Android

# Architektur

Architektur: MVVM

* Model
* View
* ViewModel

# Verzeichnisse

Innerhalb des Projekts arbeiten viele verschieden Dateien zusammen. Dabei gibt es verschieden Verzeichnisse für verschiedene Funktionen.

Die wichtigsten Verzeichnisse sind folgende:

„Android\app\src\main\java\com\example\dermaai\_android\_140\ui“

Dieses beeinhaltet weitere Unterverzeichnisse, die die Klassen (Fragment und ViewModel) zur zugehörigen XML-Datei beeinhaltet.

„Android\app\src\main\java\com\example\dermaai\_android\_140\myClasses“

Hier befinden sich die Helper-Klassen, wie „Storage“, „AppModule“ oder „RequestCallback“.

„Android\app\src\main\java\com\example\dermaai\_android\_140\repo“

Enthält die Interfaces für die Klassen in folgendem Verzeichnis:

“Android\app\src\main\java\com\example\dermaai\_android\_140\repoImpl”

Enthält den Code, der letztendlich den Aufruf zur API tätigt.

„Android\app\src\main\res“

Hier befinden sich weitere Unterverzeichnisse, die ausschließlich XML-Dateien beinhalten, und sich erneut jeweils von ihrer Funktion unterscheiden. Die wichtigsten Dateien befinden sich dabei im Verzeichnis:

„Android\app\src\main\res\layout”

Grundlegende Layout-Dateien, wie die Homepage oder Login-Page, sind hier hinterlegt.

# UI

## Design

Für das Design der App wurde Material Design 3 Dark als Designsystem gewählt, da es eine moderne und benutzerfreundliche Ästhetik bietet, die sich in den Icons und der vordefinierten Farben äußert.

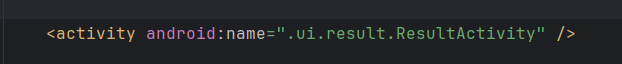
Um ein ansprechendes Design für die App zu gestalten, wurden simple Vektorgraphiken mit der Website haikei.app erstellt. Beim Hinzufügen eines neuen Vektor-Assets in Android Studio erfolgt automatisch eine Konvertierung in eine XML-Datei, was die Integration in die App erheblich vereinfacht, da sich diese an alle Bildschirmgrößen anpasst.

# Manifest

Das Manifest mit dem Namen „AndroidManifest.xml“ spielt eine wichtige Rolle bei der Konfiguration des Projekts und wird in jeder Android-Applikation implementiert.

## Probleme

Wenn der Benutzer die Activity wechselt, um beispielsweise die Seite zu wechseln, stürzt die App ab. Um diesen Fehler zu korrigieren, muss die Activity im Vorfeld im Manifest folgendermaßen registriert werden:



## Permissions

Aufgrund der Funktionalität, die die Applikation bietet, benötigt diese verschiedenen Berechtigungen, um ordnungsgemäß arbeiten zu können.



Innerhalb dieses Blocks wird definiert, dass die Applikation die Funktionalität bereitstellt, Fotos schießen zu können. Diese Zeile Code berechtigt die App nicht automatisch zur Verwendung der Kamera, diese muss weiterhin explizit vom Benutzer genehmigt werden. Damit dies überhaupt erst möglich ist, muss es jedoch im Manifest wie im Code-Abschnitt angegeben werden.

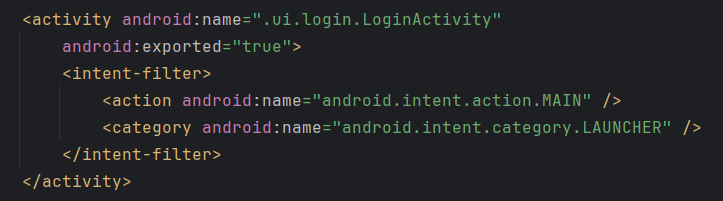
A close up of a text

Description automatically generated

# Activitys

## 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.

# Fragments

## Probleme

Nach dem Erstellen der Fragments entstand das Problem, dass diese zu klein ausgefallen sind. So wurden anfangs einzelne Fragmente in der Login-Page für die Buttons und den User-Input erstellt. Dies hatte den Vorteil, dass die einzelnen Elemente wiederverwendet werden können, allerdings auch den Nachteil, dass es nicht möglich war Daten zwischen den einzelnen Fragmenten auszutauschen. Daher bestand die Lösung darin, nur zwei Fragmente für die Login-Page und Register-Page zu erstellen, da hier keine Daten zwischen den Klassen fließen.

# Technologien

## Sprachen / IDE / Architektur

Programmiersprache: Kotlin

IDE: Android Studio

Architektur: MVVM

Build-Tool: Gradle Kotlin DSL

## Design

Designsystem: Material Design 3 Dark

Hintergrundbilder: erstellt mit der Platform haikei.app

## Libraries / Dependencys

-

## Gradle (SDKs/JDKs)

### 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:

Zusätzlich musste die Systemvariable ANDROID\_HOME gesetzt werden, welche üblicherweise automatisch bei der Installation von Android Studio gesetzt werden sollte:



## Build-Time

# Optimierung

Caching, Asynchron, -

# Kamera

Auswahl:

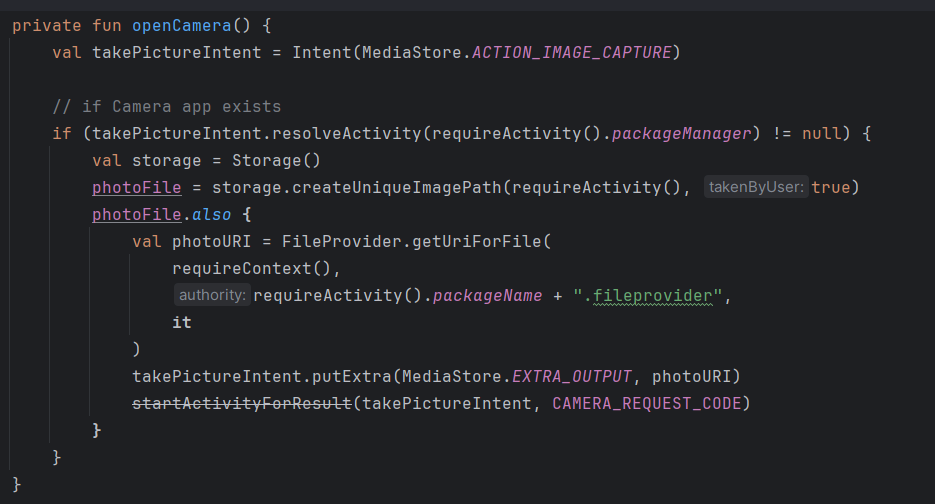
* CameraX
* Camera2
* Camera

Verwendet: Camera

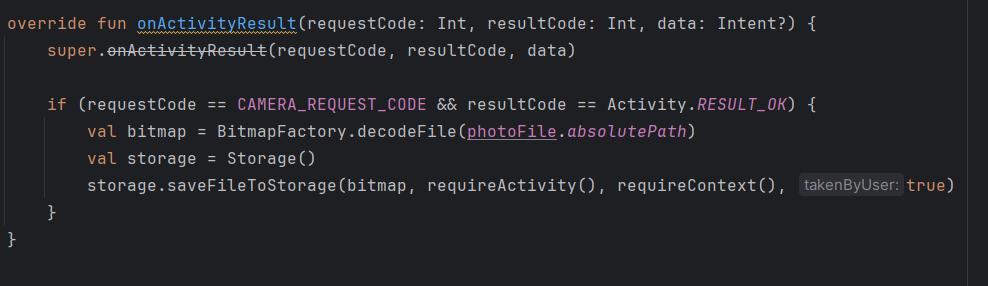
Android liefert bereits eine Kamera, zu der der Benutzer vollen Zugriff hat. Da die Kamera-Funktionen, die von unserer App verwendet werden, auf Bilder beschränkt sind, ist keine komplexere Library wie CameraX nötig, da diese zusätzlich Videos und Audios aufnehmen kann. Ein Umstieg auf die komplexere 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.

## 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. 

## Probleme

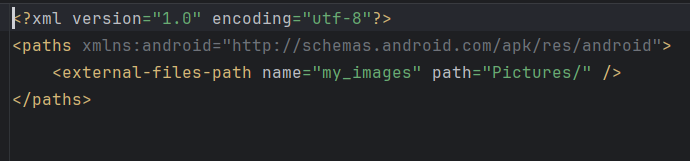
Beim Schießen der Fotos fiel auf, dass die Auflösung (256\*192) nicht hoch genug war, um potenzielle Hautläsionen zu erkennen oder angemessen zu bewerten. Grund dafür war, dass „MediaStore.*ACTION\_IMAGE\_CAPTURE*“ nur das Thumbnail zurückgibt.



Dieses Problem konnte mit der Implementierung eines „File Providers“ gelöst werden, der erneut im Manifest registriert werden muss. Die Bezeichnung des Providers wird mit dem „android:authorities“-Tag spezifiziert.



Zudem war die Implementierung der Datei „file\_paths.xml“ nötig, welche dem File Provider das Verzeichnis preisgibt, auf das er zugreifen darf.



Durch die Implementierung des File Providers wird das hochauflösende Bild (4096\*3072) direkt im externen Speicher abgelegt, anstatt auf das Thumbnail zurückzugreifen.

# Storage

## Speichern von Bildern

Alle für die zur Speicherung benötigten Methoden befinden sich in der Klasse „Storage“. Diese Klasse ist dafür zuständig, Bilder, die mit der Kamera aufgenommen wurden zu verwalten und zu speichern.

Der Speicherort für die Bilder, die der Benutzer schießt, ist folgendermaßen definiert: “/storage/emulated/0/Android/data/com.example.dermaai\_android\_140/files/Pictures/Photo\_User/<filename>.jpg”