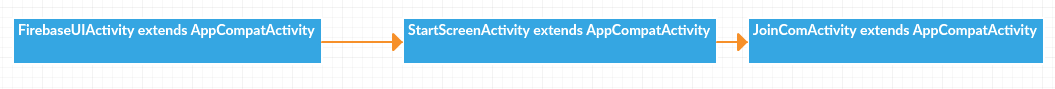
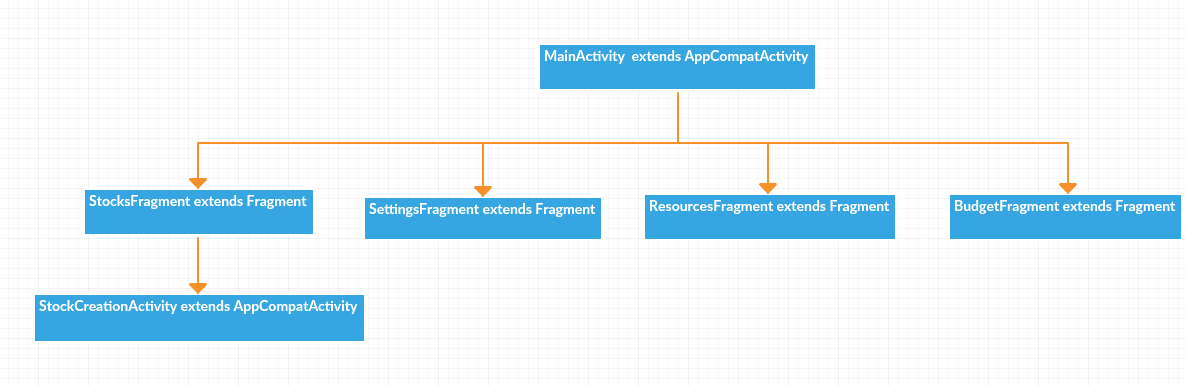
**Entwurf und Implementierung**

Die Applikation soll nach dem Model–view–viewmodel Pattern implementiert werden. Bei diesem Pattern wird das Model und die View voneinander getrennt und kommunizieren nur über das Viewmodel. Das bedeutet konkret das jede Activity ein eigenes Viewmodel und eine eigene View besitzt. Zudem wird für die MainActivity ein Navigator sowie 4 Fragments implementiert, um die Startseite übersichtlicher und leicht zu navigierbar zu gestalten. Die Kommunikation soll vor allem über Events erfolgen, die zu einem späteren Zeitpunkt serialisiert werden. Der Login wurde über Firebase gebildet. Das Backend und die Datenbank sind ebenfalls über Firebase realisiert. Zum Scannen der Barcodes wird eine von Google bereitgestellte BarcodeScanningActivity verwendet. Diese wird zusammen mit der DatabaseActivity in einen seperaten util Ordner eingeordnet.

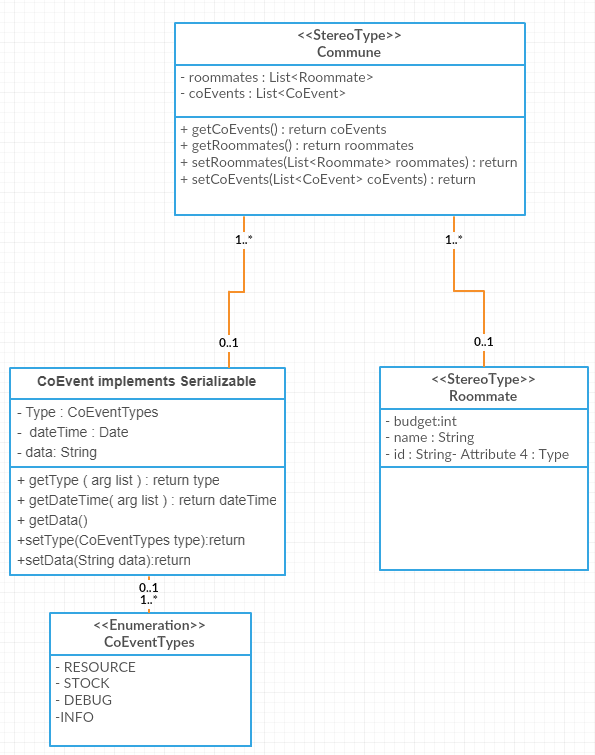
**Activities**

Es folgt der login flow der Activites

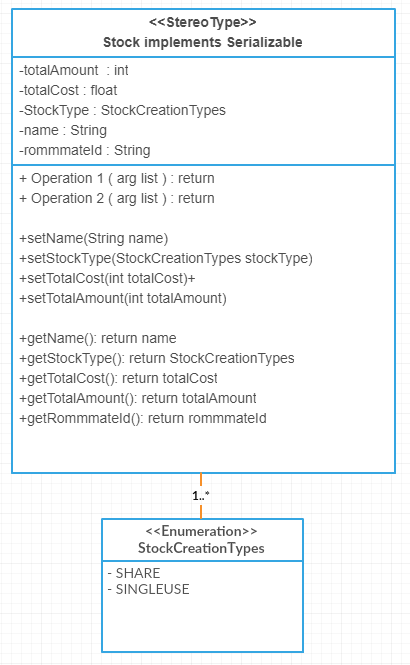
Beim Start der Applikation wird ein Check durchgeführt. Dieser prüft, ob ein User bereits eingeloggt ist. Ist dies nicht der Falle, wird per Intent die firebaseUIActivity aufgerufen. Diese von Google bereitgestellte Activity erzeugt einen typischen Signin Screen, der über ein onClick Event die Anmeldeanfrage an das Backend schickt. Die Anmeldung und deren Identity provider muss zuvor in der Firebasekonsole registriert werden. Wenn dies geschehen ist, wird der User, auf die startscreenactivity geleitet. Diese checkt, ob er bereits Teil einer Wohngemeinschaft ist, und wird in diesem Fall zur MainActivity geleitet. Falls dem nicht so ist, kann er, wie in der funktionalen Anforderung beschrieben, per onClick Event eine neue Wohngemeinschaft erstellen oder einer bestehenden beitreten. Beim erstellen einer Wohngemeinschaft wird ein Event an das Backend geschickt und in der Datenbank eine neue Wohngemeinschaft mit einem User erstellt.

Ist der User registriert und Teil einer Wohngemeinschaft, befindet er sich in der Mainacitvity, welche sich in 4 Fragments teilt. Alle diese Fragments zeigen eine Listview. Das Budgetfragment braucht kein Model, um ein Listobjekt anzuzeigen, stattdessen liest es die Eventliste des Commune Objekt und generiert daraus die einzelnen Kosten. Wird eine der Kosten als beglichen markiert, wird dies ebenfalls als Event geloggt. Das Settingsfragment zeigt statische Listenelemente, die es dem User ermöglichen sich auszuloggen oder den Eventlog zu betrachten. Das Stocktfragment lest ebenfalls den Eventlog der Wohngemeinschaft und generiert aus allen Events mit dem Stocktyp eine Liste.

Zudem kann der floating Button mit einem onclick event auf die stockcreationactivity leiten. Diese kann ein Stockevent erstellen und dem Log hinzufügen. Außerdem hat die stockcreationactivity Zugriff auf die BarcodeScanningActivity. Damit werden 1. Barcodes gelesen und zu den Events hinzugefügt und 2. Können Barcodes gescannt und mit Eventlog verglichen werden, gibt es einen Treffer werden die Daten aus dem gefundenen Event geladen.

**Models**

Um den Anforderungen gerecht zu werden, werden die Objekte als einfache JavaObjekte umgesetzt. Sie dienen der Serialisierung der vom Backend erhaltenen Daten. Dabei existieren im Backend nur Communes und User. Die Communes enthalten eine Liste von Events. Diese sind mit einem Zeitstempel und einem Typ versehen. Dadurch kann die Applikation auch offline operieren und bei späterer Verbindung zu den Servern ihre Ergebnisse synchronisieren. Die Events selber tragen als Json gespeicherte Daten. Im Moment werden die Daten noch durch andere Models serialisiert, dies wird aber im späteren Verlauf der Entwicklung noch dynamischer gestaltet.

Für Testzwecke ist in dieser Version auch das Stockmodel implementiert, um die Daten eines Events zu serialisieren. Daher wird es auch mit dem Interface Serializable erweitert. Es enthält Daten über Anzahl der gekauften Waren, Preis, Usagetyp welcher als Enum implementiert ist (sollte später noch geändert werden wegen Ineffizienz von Enum in android), Stockname und Id des Creators. Dieses Model wird in der Stockactivity genutzt, um je nach Usagetyp Beträge zu errechnen.