sent to the servlet, it will first check if the requested file is stored in its Memcache from a previous request. If that is the case, it will retrieve the file from its Memcache and return it to the sender, if not it will connect to Foodtruck Deutschland's API, retrieve the file from there, and store it in Memcache, before returning it.

App Engine's Memcache is a basic key-value store. The key that is used to store each file will be build from the file name that was sent with the fetch parameter concatenated with the values from any additional parameters that may have been sent with the request. For the value, the servlet will compress the returned JSON data before it is stored, to stay under the Memcache size limit of one megabyte.

For testing purposes the servlet can also receive a parameter named update with the request, that will force the servlet to retrieve new data from Foodtrucks Deutschland instead of checking in Memcache first.

As already mentioned, in its current implementation the servlet uses a version of Memcache that is shared with other apps running on app engine and is not persistent, meaning the servlet has no control over how long the data is stored in memory, as app engine will clear caches when other apps using the same shared memory require it. Initial tests have shown that the data stored in Memcache persisted on average for about 40 minutes. These tests have been run with a single user accessing the servlet, the resulting duration might change if more users access the servlet.

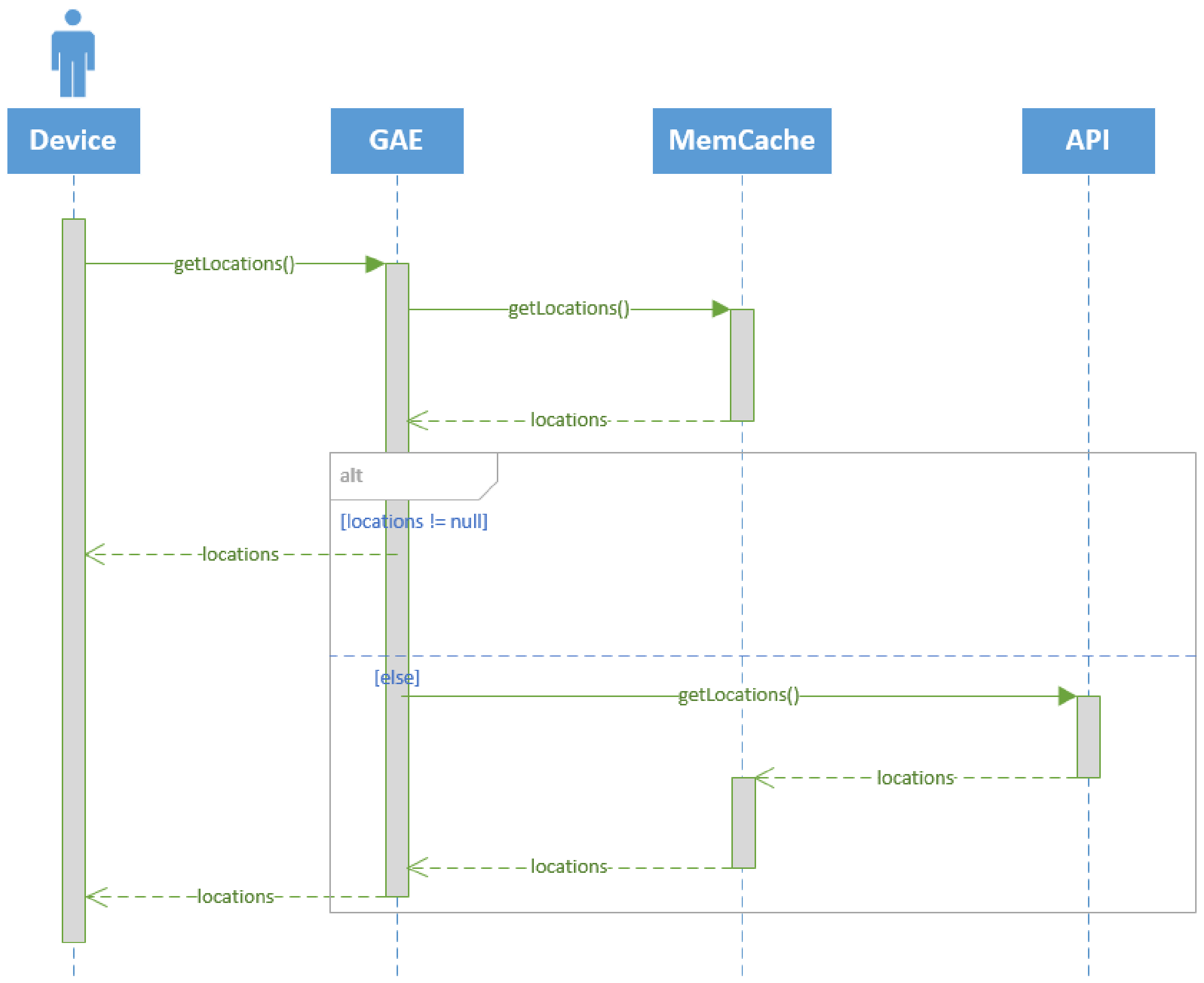


Figure 27 Sequence of events connecting to API through App Engine

As can be seen in **Figure 27**, when a device sends a request to Google App Engine, it checks if the data for the request has been stored in Memcache from a previous request, if it finds data stored in memory it will return the data, if not App Engine will send a request to the API, store the data returned from the API in its Memcache and send a response containing the data.

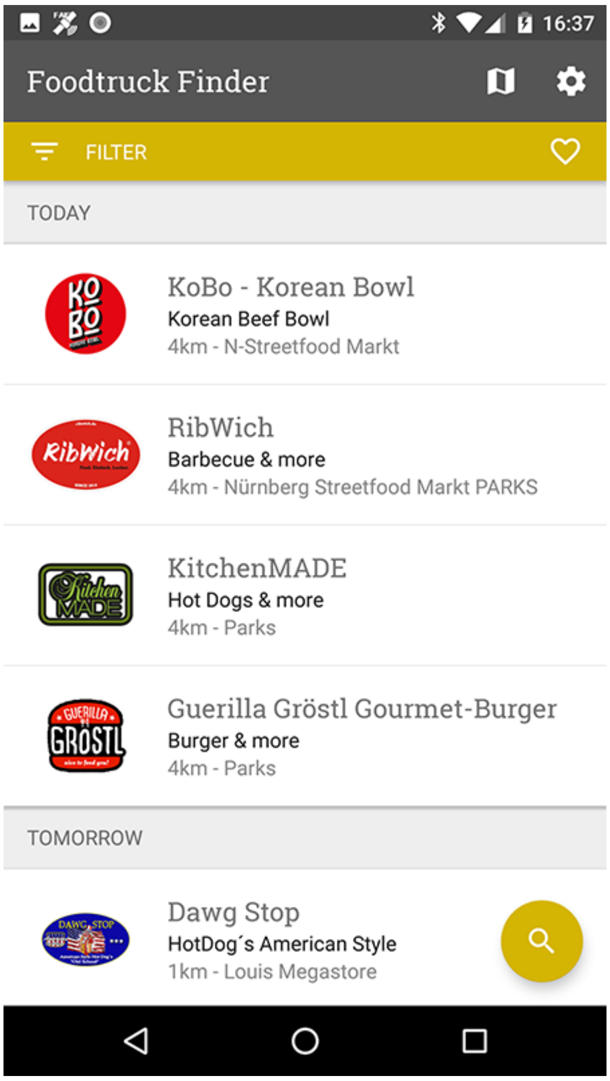
When testing the servlet with requesting a week's worth of location data, which is the largest data request that Foodtruck Deutschland's API provides, an average request to the servlet took 180 milliseconds, including the initial request, where App Engine does not find any data in its Memcache, that took about 550 ms. Tested on the same internet connection, the average request directly to the API took almost twice as long with 350 milliseconds.

As data from the existing Foodtrucks app shows, usage heavily peaks between 9 am and 1 pm in around lunchtime, as well as in the evening around 7 pm, it is expected that during those times app engine will hold the data almost consistently in its memory, taking large amounts of load from the API and providing consistent speeds to the users.

7 Using Foodtruck Finder

Since Foodtruck Finder is focused on letting its users find something to eat, it is obvious that its usage peaks around lunch or dinnertime. During those times, the information that people are looking for is simply, what is available around me, and how far is it away. Foodtruck Finder wants to give users exactly this information, the second they open the app. By letting users set the radius in which trucks are listed, users have the ability to set the distance they are willing to drive to get something to eat, which limits the results shown to only relevant items. If however, a user would want to look for something further away, Foodtruck Finder has the added possibility of seeing all the available locations on a map. Considering this, there are two major usage scenarios for using Foodtruck Finder. The first one is, looking for food nearby, at the moment the app is being used.

Upon opening the app, the user will be presented with a list of currently available food trucks, that are in the user's defined range. In the example shown in **Figure 28**, there are four entries available to the user.



KoBo - Korean Bowl
Korean Beef Bowl
4km - N-Streetfood Markt

RibWich
Barbecue & more
4km - Nürnberg Streetfood Markt PARKS

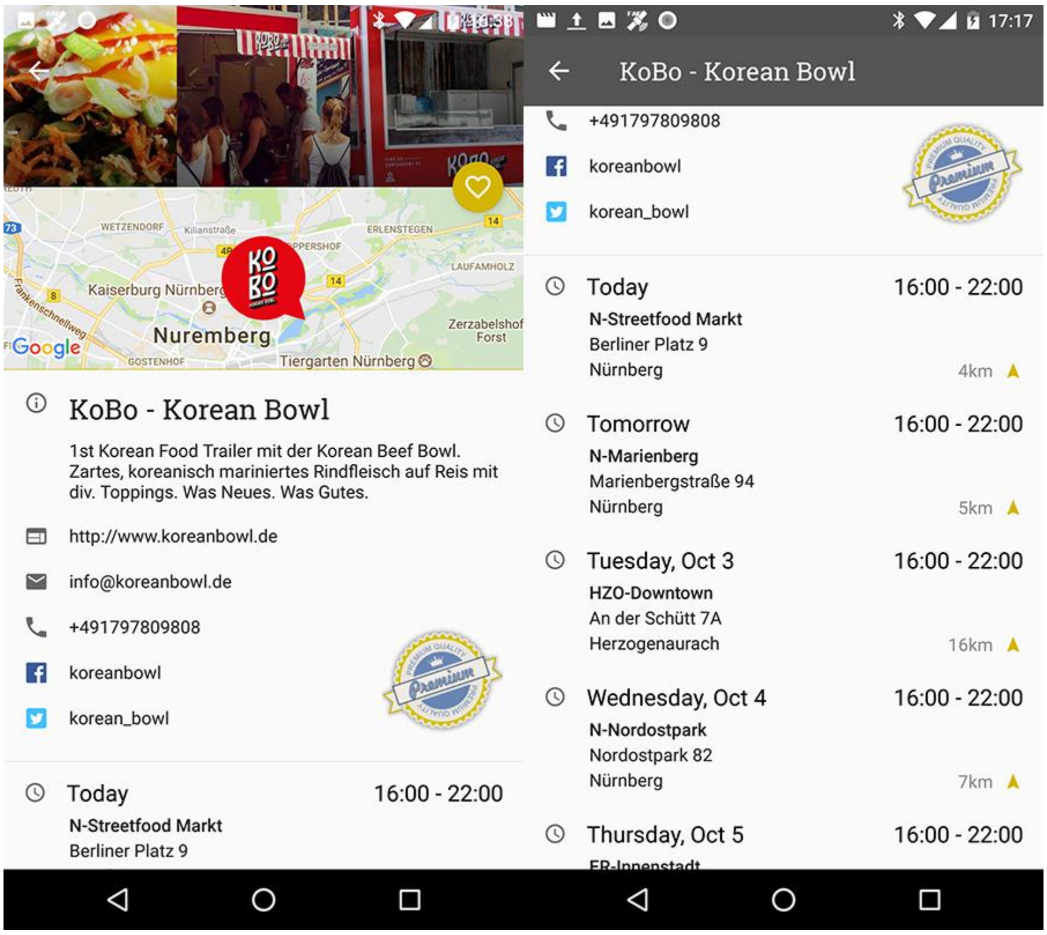
KitchenMADE
Hot Dogs & more
4km - Parks

Guerilla Gröstl Gourmet-Burger
Burger & more
4km - Parks

Dawg Stop
HotDog´s American Style
1km - Louis Megastore

Figure 28 Foodtruck Finder showing four active trucks

Assuming one of the entries has peaked the user's interest, upon selecting the entry Foodtruck Finder takes the user to the details page of the selected entry, as demonstrated in **Figure 29**. Here the user gets more information about what he can expect, and where exactly the truck is located. Furthermore, the shown schedule instantly lets the user know that the truck is open today until ten o'clock, and after scrolling upwards the schedule for the rest of the week reveals upcoming dates and locations. If the user would want to visit the location right away clicking on the distance, marked by the navigation symbol will open point to point navigation on the device, in the user's preferred navigation app.


KoBo - Korean Bowl

1st Korean Food Trailer mit der Korean Beef Bowl. Zartes, koreanisch mariniertes Rindfleisch auf Reis mit div. Toppings. Was Neues. Was Gutes.

http://www.koreanbowl.de

info@koreanbowl.de

+491797809808

koreanbowl

korean_bowl

Today 16:00 - 22:00
N-Streetfood Markt
Berliner Platz 9

KoBo - Korean Bowl

+491797809808

koreanbowl

korean_bowl

Today 16:00 - 22:00
N-Streetfood Markt
Berliner Platz 9
Nürnberg 4km

Tomorrow 16:00 - 22:00
N-Marienberg
Marienbergstraße 94
Nürnberg 5km

Tuesday, Oct 3 16:00 - 22:00
HZO-Downtown
An der Schütt 7A
Herzogenaurach 16km

Wednesday, Oct 4 16:00 - 22:00
N-Nordostpark
Nordostpark 82
Nürnberg 7km

Thursday, Oct 5 16:00 - 22:00

Figure 29 Details for selected food truck, schedule on left side is shown after scrolling

A second usage scenario could be, a user is going to visit a certain location and want's to see what is available at that location ahead of time. The user can now either use the map to check what's available at the desired location or go into the settings and set a custom location to be used. As displayed in **Figure 30**, by using the integrated map, the user can browse to the desired location and switch to the “this week” tab in the bottom bar to see all available locations for the upcoming week. Tapping one of the locations will display the entry's next available date in the locations info window. To get to the locations detail page the user simply has to tap the info window that displays the schedule.

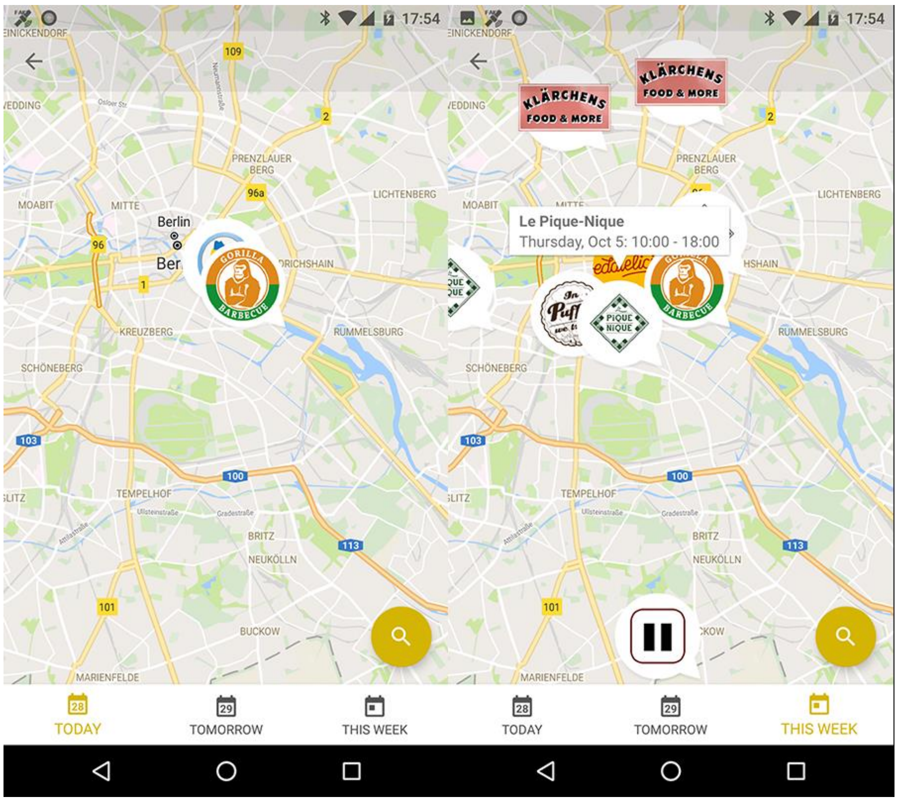
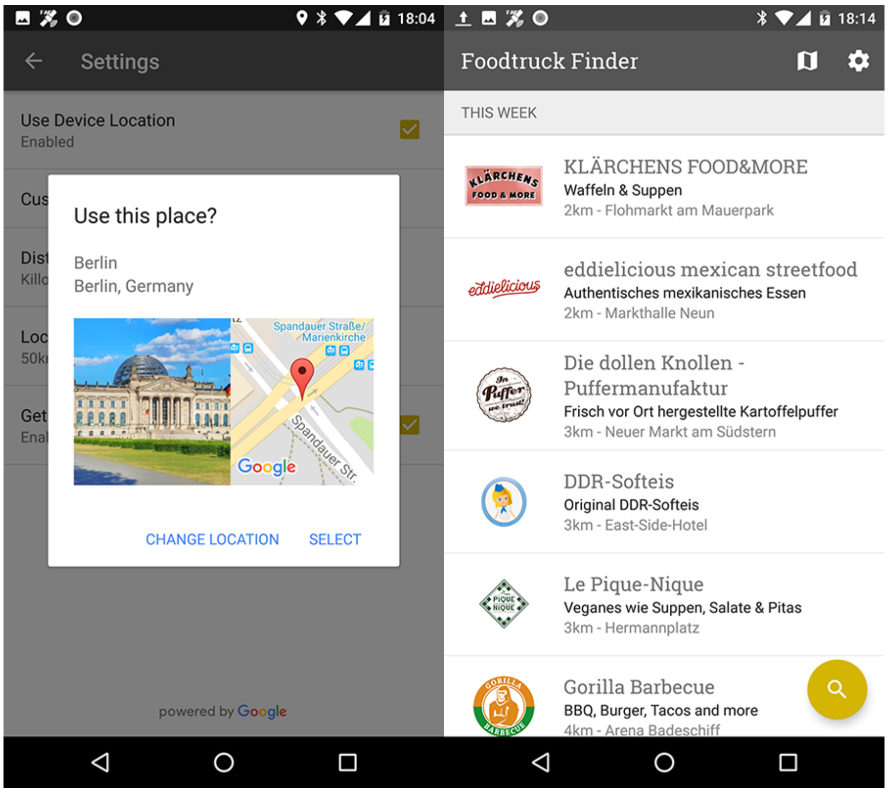


Figure 30 Integrated map, showing available items today and for the current week.

Alternatively, the user can select the upcoming destination as the custom location in Foodtruck Finder's settings, which will immediately trigger the list of food trucks to reload with locations in the radius of the selected area, which is shown in **Figure 31**.


Settings

Use Device Location
Enabled

Use this place?
Berlin
Berlin, Germany

CHANGE LOCATION SELECT

powered by Google

Foodtruck Finder

THIS WEEK

KLÄRCHENS FOOD&MORE
Waffeln & Suppen
2km - Flohmarkt am Mauerpark

eddielicious mexican streetfood
Authentisches mexikanisches Essen
2km - Markthalle Neun

Die dollen Knollen - Puffermanufaktur
Frisch vor Ort hergestellte Kartoffelpuffer
3km - Neuer Markt am Südstern

DDR-Softeis
Original DDR-Softeis
3km - East-Side-Hotel

Le Pique-Nique
Veganes wie Suppen, Salate & Pitas
3km - Hermannplatz

Gorilla Barbecue
BBQ, Burger, Tacos and more
4km - Arena Badeschiff

Figure 31 Selecting a custom location in the settings

To explore the available locations for this week, the user can scroll up to the “this week” group in the list of trucks, and explore the locations available. As Foodtrucks Deutschland only tracks location information for the ongoing week, this scenario is always limited to that time frame.

8 User Feedback

At the time of writing, a public beta was made available in the Google Play Store, which not only helped to find and fix some occurring bugs but was also a great opportunity to gather some feedback from users. Beta testers have been asked to fill out a survey focused on usability and performance of the app.

When asked to rate how easy it was to find out how to use Foodtruck Finder from 1, could not find anything in the app, to 10, found everything right away. All of the testers that filled out the survey responded with 7 or higher, with close to 50% of the users rating the usability of the app at 9 and 10. **Figure 32** shows the total results.

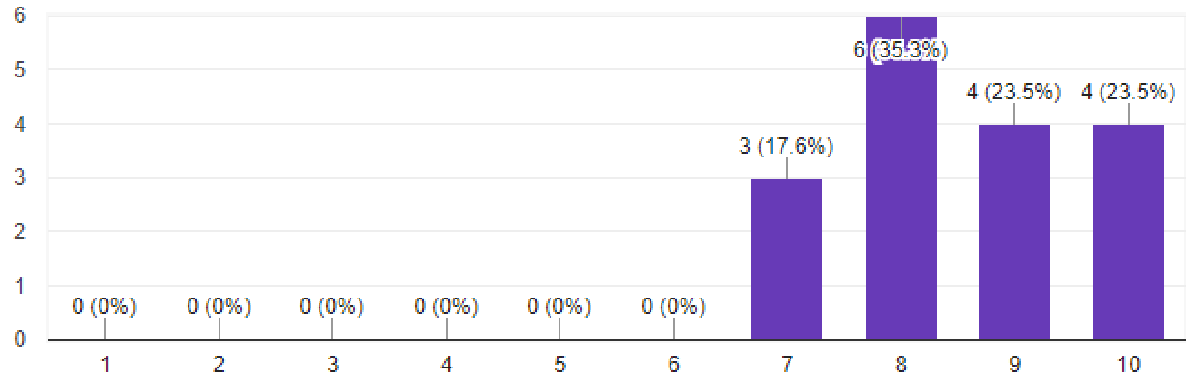


Figure 32 Results user survey, ease of use

Upon a question if the initial radius of 50km set by the app was sufficient to find relevant food trucks, all of the respondents answered yes, however it has to be taken into consideration that most testers were acquired using Foodtruck Deutschlands social media accounts, so people signing up for the beta test are very likely to live in an area with a large food truck presence. This also has to be considered for the answers to the question: how easy was it to find food trucks using Foodtruck Finder, whereas can be seen in **Figure 33** all of the respondents answered with seven or higher. As of the writing of this, the beta has only been active for two weeks, results to specifically this question become a bit more meaningful when the user base grows beyond the initial food truck enthusiasts.

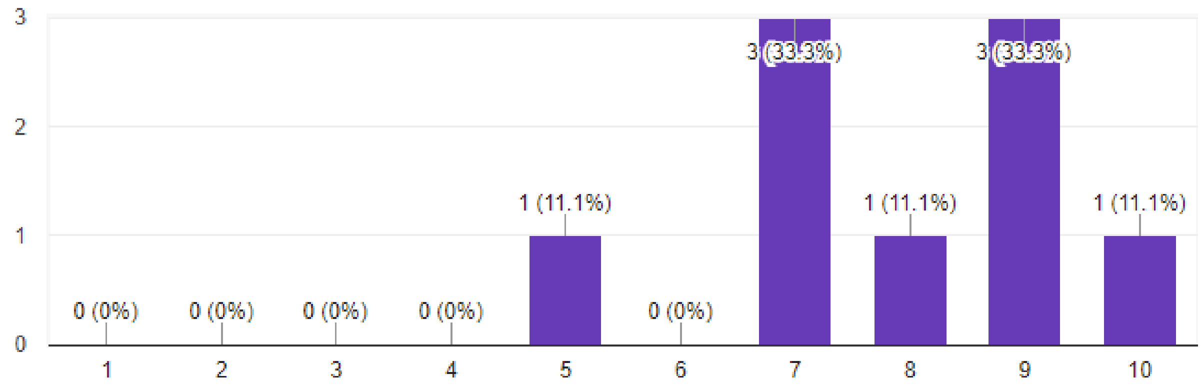


Figure 33 Results user survey, user friendliness

When asked about which of the apps looked better design wise, all of the respondents answered Foodtruck Finder was better designed, as can be seen in **Figure 34**.

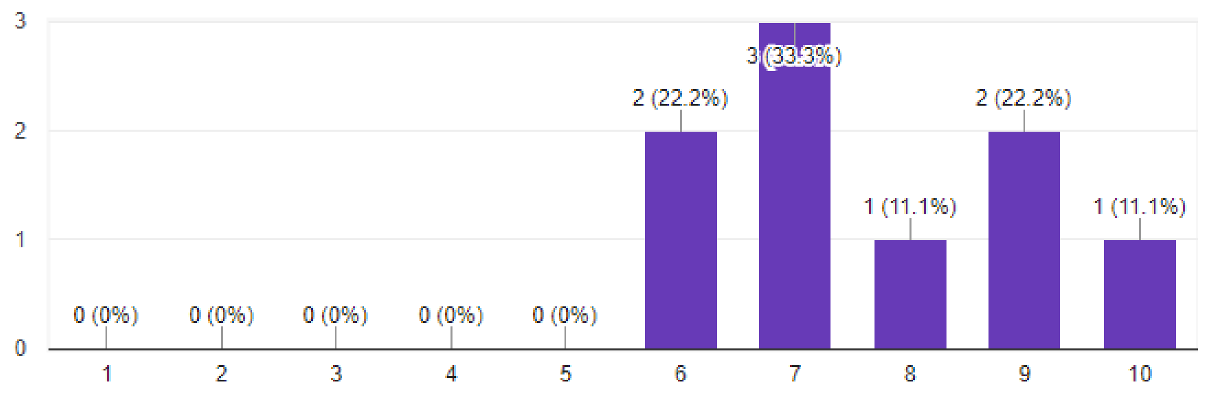


Figure 34 Results user survey, design

Although answers to this questions are very subjective, this highlights how sticking close to recommended design guidelines, and applying familiar design patterns results in an app that users find appealing and enjoy using.

9 Summary

This paper described the implementation of an android application called Foodtruck Finder. Foodtruck Finder provides a way for users to quickly find nearby food trucks. For comparison, an existing app that provides the same functionality as Foodtruck Finder has been analyzed, highlighting mistakes made in its implementation as well as showing functionality that was carried over into the implementation of Foodtruck Finder. Instead of constantly polling for location updates while the app is running, Foodtruck Finder implements a location strategy that is less resource intensive by only using the minimum of location data needed, in the form of the device's last known location. In doing so it provides the same functionality as the existing app, in a more resource friendly way.

Foodtruck Finder retrieves its required data using a background service that connects through a web application running on App Engine to reduce the load on Foodtrucks Deutschland's API and provide improved data transfer speeds to its users. The retrieved data is stored locally on the device using an SQLite database accessed through a ContentProvider, to allow for the use of loaders in combination with a RecyclerView that makes up the main user interface of the app, providing a scrollable grouped list of food trucks in a user's perimeter, that automatically reloads its data when needed. Furthermore, Foodtruck Finder makes use of geofences to provide notifications of nearby food trucks to the user without having to constantly ask for location updates.

As demonstrated by the usage examples in chapter 7, Foodtruck Finder successfully provides its core functionality of finding food trucks, either at the moment of the app usage and in close proximity to the user or on a future date at a different location.

10 Outlook

As mentioned previously, at the writing of this thesis a public beta version of Foodtruck Finder is available in the Google Play Store. In close collaboration with Foodtrucks Deutschland, the app is in active development with a planned release date within a month of this writing.

Although Foodtruck Finder in its current state provides a significant improvement over the existing app, there are quite a few possible ways of improving the user's app experience even further.

One area that is currently considered to be improved is the geofence functionality. For example, by incorporating push notifications into this functionality the app could be aware of changes in schedule and location for the trucks. This change would require server-side changes for Foodtrucks Deutschland which at the writing of this are in active consideration.

Another area where this could be utilized would be the ability to notify the user about food trucks they have previously added to their favorites. For example, if a user would discover a truck that really peaked the user’s interest, the app could suggest to add the truck to the user’s favorites and notify the user once the truck is on the road again.

Another area that could use some improvements is the calculation of the distances to the user, as it stands currently all distances are calculated using a simple beeline, which means actual driving distances to the truck's location differ from the distances shown by Foodtruck Finder. Google offers an API that allows the calculation of distances using point to point driving navigation, however, a number of calculations needed for Foodtruck Finder would exceed the API’s free tier and would require paid access to the API. As Foodtruck Finder currently does not generate any revenue this is not an option, but will be considered in the future.

There are plans to monetize Foodtruck Finder when it is finally released. Those plans are to include advertisement within the app, but also allow users to purchase an in-app payment to use Foodtruck Finder without ads. In the survey given to testers, people were asked how they would feel about this, and if they would pay for the ads to be removed. Results were equally split between people who said, they are ok with ads and people who said they would pay to remove them.

One more improvement that is already in process is improving the experience in MapActivity, there are various features and improvements already in active development. Foremost there is currently an issue present when multiple trucks are listed at the same location, which results in overlapping map markers, where it gets difficult to distinguish locations. There are multiple solutions that are being experimented with, like for example a consolidated, bigger markers that will point to multiple trucks, similar to a marker cluster, but rather than showing the number of trucks it would show all of the trucks logos. Another option that is being experimented with is rotating the markers to point into a different direction, currently the markers point to the bottom right, on a location with multiple trucks there could be four markers pointing into each direction. Tests are sure to show which of these solutions will work best. Additionally, an option to be able to search the map for more than just truck locations is considered, so a user could also search for cities or regions.

Literature

AndroidDashboard	Platform Versions https://developer.android.com/about/dashboards/index.html last download: 28/9/2017
AndroidDevActivity	Android Developer Reference: Activity https://developer.android.com/reference/android/app/Activity.htm l last download: 28/9/2017
AndroidDevBlog	How we’re helping people find quality apps and games on Google Play https://android-developers.googleblog.com/2017/08/how-were- helping-people-find-quality.html last download: 28/9/2017
AndroidDevView	Android Developer Reference: View https://developer.android.com/reference/android/view/View.html last download: 28/9/2017
AndroidDevViewPager	Android Developer Reference: ViewPager https://developer.android.com/reference/android/support/v4/view/ ViewPager.html last download: 28/9/2017
AndroidFundamentals	Android Developer Guide: Application Fundamentals https://developer.android.com/guide/components/fundamentals.ht ml last download: 28/9/2017
CloudComp	Richard Hill, Laurie Hirsch, Peter Lake, Siavash Moshiri (2013): Guide to Cloud Computing - Principles and Practice. London: Springer
ComScore	The 2017 U.S. Mobile App Report https://www.comscore.com/Insights/Presentations-and- Whitepapers/2017/The-2017-US-Mobile-App-Report last download: 28/9/2017
Glide	An image loading and caching library for Android focused on smooth scrolling https://github.com/bumptech/glide last download: 28/9/2017
GoogleDevGcm	Google Developer Reference: GcmNetworkManager

	https://developers.google.com/android/reference/com/google/android/gms/gcm/GcmNetworkManager last download: 28/9/2017
GoogleDevMapView	Google Developer Reference: MapView https://developers.google.com/android/reference/com/google/android/gms/maps/MapView last download: 28/9/2017
GoogleDevMapFragment	Google Developer Reference: MapFragment https://developers.google.com/android/reference/com/google/android/gms/maps/MapFragment last download: 28/9/2017
GoogleDevLocation	Google Developer Reference: Fused Location Provider https://developers.google.com/location-context/fused-location-provider/ last download: 28/9/2017
JavaEE7Tutorial	William Markito, Kim Haase, Ian Evans, Ricardo Cervera-Navarro, Eric Jendrock (2014): The Java EE 7 Tutorial: Volume 1, Fifth Edition Boston: Addison-Wesley
JSON	Introducing JSON http://www.json.org/ last download: 28/9/2017
MaterialGuidelines	Components https://material.io/guidelines/components/ last download: 28/9/2017
Meier	Reto Meier (2012): Professional Android 4 Application Development, 3rd Edition (Chapter 8) Birmingham: Wrox Press
MobileCuisine	Foodtruck Industry Statistics https://mobile-cuisine.com/trends/2015-food-truck-industry-statistics-show-worth-of-1-2b/ Last downloaded: 28/9/2017
Newzoo	Top 50 countries by smartphone users and penetration https://newzoo.com/insights/rankings/top-50-countries-by-smartphone-penetration-and-users/ Last downloaded: 28/9/2017
OkHttp	An HTTP & HTTP/2 client for Android and Java applications http://square.github.io/okhttp/ Last downloaded: 28/9/2017
ProAndroid5	Grant Allen, Satya Komatineni, Dave MacLean (2015): Pro Android 5, 5th Edition New York: Apress
StatcounterMobile	Mobile Operating System Market Share Worldwide http://gs.statcounter.com/os-market-share/mobile/worldwide Last downloaded: 28/9/2017
StatcounterOS	Operating System Market Share Worldwide

	http://gs.statcounter.com/os-market-share Last downloaded: 28/9/2017
StatistaApps	Number of apps available in leading app stores as of March 2017 https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/ Last downloaded: 28/9/2017
StatistaAppDownloads	Number of app downloads per month of smartphone users in the United States as of June 2017 https://www.statista.com/statistics/325926/monthly-app-downloads-of-us-smartphone-users/ Last downloaded: 28/9/2017
StatistaPublishers	Leading mobile app publishers in the United States as of July 2017, by share of total app time spent https://www.statista.com/statistics/235048/share-of-time-spent-on-mobile-apps-by-ranking/ Last downloaded: 28/9/2017
TechCrunch	App Fatigue https://techcrunch.com/2016/02/03/app-fatigue/ Last downloaded: 28/9/2017