Android

# Architektur

Architektur: MVVM

* Model
* View
* ViewModel

# UI

-

## Design

Material Design

# Manifest

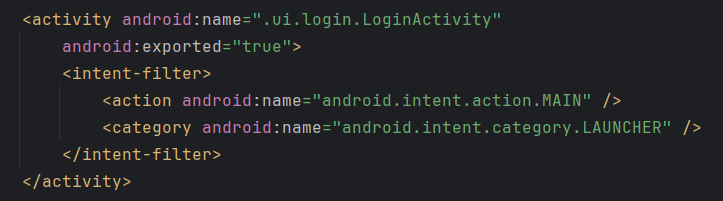
-

## Permission

-

## Activity on Startup

Standardmäßig wird beim Starten der App die MainActivity ausgeführt, die dann wiederum die UI befüllt. Da unsere App jedoch einen Account benötigt, indem die Ergebnisse gespeichert werden, soll die Login-Seite jene sein, die zuerst geladen werden soll. Dazu sind folgende Änderungen im Manifest nötig:



Mit dem <intent-filter> wird dem Betriebssystem mitgeteilt, wie es auf eine bestimmte Aktion reagieren soll. In diesem Fall wird unter <action> definiert, dass es sich bei der „LoginActivity“ um die Hauptaktivität handelt, und unter <category>, dass ein Launcher der App erzeugt wird, mit dem man die App starten kann.

# Technologien

Auswahl: Kotlin/Flutter

IDE: Android Studio

Build-Tool : Gradle Kotlin DSL

## Libraries

-

## Dependencys

-

## Gradle

### Probleme

Dadurch, dass wir das Projekt über GitHub über mehrere Geräte synchronisieren und verändern, entstand folgendes Problem:

A screenshot of a computer error

Description automatically generated

Da nun die Pfade nicht mehr zusammenpassen, müssen diese geändert werden, damit die JDKs vom Projekt gefunden und verwendet werden können. Dazu war es nötig folgende Systemvariable bei Windows hinzuzufügen:

# Optimierung

Caching

# Kamera

Auswahl:

* CameraX
* Camera2
* Camera

Verwendet: Camera

Android liefert bereits eine Kamera, zu der der Benutzer vollen Zugriff hat. Da die Kamera-Funktionen, die von unserer App verwendet werden, auf Bilder beschränkt sind, ist keine komplexere Library wie CameraX nötig, da diese zusätzlich Videos und Audios aufnehmen kann. Ein Umstieg auf die komplexere CameraX Library wäre jedoch zu einem späteren Zeitpunkt nützlich, da sie zusätzlich Funktionen zur Bildanalyse bereitstellt, die nützlich sind, sollten wir das KI-Modell direkt am Android Device bereitstellen. So bietet CameraX Funktionen direkt zur Übergabe an ML Kit, womit wir mit TensorFlow Lite oder PyTorch Edge unser eigenes Modell integrieren und einbinden können.1

## Funktionsweise

Bevor die Kamera geöffnet wird, wird überprüft, ob die erforderliche Berechtigung zur Nutzung der Kamera erteilt wurde. Wenn die Berechtigung noch nicht gewährt wurde, wird eine Anfrage zur Erteilung der Kamera-Berechtigung gesendet. Sobald die erforderlichen Rechte erteilt werden, wird die Methode „openCamera()“ aufgerufen, in der ein Intent erstellt wird. „MediaStore.ACTION\_IMAGE\_CAPTURE“ gibt dabei an, dass ein Foto gemacht werden soll. Anschließend wird überprüft, ob eine Kamera-App existiert und anschließend wird die Aktivität mit „startActivityForResult()“ gestartet.

Nachdem der Benutzer das Foto geschossen und akzeptiert hat, wird der Benutzer in die Methode „onActivityResult()“ geleitet, in der das Foto via der Klasse „Storage“ gespeichert wird.

# Storage

## Speichern von Bildern

Alle für die zur Speicherung benötigten Methoden befinden sich in der Klasse „Storage“.