# Stand der Technik

Ziel dieses Projekts war es eine App für mobile Endgeräte zu entwickeln, welche dem Nutzer einen möglichst weiten Umfang an anwernderfreundlichen Funktionen bietet, ohne dass hierfür alle Bausteine, von Grund auf, ausprogrammiert werden müssen. Um die App einer möglichst großen Gruppe an Nutzern zugänglich zu machen, wird das API-Level 21 verwendet, was der Android-Version 5.0 entspricht. So haben 80% der Android Nutzer Zugang zu der App.

So werden, um dem Nutzer Zeitraumwähler und *Floating-Action-Button*s mit hoher Usability anzubieten, bereitgestellte Lösungen von anderen Android-Entwicklern implementiert.

Des Weiteren, wird die aktuelle Version der *Support*-Bibliothek benutzt, welche zum Zeitpunkt des Verfassens dieses Dokuments die Version 26.1.0 ist. Das Projekt implementiert die Bausteine *support, appcompat, design* und *constraint-layout*. Durch Verwendung dieser Bibliotheken können Funktionen neuerer Android-Versionen auch auf älteren Geräten verwendet werden.

Um Unit-Tests für das Projekt zu schreiben, wird die in Java üblicherweise verwendete Bibliothek JUnit in der Version 4.12 verwendet.

Da, wie in diesem Dokument schon beschrieben, der Nutzer die Möglichkeit haben soll, sich mit seinem Google-Account identifizieren zu können und Zugang zu einer Echtzeitdatenbank haben soll, werden verschiedene Projekte von Google implementiert. Dies betrifft *die play-services plus* und *auth*, sowie *firebase auth* und *database*. Die genannten Projekte werden wieder in der aktuellen Version in das Projekt implementiert, welche zum Zeitpunkt des Verfassens dieses Dokuments die Version 11.8.0 ist.

# Implementierung

## Autoauswahl

Ein elementarer Bestandteil der Anwendung ist die Autoauswahl. Ein Nutzer soll dort das Modell seines Fahrzeugs wiederfinden können, um seine Log-Einträge eindeutig diesem zuordnen zu können.

Um dem Nutzer eine Listenansicht dieser Fahrzeuge anbieten zu können, welche sich unter verschiedenen Bedingungen ändern kann wurden unterschiedliche Vorgehensweisen getestet.

Es wurde eine Textdatei im JSON Format im res/raw Ordner der Applikation bereitgestellt, welche etwa 27.000 verschiedene Autoeinträge enthält. Aus dieser kann mit der GSON-Bibliothek eine Liste von Java-Objekten erzeugt werden, welche wiederrum durch die Implementierung eines ArrayAdapters diese auf einer ListView darstellen kann. Um die Einträge dem Nutzer möglichst Übersichtlich darstellen zu können, wurde hierfür ein neues Layout im XML-Format erzeugt, welches die einzelnen Views und Viewgroups beschreibt, in denen die Informationen der Fahrzeuge dargestellt werden.

Nach zunächst erfolgreichem Testen wurde, trotz vollständiger und korrekter Darstellung der Liste, diese Vorgehensweise wieder verworfen, da mehrere Probleme auftraten. Beim erstmaligen erzeugen der Liste erfordert dies einen recht hohen Rechenaufwand, wodurch die App für etwa zwei Sekunden nicht antwortet. Dies geschah auch bei Ausführung der Aufgabe in einem zweiten Thread, in einem Handler oder in einem AsyncTask. Weiterhin sollte die Menge der angezeigten Fahrzeuge von dem Nutzer über eine Suchfunktion zu filtern sein. Um dies mit einem ArrayAdapter zu ermöglichen, muss bei jedem Tastendruck des Nutzers eine neue Liste erzeugt werden, ausschließlich mit den anzuzeigenden Fahrzeugen. Da hierfür jeder Eintrag der Liste auf den vom Nutzer eingegebenen Filter untersucht werden muss, stellte sich dies als zusätzlich inperformant heraus.

Um diesen Problemen entgegenzutreten wird in der App nun auf ArrayAdapter und GSON einschließlich der json-Datei verzichtet. Stattdessen befindet sich nun in dem res/raw eine sqlite Datenbank, welche bei Start der App in den für die Anwendung bereitgestellten databases Ordner kopiert wird. Dieser befindet sich in /data und ist von einem normalen Nutzer ohne root-Rechte nicht einsehbar und durch ständiges Überschreiben bei Appstart wird gewährleistet, dass auch bei Änderung der bereitgestellten Datenbank bei Aktualisierung der App durch den PlayStore die Datenbank auf dem Gerät auf dem aktuellen Stand ist.

Es muss kein weiterer SQLite Client implementiert werden, da dieser schon in den Standardbibliotheken von Android enthalten ist, im package *android.database.sqlite.* So kann mit einer einfachen Erweiterung der Klasse SQLiteOpenHelper der Zugriff auf die Datenbank ermöglicht werden. Bei Ausführen einer Anfrage auf dem vom Helper erzeugten SQLiteDatabase Objekt liefert dieses uns ein Cursorobjekt zurück, welches das Ergebnis dieser Anfrage repräsentiert. Die wichtigsten Anfragen die wir für unsere Liste benötigen, ist die Abfrage nach allen Daten (*SELECT \* FROM cars*), sowie zusätzlich die Möglichkeit nach einem Feld zu filtern (*WHERE field LIKE* Statement).

Wie bereits erwähnt wird kein ArrayAdapter verwendet, welcher auf die Verwendung einer Liste von Objekten im Speicher besteht, sondern der Cursor wird mit einem Cursoradapter an die ListView gebunden. Hierfür wurde ein eine Klasse geschrieben, welche den Cursoradapter erweitert und beschreibt wie jedes Ergebnis der Datenbankabfrage, welches sich im Cursor befindet, auf einem View dargestellt wird.

Hierfür wird wieder das vorher erwähnte Layout von dem Cursoradapter verwendet, um einen View zu erzeugen und den Inhalt der Spalten der Datenbanktabelle den TextViews zuzuordnen und zu befüllen.

Durch das Wegfallen der aufwändigen Listen können so nun ohne große Verzögerungen auf Anfragen vom Nutzer reagiert werden. So wird in der QueryTextChange()-Methode der implementierten Suche, welche aufgerufen wird wenn der Nutzer den Text in der Sucheingabe ändert, eine neue Anfrage an die Datenbank gestellt, der Cursor des Adapters ausgetauscht und die ListView mit dem Adapter aktualisiert.

## LoginActivity

Um dem Nutzer eine einfache Möglichkeit zu bieten seine Daten über mehrere Geräte zu synchronisieren, kann dieser sich durch sein Google Konto authentifizieren. Hierfür wurde eine weitere Activity erzeugt, welche in der Manifestdatei mit dem theme *AppCompat.Light.Dialog* versehen wurde. So erhält eine vollwertige Activity die Vorteile eines Dialogs, wie etwa die Möglichkeit die Activity abzubrechen durch tippen auf den Hintergrund.

Die Activity baut das Layout mit einem Knopf auf, welcher mit der Funktion versehen ist, einen mit den Google Bibliotheken erzeugten SignIn-Intent abzuschicken. Wenn der Nutzer einen Account gewählt oder den Vorgang abgebrochen hat, wird die Funktion onActivityResult mit dem Ergebnis aufgerufen. Diese wiederum informiert die aufrufende Aktivität und beendet sich selber. In der Zukunft ist es sehr einfach in diese Aktivität weitere Authentifizierungsmöglichkeiten einzufügen.

## MainActivity

Die Aktivität welche bei Start der Anwendung gestartet wird, deckt viele wichtige Aufgabenbereich ab. Die Navigationsleiste auf der linken Seite wird erzeugt und ein Listener implementiert, welcher bei Auswahl eines Eintrags aufgerufen wird. Dieser unterscheidet zwischen den Menüs und dem Login/Logout Eintrag. Wenn der Nutzer das Menü wechseln möchte, wird der Inhalt der Hauptaktivität durch ein passendes Fragment ersetzt, entsprechend der Auswahl. Der Login/Logout Button öffnet die bereits beschriebene LoginActivity, sollte der Nutzer nicht eingeloggt sein oder loggt den Nutzer aus. Da auch die Nutzung der App ohne eine Authentifizierung mit Google zugelassen werden sollte, wird der Nutzer nicht nur ausgeloggt, sondern authentifiziert sich mit einer einmaligen User-ID, welche beim Einloggen in der App an den Google-Account gebunden wird. Die anonyme Authentifizierung findet auch beim Start der App statt, sollte kein Nutzer eingeloggt sein, anonym oder über Google.

## Zusammenfassung

Über den Zeitraum des Projekts wurden die Mitglieder mit zahlreichen unterschiedlichen Aufgaben und Problemen konfrontiert, welche sie bewältigen mussten. Hierfür wurden die Aufgabenbereiche auf die Teammitglieder verteilt, jedoch stellte sich heraus, dass alle Mitglieder ein gutes Verständnis über die wichtigsten Teile der Android-SDK benötigten. So mussten unter anderem die für die Oberfläche und Logik zuständigen Personen häufig gemeinsam an einem Problem arbeiten, wenn dies eine gute Kenntnis über verschiedene Komponenten der gegebenen Bibliotheken und der entwickelten Struktur erforderte. Dazu zählen unter anderem die Performanceprobleme, welche sich des Öfteren in unerwarteten Situationen offenbarten. Zusätzlich musste die zu Beginn abgesprochene Struktur der Daten wiederholt überarbeitet werden, sobald festgestellt wurde, dass diese nicht oder nicht optimal in der App implementiert werden konnte.