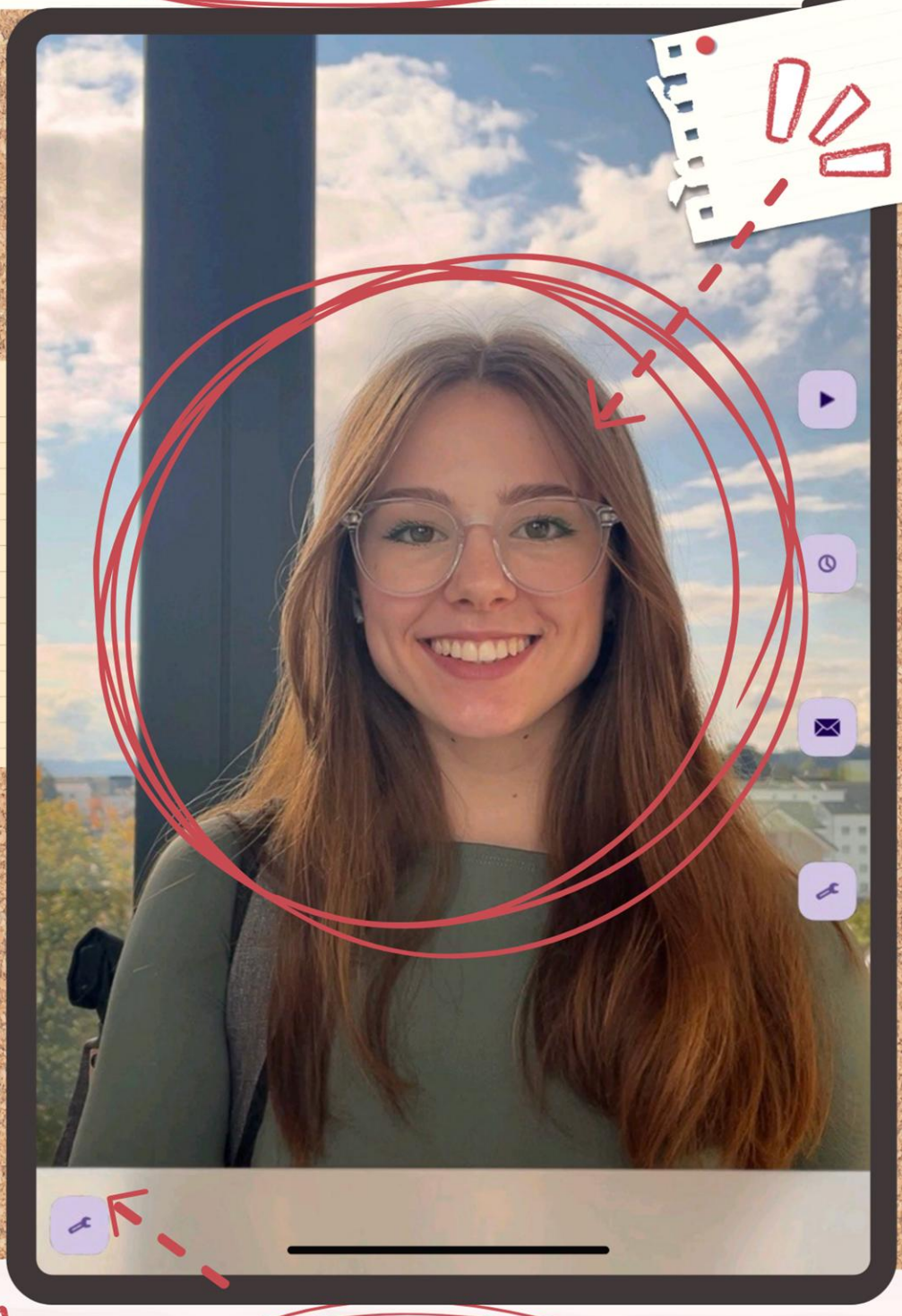
Integrales Informatik Praxisprojekt

PERSÖNLICHER SPIEGEL



HSLU Hochschule Luzern

Alenka Isenring, Nico Graf, Maria Micioni, Andrea M. Sustic

I. Abstract

Dieser Bericht dokumentiert den Planungs- und Realisierungsprozess zur Entwicklung einer interaktiven Applikation. Im Rahmen des Integralen Informatik Praxisprojekts 1 (IIP1) an der Hochschule Luzern wird eine Applikation entwickelt, die mit Hilfe eines Tablets und einer integrierten Webcam eine zuverlässige Gesichtserkennung für bis zu fünf Personen realisiert. Das System zeigt personalisierte Informationen wie Begrüssungen, Alarme oder Kalenderdaten abhängig von der erkannten Person.

II. Inhaltsverzeichnis

1 Aufgabenstellung 1

2 Stand der Technik 2

2.1 Entwicklungsumgebung 2

2.1.1 Android Studio 2

2.1.2 Kivy 2

2.2 Gesichtserkennung 3

2.2.1 OpenCV 3

2.2.2 ML Kit Face Detection API 3

2.3 XCamera 4

2.4 Room Datenbank 5

3 Ideen und Konzepte 6

3.1 Konzept 6

3.2 Hardware 7

3.3 Software 7

3.4 Use Cases 8

4 Methoden 9

4.1 Projektplan 9

4.2 Arbeitsjournal 10

5 Realisierung 11

5.1 Aufbau und Struktur der App 11

5.2 Ordnerstruktur 12

5.3 Berechtigungen 14

5.4 Kameraeinbindung 14

5.5 Datenbank 16

5.6 Funktionen der Applikation 17

5.6.1 Persönliche Songauswahl 17

5.6.2 Alarm 19

5.6.3 Kalender 21

5.6.4 Einstellungen 22

5.7 Sicherheitsaspekte 23

5.7.1 Datenverschlüsselung 23

5.7.2 Implementierungsschritte 23

6 Evaluation und Validation 27
6.1 Scope des Projekts 27
6.2 Projektergebnisse und Umsetzung 27
6.3 Projektergebnisse 28
6.4 Arbeitsweise und Workflow 28
6.5 Herausforderungen 28
7 Ausblick 30
8 Abkürzungs-, Abbildungs-, Tabellenverzeichnis 31
8.1 Abkürzungsverzeichnis 31
8.2 Abbildungsverzeichnis 31
8.3 Tabellenverzeichnis 31
9 Literaturverzeichnis 32

1 Aufgabenstellung

Das Ziel dieses Projekts ist die Entwicklung eines persönlichen Spiegels und Infocenters auf Basis eines Tablets mit integrierter Webcam. Das System soll in der Lage sein, mithilfe einer zuverlässigen Gesichtserkennung, bis zu fünf verschiedene Personen eindeutig zu identifizieren. Sobald eine Person erfolgreich erkannt wird, soll das Spiegelbild dieser Person in Echtzeit auf dem Tablet dargestellt werden. Danach sollte eine persönliche Begrüssung mit dem Namen der Person erscheinen, gefolgt von der Möglichkeit, relevante persönliche Informationen wie Kalenderdaten, Alarmfunktionen oder andere personalisierte Inhalte anzuzeigen. Die Benutzeroberfläche sollte intuitiv gestaltet sein, um eine positive Benutzererfahrung zu gewährleisten, mit klaren Symbolen, verständlichen Anweisungen und einem ansprechenden Design, die dazu beitragen, dass die Benutzer sich wohl fühlen und das System gerne benutzen.

Die spezifischen Ziele des Projekts sind:

- **Entwicklung einer robusten und präzisen Gesichtserkennung** für die zuverlässige Identifizierung von bis zu fünf Personen.
- **Echtzeit-Darstellung des Spiegelbilds** der erkannten Person auf dem Tablet, um eine unmittelbare Interaktion zu ermöglichen.
- **Implementierung personalisierter Begrüssungen** und die Anzeige relevanter Informationen, die auf die Bedürfnisse der jeweiligen Person abgestimmt sind.
- **Sicherstellung des Datenschutzes** und der sicheren Verarbeitung persönlicher Daten, um die Privatsphäre der Nutzer zu gewährleisten.
- **Kreative Bestimmung und Umsetzung von Use-Cases**, wie beispielsweise die Integration einer persönlichen Weckfunktion oder die Darstellung von Kalenderinformationen.

2 Stand der Technik

Im Bereich der Softwareentwicklung gibt es bereits viele verschiedene Tools und Umgebungen, die den Entwicklungsprozess erheblich unterstützen können. In diesem Kapitel werden Werkzeuge und Entwicklungsumgebungen kurz vorgestellt und miteinander verglichen, die für das Projekt von Interesse sein könnten.

2.1 Entwicklungsumgebung

Es gibt verschiedene Entwicklungsumgebungen, die die Entwicklung einer Applikation erleichtern können. Zwei bekannte Umgebungen, die für die Entwicklung von Android-Anwendungen genutzt werden, sind Android Studio und Kivy.

2.1.1 Android Studio

Android Studio ist eine weit verbreitete Entwicklungsumgebung für Android-Applikationen. Es bietet Funktionen zur Fehlerbehebung, Tests und eine benutzerfreundliche Oberfläche. Ein Vorteil von Android Studio ist dabei die Möglichkeit Emulatoren zu verwenden, die Tests auf verschiedenen Android-Geräten ohne physischer Hardware ermöglichen (Android Developers, 2024). Da Android Studio speziell auf die Android-Entwicklung ausgelegt ist, können Android-spezifische Funktionen relativ einfach implementiert werden.

2.1.2 Kivy

Kivy ist ein Open-Source-Python Framework zur Entwicklung von plattformübergreifenden Anwendungen und kann auf verschiedenen Betriebssystemen wie Android, iOS, Linux und Windows eingesetzt werden. Dadurch, dass Kivy jedoch nicht speziell auf Android ausgerichtet ist, müssen allenfalls API oder komplexere Codierung verwendet werden um auf Android-spezifische Funktionen wie die Kamera zuzugreifen oder systemeigene APIs zu nutzen (*Kivy*, o. J.).

Kriterium	**Android Studio**	**Kivy**
Plattformunterstützung	Nur Android	Plattformübergreifend (Android, iOS, Linux, Windows)
Programmiersprache	Java, Kotlin	Python
API-Integration	Native Unterstützung für Android-APIs	Zusätzliche Anpassungen erforderlich
Benutzerfreundlichkeit	Hoch	Mittel
Testing / Debugging	Integrierte Emulatoren für verschiedene Geräte	Externe Lösungen erforderlich
Flexibilität	Spezifisch für Android	Plattformübergreifende Anwendungen
Einarbeitungszeit	Mittel bis hoch	Mittel
Kosten	Kostenlos, Open Source	Kostenlos, Open Source
Eignung für Android	Optimal	Eingeschränkt, zusätzlicher Aufwand für APIs

Tabelle 1: Übersicht Android Studio vs. Kivy

2.2 Gesichtserkennung

Für die Implementierung der Gesichtserkennung wurden zwei verschiedene Ansätze in Betracht gezogen: OpenCV und ML Kit Face Detection API. Nach Abwägung der Optionen wurde OpenCV für die Umsetzung gewählt, aufgrund der lokalen Funktionalität und der Existenz einer hilfreichen Community.

2.2.1 OpenCV

OpenCV (Open Source Computer Vision Library) ist eine umfassende Bibliothek für Computer Vision und Bildverarbeitung. Sie enthält zahlreiche Funktionen zur Implementierung von Gesichtserkennung und Bildbearbeitung, die die Entwicklung solcher Anwendungen erheblich vereinfachen. OpenCV ermöglicht es Entwicklern, gängige Algorithmen für die Bildverarbeitung schnell zu implementieren und anzupassen. Die Integration von OpenCV in Android-Anwendungen ist durch die umfassende Dokumentation und die aktive Community gut unterstützt (OpenCV, 2024).

2.2.2 ML Kit Face Detection API

Die ML Kit Face Detection API ist ein speziell für die Gesichtserkennung in Android-Apps entwickeltes Machine Learning Toolkit. Im Vergleich zu OpenCV bietet die ML Kit Face Detection API vortrainierte Modelle, die einfacher in die Anwendung integriert werden können. Diese API nutzt die Vorteile des maschinellen Lernens, um präzise und schnelle Gesichtserkennungen durchzuführen, wobei sie auf Googles Infrastruktur aufbaut und kontinuierlich optimiert wird. Sie ist besonders benutzerfreundlich und bietet eine hohe Erkennungsgenauigkeit. Jedoch ist zu beachten, dass bei der Nutzung von ML Kit eine externe API verwendet wird, die Daten mit Google austauscht. Daher erfordert diese Lösung zusätzliche Sicherheitsvorkehrungen im Hinblick auf Datenschutz und Cybersecurity, da Daten in die Cloud übertragen werden (ML Kit, 2024).

Kriterium	**OpenCV (lokale Verarbeitung)**	**ML Kit Face Detection API (Cloud-Integration)**
Datenschutz	Lokale Verarbeitung, keine externe Datenübertragung	Externe Verarbeitung, Datenübertragung an Google-Server erforderlich
Anpassungsfähigkeit	Sehr flexibel, Algorithmen individuell anpassbar	Begrenzt, vortrainierte Modelle ohne Anpassung
Integration	Gute Dokumentation, aber höherer Entwicklungsaufwand	Sehr einfache Integration, benutzerfreundlich
Erkennungsgenauigkeit	Präzise, abhängig von der Implementierung der Algorithmen	Sehr hoch, optimiert durch maschinelles Lernen
Plattformunterstützung	Plattformübergreifend (Android, iOS, Linux, Windows)	Android und iOS
Offline-Funktionalität	Vollständig offline	Teilweise offline, Cloud-Funktionen oft erforderlich
Dokumentation	Sehr umfassend	vorhanden

Tabelle 2: Übersicht OpenCV vs. ML Kit Face Detection API

2.3 XCamera

XCamera ist eine Open-Source Kamera-Bibliothek für Android, die die Verwendung der Kamera-Funktionen vereinfacht. Sie bietet eine abstrahierte Schnittstelle, die es Entwicklern ermöglicht, Bilder und Videos mit minimalem Aufwand zu erfassen. XCamera ist besonders nützlich, wenn komplexe Kamerafunktionen benötigt werden. Die Bibliothek ist flexibel und lässt sich einfach in Android-Projekte integrieren, ohne dass ein vertieftes Wissen über die technische Steuerung der Hardware erforderlich ist (Android Developers, 2024).

Kriterium	**XCamera**
Einfache Integration	Lässt sich problemlos in Android-Apps integrieren.
Benutzerfreundlichkeit	Einfache API, die die Nutzung der Kamera ohne komplizierte Einstellungen ermöglicht.
Flexibilität	Unterstützt grundlegende und erweiterte Kamerafunktionen (z. B. Fokus, Belichtung).
Wenig Aufwand	Minimiert den Entwicklungsaufwand für die Implementierung von Kamera-Funktionen.
Lokale Verarbeitung	Alle Bild- und Videoaufnahmen bleiben lokal auf dem Gerät, keine Datenübertragung an Server erforderlich.
Optimiert für Android	Speziell für Android entwickelt und gut mit Android-Geräten kompatibel

Tabelle 3: Übersicht XCamera

2.4 Room Datenbank

Die Room-Datenbank ist eine Persistence-Bibliothek von Google, die die Arbeit mit lokalen SQLite-Datenbanken in Android-Apps stark vereinfacht. SQLite ist eine Relationale Datenbank, in der die Daten in Tabellen gespeichert werden und die Abfragesprache ein SQL Dialekt ist. Anstatt selbst komplizierte SQL-Befehle zu schreiben, bietet Room eine Zwischenschicht, mit der mithilfe spezieller Markierungen (Annotations) und sogenannter DAOs (Data Access Objects) auf die Daten zugegriffen werden kann. Die Room-Library gilt als Best Practice, wenn lokale Daten in einer Android-App gespeichert werden sollen.

Zentrale Bausteine von Room:

Entitäten (Entities)
Datenbank-Tabellen werden in Room über Entity-Klassen abgebildet. Jede Entity-Klasse repräsentiert eine Tabelle, und die Klassenattribute repräsentieren die Spalten.

DAO (Data Access Objects)
Ein DAO enthält Methoden, um Datenbankoperationen (INSERT, UPDATE, DELETE, SELECT) auszuführen. DAO-Methoden verwenden Markierungen wie @Insert, @Update, @Delete und @Query für ihre Datenbank-Operationen.

Database-Klasse
Die zentrale Datenbank-Klasse, die mit @Database annotiert ist, enthält Referenzen auf alle DAOs.

Die Room Datenbank bietet folgende Vorteile (Android Developers, 2025):

Kriterium	***Room Datenbank***
SQL-Abfrageprüfung	SQL-Abfragen werden bereits beim Kompilieren auf Fehler geprüft.
Reduzierung von Boilerplate-Code	Durch Annotationen lassen sich wiederholende und fehleranfällige Boilerplate-Codefragmente minimieren.
Effiziente Anpassungen	Room stellt effiziente Wege für Datenbankanpassungen bereit.
Integration	Lässt sich problemlos in Android-Apps integrieren.
Benutzerfreundlichkeit	Bietet eine einfache API zur Interaktion mit der Datenbank.
Optimiert für Android	Speziell für Android entwickelt und nahtlos integriert.

Tabelle 4: Übersicht Room Datenbank

3 Ideen und Konzepte

3.1 Konzept

Die Applikation soll eine intuitive und benutzerfreundliche Oberfläche bieten, wobei die Benutzeroberfläche sich in 3 Hauptbereiche gliedern soll.

Personalisierte Begrüssung: Sobald eine Person durch die Gesichtserkennung identifiziert wird, erscheint ihr Spiegelbild in Echtzeit auf dem Bildschirm, begleitet von einer personalisierten Begrüssung, die für eine bestimmte Zeit angezeigt wird. Neben dem Begrüssungstext soll dabei der Name der erkannten Person erscheinen.

App-Icons und Einstellungen: Auf der rechten Seite des Bildschirms werden App-Icons angezeigt, die einen klaren Überblick und einen schnellen Zugriff auf persönliche Daten ermöglichen. Die klaren Symbole ermöglichen eine einfache und benutzerfreundliche Bedienung der Applikation. Durch das Anwählen des gewünschten Symbols werden die entsprechenden persönlichen Inhalte auf dem Monitor angezeigt, wie beispielsweise der Kalender, Musik oder Wecker. Zusätzlich können mithilfe eines Einstellungs-Icons die Einstellungen zur Personalisierung der Anwendung geändert werden.

Anzeige Persönliche Daten / Settings: Durch das Anwählen des entsprechenden Applikations-Icons können die persönlichen Daten oder die Applikationseinstellungen separat angezeigt und bearbeitet werden.

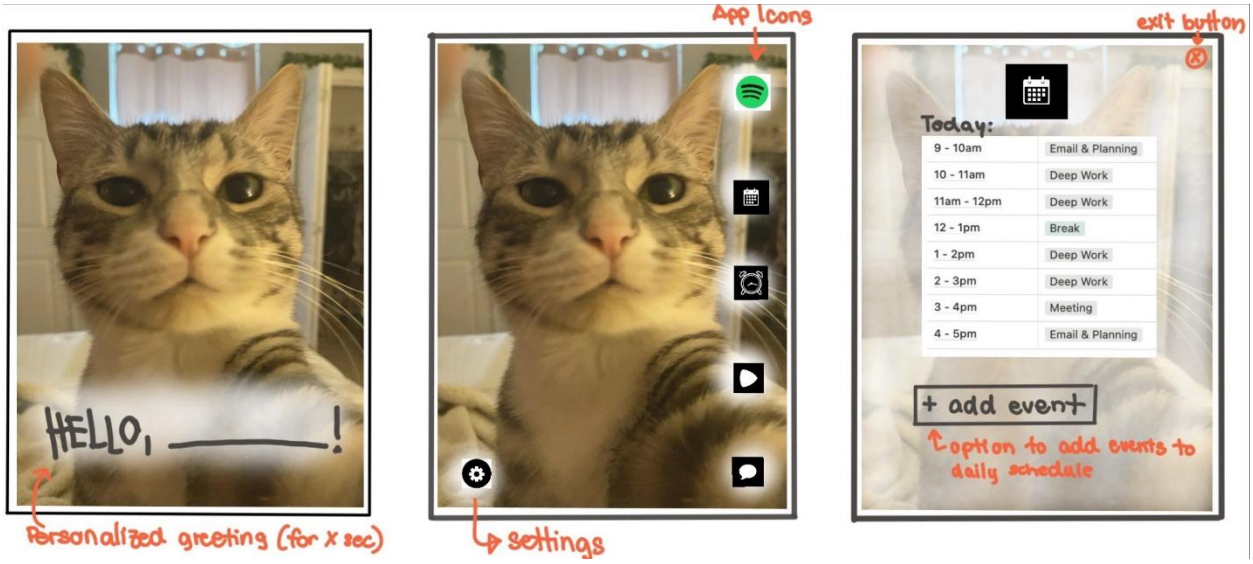


Abbildung 1: Konzept Personal Mirror Begrüssungs-, Haupt- und Kalenderansicht

3.2 Hardware

Als Hardware wurde ein Android Tablet mit integrierter Webcam gewählt. Die Verwendung eines Tablets bietet gegenüber einem Smartphone den Vorteil, dass es über einen grösseren Bildschirm verfügt, der eine übersichtliche Darstellung des Spiegelbildes und der eingeblendeten Informationen ermöglicht. Dies verbessert die Benutzerfreundlichkeit und ermöglicht es die Inhalte klar und deutlich anzuzeigen.

Als Betriebssystem wurde Android gewählt, da einige Mitglieder des Teams bereits Erfahrung mit der Programmiersprache Java haben, die häufig für die Entwicklung von Android-Apps verwendet wird. Im Gegensatz dazu würde die Entwicklung für iOS eine neue Programmiersprache wie Swift erfordern, was eine zusätzliche Herausforderung und Mehraufwand darstellen würde. Ausserdem gilt Android im Vergleich zu iOS als zugänglicher und flexibler in der App-Entwicklung.

Ein weiterer Vorteil von Android ist die Entwicklungsumgebung Android Studio. Dieses Entwicklungsumgebung ist speziell für die Programmierung von Android-Apps optimiert und bietet viele nützliche Funktionen wie integriertes Debugging, Emulatoren und eine benutzerfreundliche Oberfläche.

3.3 Software

Für die Entwicklung der Applikation wurde Android Studio als Programmierungsumgebung ausgewählt. Android Studio ist speziell für die Entwicklung von Android-Apps konzipiert und bietet eine benutzerfreundliche Oberfläche, was die Erstellung und Implementierung von Applikationen erleichtert. Zusätzlich bietet die Umgebung den Vorteil, dass der geschriebene Code sowohl mit echter Hardware wie auch mithilfe eines Emulators (virtuellem Gerät) getestet werden kann. Dadurch kann jedes Teammitglied den geschriebenen Code in der Umgebung laufen lassen und debuggen ohne auf das Tablett warten zu müssen.

Als Programmiersprache fiel die Wahl auf eine Kombination aus Java und Kolin. Diese Wahl ist ideal, da die zwei Programmiersprachen in Android Studio ohne Probleme parallel verwendet und kombiniert werden können. Kotlin wurde gewählt, weil sie für die Entwicklung von Android-Anwendungen optimiert ist und in Android Studio als empfohlene Sprache integriert ist. Java wurde verwendet, um die Funktionen von OpenCV zu integrieren, da viele Anleitungen und Bibliotheken für OpenCV in Java verfügbar sind. Die Syntax von Kotlin ist Java sehr ähnlich, wodurch es keinen zu grossen Umstieg für das Team darstellt.

Die Gesichtserkennung bildet einen zentralen Bestandteil der App und wurde mit OpenCV Version 4.10.0 umgesetzt. Die Wahl fiel auf OpenCV, weil die Daten lokal verarbeitet werden können. Im Gegensatz zu cloud-basierten Lösungen wie der ML Kit Face Detection API, bei denen persönliche Daten an externe Server übertragen werden, bleibt bei OpenCV alles auf dem Gerät. Dadurch profitieren Nutzer von einer schnelleren Reaktionszeit und gleichzeitig bleibt die Kontrolle über den Datenschutz gewahrt.

Zur Unterstützung der Softwareentwicklung stellte die Hochschule Luzern (HSLU) ein GitLab-Repository bereit, das die Zusammenarbeit im Team ermöglicht. Diese Infrastruktur erleichtert die gemeinsame Arbeit am Projekt, da alle Teammitglieder jederzeit Zugriff auf den Code haben und ihn bearbeiten können. Ausserdem können Codeänderungen nachvollzogen und die Versionen der Anwendung synchron gehalten werden.

3.4 Use Cases

Persönliche Wecker Funktion

Als User möchte ich einen persönlichen Wecker erstellen können, damit ich um diese Zeit eine Benachrichtigung mit einer Persönlichen Notiz erhalten kann.

Persönliche Songauswahl Funktion

Als User möchte ich einen persönlichen Song auswählen können, der beim Öffnen der App automatisch abgespielt wird, um eine personalisierte Erfahrung zu schaffen.

Persönliche Kalender Funktion

Als User möchte ich einen persönlichen Termin in einem Kalender erstellen und ansehen können, um einen Überblick über meine Termine zu haben.

4 Methoden

Das Projekt wurde auf eine Art geführt, die unter die agilen Methoden fällt. In einem initialen Schritt wurde der Umfang und die wichtigsten Use-Cases definiert. Danach wurden in iterativen Schritten die Zwischenziele definiert und abgearbeitet.

Für die Umsetzung des Proof-of-Concept wurde das Team in zwei Gruppen aufgeteilt. Eine Gruppe arbeitete an der Backend-Entwicklung, insbesondere der Gesichtserkennung, während die andere sich auf das User-Interface und die Applikation konzentrierte. So konnten beide Teams parallel und effizient Fortschritte erzielen.

Sobald die Grundstruktur der Anwendung stand, wurden in wöchentlichen Teammeetings weitere Entwicklungspunkte definiert und priorisiert. Jedes Teammitglied wählte aus einem gemeinsamen Aufgabenpool spezifische Punkte aus, die bis zur festgelegten Frist umgesetzt werden sollten. Eine Abbildung des Aufgabenpools ist in Teilkapitel 4.2 angefügt.

Um den regelmässigen Austausch unter den Gruppenmitgliedern sicherzustellen, wurde ein wöchentliches Meeting am Mittwoch um 13:00 Uhr festgelegt. In diesen Meetings wurden der Projektfortschritt, offene Fragen, mögliche Probleme und das weitere Vorgehen besprochen. Für die Kommunikation ausserhalb der Teammeetings wurde Microsoft Teams genutzt, das auch gleichzeitig als zentrale Dokumentenablage diente.

Die kollaborative Programmierung und die Verwaltung des Codes erfolgte über GitLab. Änderungen am Code wurden regelmässig gepusht und durch Pull Requests geprüft. Die Architektur und Designs sowie weitere Notizen wurden in OneNote erstellt und mit dem gesamten Team geteilt, um eine gemeinsame und transparente Grundlage für die Entwicklung zu gewährleisten.

4.1 Projektplan

Im ersten Teammeeting wurde ein grober Zeitplan erstellt, der alle relevanten administrativen Termine sowie Meilensteine des Projekts festlegte. Zusätzlich wurde ein weiteres Planungstool genutzt, um die kommenden Projektaufgaben und Zwischenziele zu organisieren.

Event	Datum	Verbleibende Tage	Bemerkung
Kick-Off Meeting IIP	21.09.2024	-24	Samstag, 09:00-10:30 Uhr
Teamsitzung	25.09.2024	-20	Erstes Kick-Off
Teamsitzung	02.10.2024	-13	
Obligatorisches Status-Meeting	09.10.2024	-6	Findet um 13:00 online statt
Teamsitzung	16.10.2024	1	
Teamsitzung	23.10.2024	8	
Teamsitzung	30.10.2024	15	
Teamsitzung	06.11.2024	22	
Teamsitzung	13.11.2024	29	
Präsentation Bachelor Infotag	16.11.2024	32	
Teamsitzung	20.11.2024	36	
Teamsitzung	27.11.2024	43	
Teamsitzung	04.12.2024	50	
Obligatorisches Status-Meeting	09.12.2024	55	Findet zwischen 09.12.-13.12.24 statt
Teammeeting	11.12.2024	57	
Schlusspräsentation	16.12.2024	62	Findet zwischen 16.12.-20.12.24 statt
Abgebe Bericht & Code	20.12.2024	66	23h55 in Ilias via

Abbildung 2: Terminplan

4.2 Arbeitsjournal

Im Rahmen des Projekts wurden alle Arbeiten und Tätigkeiten im zur Verfügung gestellten Arbeitsjournal festgehalten Zusätzlich wurde, wie bereits in Kapitel 4 erwähnt, ein Planungstool genutzt, um die Aufgaben in Zwischenziele zu gliedern und effizient im Team zu verteilen. Die definierten Arbeitsstücke wurden dabei wie folgt verwaltet:

Abbildung 3: Pool der definierten und terminierten Arbeitsschritte

5 Realisierung

In diesem Kapitel wird auf die Umsetzung des Persönlichen Spiegels eingegangen. In einem ersten Teil steht der Aufbau der Applikation und das User-Interface der Software im Zentrum. Anschliessend wird die Gesichtserkennung thematisiert und am Ende die Sicherheitsfaktoren.

Im Rahmen des Projekts wurde eine App vollständig lokal entwickelt.

5.1 Aufbau und Struktur der App

Die App wurde von Grund auf neu mit Android Studio entwickelt. Dabei wurden verschiedene Activites und Hilfsklassen verwendet, um verschiedenen Funktionen und Ansichten klar voneinander zu trennen.

Jede **Activity** stellt eine Bildschirmansicht in dar, mit dem der Nutzer in der App interagieren kann. In der „Persönlicher Spiegel"-App gibt es mehrere Activities, von denen jede eine spezielle Funktion übernimmt wie beispielsweise:

- **Login-Ansicht**: Der Start-Bildschirm der Angezeigt wird beim Öffnen der App.
- **Haupt-Ansicht:** Nachdem der User eingeloggt wird, kommt er auf diese Ansicht, die die zentralen Funktionen der App zur Verfügung stellt.
- **Alarm-Ansicht**: Steuert die persönliche Alarmfunktion beim Klick auf das Alarm-Icon in der Hauptansicht.

Für jede Activity wird ein passendes Layout eingebunden, um die Benutzeransicht entsprechend darzustellen.

Jede dieser Activities enthält standartmässig Methoden die den Lebenszyklus der Activity steuern und von der App automatisch aufgerufen werden. Zu diesen Methoden gehören:

- ***onCreate()****:* Diese Methode wird aufgerufen, wenn die Activity startet. Hier wird das Layout gesetzt und alle wichtigen Ressourcen geladen.
- ***onStart()****:* Wenn die Activity sichtbar wird, aber noch nicht vollständig im Vordergrund ist, wird diese Methode aufgerufen.
- ***onResume()****:* Aktiviert die Activity, sodass sie Benutzereingaben akzeptieren kann. Diese Methode wird jedesmal aufgerufen, wenn man wieder auf eine Activity zurückkehrt.
- ***onPause()****:* Diese Methode wird aufgerufen, wenn die Activity in den Hintergrund tritt, aber noch nicht ganz gestoppt ist. Zum Beispiel, wenn die Einstellungen geöffnet werden.
- ***onStop()****:* Wird verwendet, wenn die Activity nicht mehr sichtbar ist, z. B. beim Wechsel zu einer anderen Activity.
- ***onDestroy():*** Beim Schliessen der Activity wird diese Methode aufgerufen, um alle Ressourcen freizugeben.

In der Persönlichen Spiegel Applikationen wurden hauptsächlich die Methoden onCreate(), onResume(), onPause() und onDestroy() verwendet.

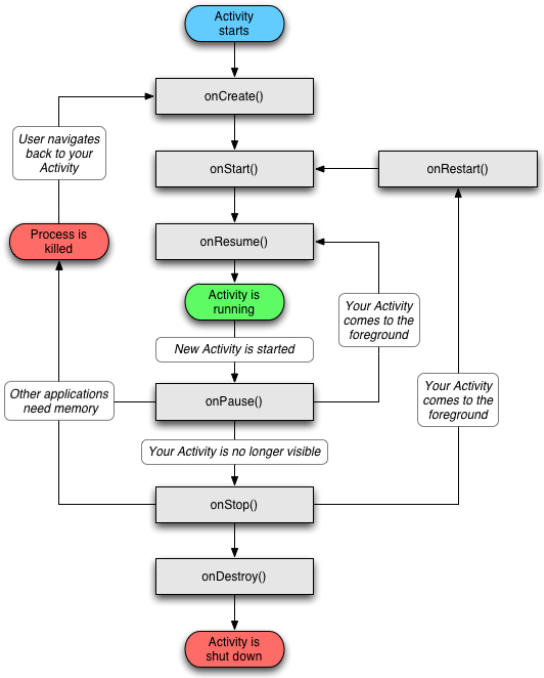


Abbildung 4: Lebenszyklus der Activities in Android Studio

Neben den Activities wurden Hilfsklassen erstellt, die zusätzliche Funktionen bereitstellen und in die Activities miteinbezogen werden können. Diese Klassen kümmern sich nicht direkt um das Benutzer-Interface, sondern bieten spezielle Funktionen. Ein Beispiel ist die CameraManager-Klasse, die für die Kamerafunktionalitäten zuständig ist oder die PermissionHandler-Klasse, die dafür sorgt, dass die App die richtigen Berechtigungen bekommt. Die Funktionen der Hilfsklassen können in den einzelnen Activities jeweils am geeigneten Ort aufgerufen werden, wodurch der Code weniger redundant und übersichtlicher wird.

5.2 Ordnerstruktur

In Android Studio ist eine bestimmte Ordnerstruktur gegeben, die die verschiedenen Komponenten der Applikation voneinander trennt.

Die Ressourcen und Quellcode der Android-App sind unter dem app-Verzeichnis abgelegt. In diesem Verzeichnis sind alle notwenigen Dateien für die Applikation abgelegt. Dabei ist das Verzeichnis wie folgt aufgebaut (Siehe Abbildungen 5 und 6):

Manifest: Diese Datei enthält alle wichtigen Konfigurationen der App. Darin werden die verschiedenen Activities und Berechtigungen, wie Zugriff auf die Gerätekamera definiert. Jede Activity muss in diesem Dokument deklariert werden, damit Android sie korrekt starten kann.

Kotlin + Java, Uimorror: In diesem Ordner befinden sich die Activites und Hilfsklassen, die mit den Layouts verknüpft sind und somit die Funktionen und Benutzerumgebung bereitstellen. Da einige Activites zusammen mit Hilfsklassen für die gleiche Funktion verwendet werden, wurden diese in einem entsprechenden Packet abgelegt. Beispielsweise wurden alle Funktionen die mit der Gesichtserkennung etwas zu tun haben im Packet «biometrie» abgelegt, um eine bessere Übersicht zu erstellen.

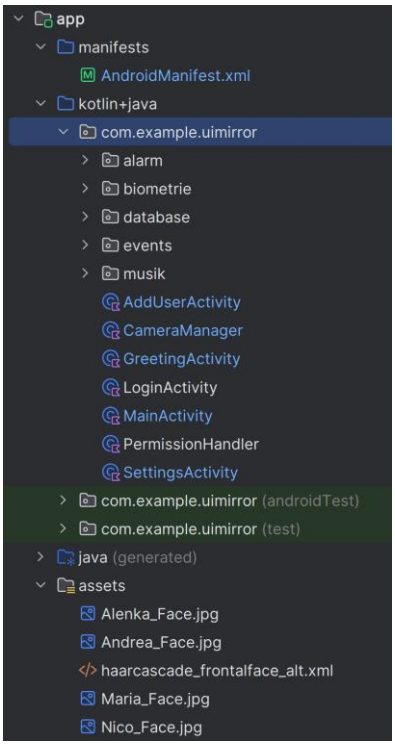


Abbildung 6: Ordnerstruktur der Applikation

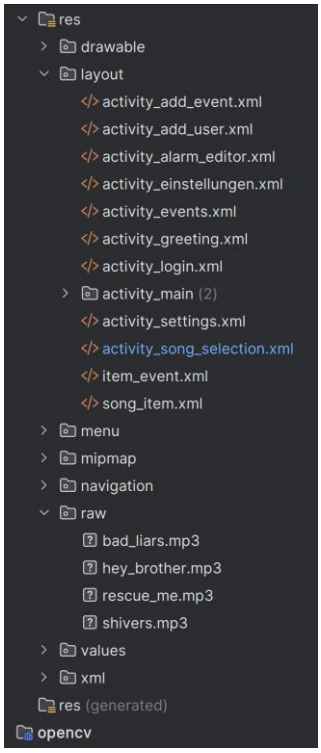


Abbildung 5: Ordnerstruktur der Applikation

Assets: Im Assets Ordner wurden die jeweiligen Portrait-Bilder der User, die für die Gesichtserkennung verwendet werden abgespeichert. Der Vorteil, die Bilder im Ordner Assets anstelle des Ordners Res zu speichern, ist dass die Dateien ein beliebiges Format haben können und von einem beliebigen Typ sein können. Zudem bleiben die Dateien beim Kompilieren des Codes unverändert, wohingegen die Dateien im Ordner Res automatisch optimiert und kompiliert werden.

Das Verzeichnis **«res»** enthält die in der App verwendeten Ressourcen. Diese können direkt von Android verarbeitet und eingebunden werden.

Drawable: In Drawable befinden sich alle grafischen Ressourcen der App, wie Bilder die in den Activites verwendet werden und dem Logo der Applikaiton.

Layout: Im Layout-Ordner befinden sich die XML-Dateien, die die Benutzeroberfläche (UI) der einzelnen Activities definieren. Jede Layout-Datei beschreibt die Struktur und das Design einer spezifischen Ansicht und enthält Elemente wie Buttons, Textfelder, Icons und Bilder.

Um die Spiegelfunktion der App sicherzustellen, wurde in jedem Layout eine Kamera-Preview integriert, die entweder den gesamten Bildschirm oder die obere Hälfte des Bildschirms einnimmt. Dadurch wird der Benutzer in Echtzeit auf dem Bildschirm angezeigt,

Raw: Im Ordner Raw wurden die MP3 Audio-Dateien abgelegt, die für die persönliche SongSelection verwendet werden. Dadurch können die MP3-Dateien direkt von der App und dem Gerät abgespielt werden.

5.3 Berechtigungen

Damit die Applikation auf Funktionen wie die Gerätekamera oder Benachrichtigungen zugreifen kann müssen diese in den App-Einstellungen des Geräts erlaubt werden.

Beim Start der Applikation wird mithilfe der Hilfsklasse PermissionHandler geprüft, ob die entsprechenden Berechtigungen erteilt wurden. Falls dies nicht der Fall ist, wird dem User eine Meldung angezeigt, die ihm erklärt weshalb die Berechtigungen gebraucht werden und ihn direkt zu den Appeinstellungen weiterleitet (Siehe Abbildung 7).

Abbildung 7: Systemdialog welcher Kameraberechtigungen anfordert

Der User muss die Berechtigungen in den Einstellungen manuell ändern, da Android aufgrund Sicherheitsaspekte nicht erlaubt, dass Applikationen die Berechtigungen selbstständig ändern. Ansonsten könnten Applikationen die Berechtigung wie zum Beispiel auf den Standort zuzugreifen aktivieren, ohne dass der User etwas davon mitbekommen würde.

Bei der Rückkehr zur Applikation wird nochmals geprüft, ob die Berechtigungen erteilt wurden, falls nicht erscheint die gleiche Meldung erneut. Falls die Kameraberechtigung für die Applikation gegeben wurde, wird die Kamera gestartet und das Bild auf dem Bildschirm angezeigt.

Die Berechtigungen werden auf jeder Activity erneut geprüft, falls der User während der Verwendung der App, die Berechtigungen ändert.

5.4 Kameraeinbindung

Die Kamera ist eine der wichtigsten Funktionen der App. Sie stellt den eigentlichen «Spiegel» dar und wird für die Gesichtserkennung verwendet, die zum Einloggen und für die Anzeige personalisierter Inhalte verantwortlich ist. Um die Kamera des Tabletts in der App zu integrieren, sodass diese angesteuert und das Bild in der App angezeigt werden kann, wurde die CameraX-Bibliothek von Android verwendet.

Einbindung von CameraX

Die Kamera wird über die Importierte Klasse «**ProcessCameraProvider»** gesteuert. Diese ermöglicht den Zugriff auf die Gerätekamera. Damit die App immer die Frontkamera des Tablets verwendet, wurde diese als Standard festgelegt. Für das Testen der Applikation über einen Emulator muss die Kamera manuell auf die Rückkamera umgeschaltet werden, da die Webcam mit den Einstellungen der Frontkamera nicht funktioniert.

Mit der Preview-Funktion von CameraX konnte eine Live-Kamerafeed eingerichtet werden, der direkt in der App angezeigt wird. Der Live-Kamerafeed erscheint in einem PreviewView, der im Layout der Benutzeroberfläche eingebunden ist. Dadurch wird die Kamera direkt auf dem Bildschirm angezeigt.

Für die Steuerung und Verwaltung der Kamera wurde eine eigene CameraManager-Klasse erstellt. Sie startet die Kamera und sorgt dafür, dass die Bilder analysiert werden. Gleichzeitig wird der Live-Kamerafeed in der Preview initialisiert und auf dem Bildschirm dargestellt.

Die Funktion *bindToLifecycle()* sorgt dafür, dass die Kamera mit dem Lebenszyklus der jeweiligen Activity verbunden ist, um Ressourcen effizient zu verwalten. Dadurch wird sichergestellt, dass die Kamera gestoppt wird, wenn die Activity pausiert oder zerstört wird.

Mithilfe der ImageAnalysis-Funktion von CameraX werden die aufgenommenen Bilder in Echtzeit verarbeitet und analysiert.

Berechtigungen prüfen und anfordern

Die Applikation prüft vor der Verwendung der Kamera ob die nötigen Kameraberechtigungen vorhanden sind. Falls die Berechtigungen fehlen, fordert die App den User über ein Dialogfeld auf, diese zu erteilen. Sobald die Berechtigungen erteilt wurden wird die Kamera gestartet und der Live-Kamerafeed angezeigt. Falls der Nutzer die Berechtigung verweigert, wird eine Fehlermeldung ausgegeben und die App kann die Kamerafunktion nicht nutzen.

Verarbeitung der Kameradaten

Die Kamera nimmt laufend Bilder auf und bindet den Feed via Preview an den Bildschirm. Gleichzeitig wird das aktuelle Frame mithilfe von OpenCV in ein passendes Format umgewandelt und analysiert.

OpenCV verwendet einen vortrainierten Haarcascade-Classifier um Gesichter zu erkennen. In einem zweiten Schritt werden die Gesichtsdaten mithilfe eines Torch-Algorithmus aus dem erkannten Gesicht extrahiert.

Gesichtserkennung und Login

Diese Gesichtsdaten werden dann mit den Einträgen in der Datenbank abgeglichen. Für den Vergleich verwendet das Modell einen Ähnlichkeitsalgorithmus, der die Distanz zwischen den Vektoren misst.

Ein Gesicht wird als erkannt eingestuft, wenn die berechnete Distanz unter 0.6 liegt. Dieser Wert wurde aus Sicherheitsgründen gewählt, um eine hohe Genauigkeit und gleichzeitig einen geringe Fehleranzahl zu gewährleisten. Bei einer Differenz von weniger als 0.6 sollte es eine genug hohe Ähnlichkeit zwischen den Gesichtern geben, um das Gesicht zuordnen zu können und die Möglichkeit einer Verwechslung, und somit einem unberechtigten Zugang möglichst klein zu halten. Wenn die Differenz jedoch über 0.6 liegt, könnte es sein, dass ein ähnliches Gesicht ist, allerdings nicht das Gesicht des Users ist.

Sobald ein gefundenes Gesicht mit einem der in der Datenbank gespeicherten Bilder übereinstimmt, wird der User automatisch eingeloggt und in der Datenbank als PrimaryUser gesetzt. Die App zeigt anschliessend einen personalisierten Begrüssungstext mit dem Namen der erkannten Person auf dem Bildschirm an.

5.5 Datenbank

Die Room-Datenbank wurde als persistente Speicherlösung innerhalb der Applikation implementiert, um Benutzerdaten und Alarme effizient zu verwalten. Der Fokus lag dabei auf der Unterstützung personalisierter Funktionen für den Hauptnutzer (Primary User). Die Datenbank enthält verschiedene Tabellen, darunter die zentrale Person-Tabelle, die Informationen wie Benutzer-ID, Alarmzeit und den Status des Primary Users speichert. Jede Zeile der Tabelle repräsentiert einen Benutzer, wobei der Primary User durch das Feld primary_user als aktiv markiert wird. Dies ermöglicht die gezielte Speicherung und Verwaltung von Daten für einen einzigen aktiven Benutzer, während andere Benutzerprofile parallel gespeichert bleiben.

Die wichtigsten Datenfelder in der Person-Tabelle umfassen:

- id: Primärschlüssel zur eindeutigen Identifikation eines Benutzers.
- alarm: Zeitstempel in Millisekunden, der die Alarmzeit definiert.
- primary_user: Boolean-Wert, der angibt, ob ein Benutzer der aktive Nutzer der Applikation ist.

Für den Zugriff auf die Datenbank wurden sogenannte DAOs (Data Access Objects) erstellt. Diese enthalten vordefinierte SQL-Abfragen, um spezifische Daten abzurufen.

Beispiele hierfür sind:

- *getAllPersons:* Gibt alle Benutzer in der Datenbank zurück (SELECT * FROM person).
- *getPrimaryUser:* Ruft den aktiven Nutzer ab, indem nach primary_user = true gefiltert wird.

Das Verhalten der Datenbank kann in Echtzeit über den Database Inspector in Android Studio überprüft werden. In der Inspektionsansicht lassen sich Queries ausführen, wie etwa:

- SELECT * FROM Person WHERE alarm IS NOT NULL; (Ruft Benutzer mit aktivem Alarm ab)
- SELECT * FROM Person WHERE alarm IS NULL; (Zeigt Benutzer ohne aktive Alarme).

5.6 Funktionen der Applikation

Sobald sich der Benutzer mithilfe der Gesichtserkennung erfolgreich in der Applikation angemeldet hat, wird die *MainActivity* geladen. Diese Hauptansicht dient als zentraler Ausgangspunkt, von dem aus die persönlichen Funktionen der App genutzt werden können.

In dieser Ansicht sieht der Benutzer sein eigenes Bild über den gesamten Bildschirm und hat die Möglichkeit, verschiedene personalisierte Funktionen und Informationen anzeigen zu lassen. Um die Benutzerfreundlichkeit zu erhöhen, wurden verschiedene, aussagekräftige Icons in der Hauptansicht übersichtlich platziert. Durch das Anklicken eines bestimmten Icons wird die zugehörige Funktion geöffnet.

5.6.1 Persönliche Songauswahl

Beim Anwählen des Play-Icons in der Hauptansicht wird der User zur *SongSelectionActivity* navigiert. In dieser Ansicht kann der User kann einen persönlichen Song aus einer Liste auswählen. Der gewählte Song wird anschliessend in der Datenbank unter dem User gespeichert und automatisch im Hintergrund abgespielt.

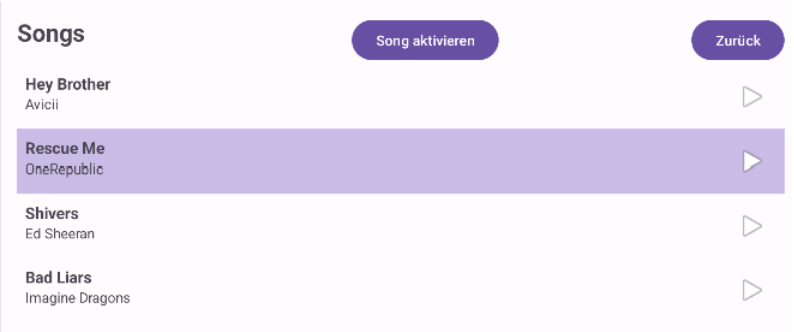


Abbildung 8: Ansicht Songauswahl

Vorschau und Änderung der Songauswahl

Damit der User die richtige Auswahl trifft, kann er mithilfe von Hörproben in jeden Song hineinhören und den gewünschten Song durch Antippen auswählen. Falls der Song geändert werden soll, ist dies jederzeit durch die Auswahl eines anderen Songs möglich.

Songaktivierung und -deaktivierung

Der User hat die Möglichkeit, die Wiedergabe von Songs individuell zu steuern. Mithilfe des «Song aktivieren» respektive «Song deaktivieren»-Buttons kann der User bestimmen, ob die automatische Songwiedergabe aktiv sein soll.

- **Aktivierter Zustand:** Der Song wird beim Start der App oder nach Aktivierung automatisch abgespielt.
- **Deaktivierter Zustand:** Der Song wird pausiert und nicht mehr abgespielt, bis die Funktion manuell wieder aktiviert wird.

Wenn die Wiedergabe eines Songs pausiert wird, speichert die App die aktuelle Abspielposition in der Datenbank. Beim erneuten Start der Wiedergabe setzt der Song an der pausierten Stelle fort.

Wiedergabe und Steuerung der Songs
Die Wiedergabe und Steuerung der Songs basieren auf dem **Android MediaPlayer-Service**. Dieser wurde so konfiguriert, dass die Wiedergabe auch im Hintergrund stabil funktioniert.

Eine Herausforderung bei der Entwicklung war die Synchronisation zwischen der Benutzeroberfläche und dem MediaPlayer-Hintergrunddienst. Dabei war besonders schwer, dass Benutzeraktionen wie das Starten, Pausieren oder Ändern eines Songs fehlerfrei verarbeitet werden, um Unterbrechungen oder Datenverluste zu vermeiden.

Automatisches Handling der Songwiedergabe
Ein Problem trat auf, wenn sowohl eine Hörprobe als auch der Hauptsong gleichzeitig abgespielt wurden, da beide MediaPlayer-Instanzen unabhängig voneinander aktiv waren. Dies führte zu ungewollten Überschneidungen bei der Wiedergabe.

Um dieses Problem zu beheben und sicherzustellen, dass immer nur ein Song abgespielt wird, wurde ein automatisches Handling implementiert. Dabei unterscheidet die App zwischen Hörproben und dem Hauptsong:

- **Hörproben:** Beim Abspielen einer Hörprobe wird der aktuelle MediaPlayer automatisch gestoppt und freigegeben, bevor der neue MediaPlayer für die Vorschau initialisiert wird.
- **Hauptsong:** Sobald ein Hauptsong ausgewählt wird, beendet die App automatisch eventuallige Wiedergabe der Hörprobe und gibt die Ressourcen des MediaPlayers frei, bevor der Hauptsong abgespielt wird.

Dieser Mechanismus verhindert Konflikte zwischen parallelen Wiedergaben und optimiert die Nutzung der Systemressourcen.

5.6.2 Alarm

Beim Anwählen des Uhrzeiger-Icons in der Hauptansicht wird der User zur *AlarmActivity* navigiert.

Die Alarmfunktion der Applikation wurde entwickelt, um den Nutzern die Möglichkeit zu geben, personalisierte Alarme zu erstellen und diese mit spezifischen Erinnerungen zu verknüpfen. Die Implementierung dieser Funktion stellte jedoch einige Herausforderungen dar, insbesondere in Bezug auf die korrekte Verwaltung von Berechtigungen und die zuverlässige Ausführung des Alarms.

Das Alarmsystem basiert auf dem Android AlarmManager, einem Systemdienst, der Alarme einplant und auslöst, selbst wenn die Applikation im Hintergrund oder geschlossen ist. Die Hauptkomponenten der Alarmfunktion sind:

1. **Alarm-Planung**: Die Nutzer können mithilfe einer intuitiven Benutzeroberfläche Zeit und Datum für den Alarm einstellen. Diese Daten werden anschliessend in ein Calendar-Objekt umgewandelt und für die genaue Zeitberechnung genutzt. Der Alarm wird mithilfe der Methode setExactAndAllowWhileIdle des AlarmManagers geplant. Diese stellt sicher, dass der Alarm auch im Energiesparmodus des Geräts zuverlässig ausgelöst wird.

2. **Alarm-Auslösung**: Wenn der geplante Zeitpunkt erreicht ist, wird ein Broadcast-Intent durch den AlarmManager an den AlarmReceiver gesendet. Dieser Receiver ist dafür verantwortlich, eine Benachrichtigung an den Nutzer zu senden, um den Alarm auszulösen.

3. **Speichern der Alarme**: Die Alarmeinstellungen eines Nutzers werden in der verschlüsselten Datenbank der App gespeichert. Dies geschieht, indem der aktuell eingeloggte Nutzer in der Person-Tabelle einen Eintrag mit der gesetzten Alarmzeit erhält. Dadurch bleibt der Alarm auch nach dem Neustart der Applikation erhalten.

Ein zentrales Problem bei der Implementierung war die Verwaltung der Berechtigungen. Ab **Android 12** (API-Level 31) führte Google strengere Anforderungen für das Planen von exakten Alarmen ein. Die App benötigt explizit die Berechtigung, exakte Alarme zu planen, da solche Alarme das Energiemanagement des Systems beeinflussen können.

Zu Beginn der Entwicklung funktionierte die Alarmfunktion nicht wie erwartet: Der Alarm wurde zwar gespeichert, aber zum festgelegten Zeitpunkt nicht ausgelöst. Der Grund dafür war, dass die Berechtigung SCHEDULE_EXACT_ALARM nicht erteilt war. Die Lösung bestand darin, sowohl eine Überprüfung der Berechtigung einzubauen als auch den Nutzer bei Bedarf in die Einstellungen weiterzuleiten, um die Berechtigung manuell zu erteilen.

Umgang mit Berechtigungen:

- Die Methode AlarmManager.canScheduleExactAlarms() wird genutzt, um zu überprüfen, ob die App über die erforderliche Berechtigung verfügt.
- Falls die Berechtigung fehlt, wird der Nutzer mithilfe eines Intents zur Einstellungsseite der App weitergeleitet, wo die Berechtigung erteilt werden kann.

Für die Alarmbenachrichtigungen wurde ein NotificationManager verwendet. Beim Auslösen des Alarms erstellt der AlarmReceiver eine Benachrichtigung, die dem Nutzer angezeigt wird. Diese Benachrichtigung enthält den Titel des Alarms und einen standardmässigen Erinnerungstext („Der Alarm wurde ausgelöst!“).

Das Benachrichtigungssystem wurde so gestaltet, dass es auch unter Android-Versionen ab 8.0 (API-Level 26) mit dem neuen Notification Channel-System kompatibel ist. Der Channel stellt sicher, dass die Benachrichtigungen mit einer bestimmten Wichtigkeit und Sichtbarkeit dargestellt werden.

Ein weiteres Problem bestand in der Synchronisierung der Benutzeroberfläche mit der Kameravorschau, die ebenfalls in der AlarmEditorActivity integriert ist. Dies erforderte die Initialisierung von Kamera- und Berechtigungsfunktionen (PermissionHandler), da die Kamera aktiviert wird, wenn der Nutzer den Alarmeditor aufruft. Falls die Kamera-Berechtigung nicht vorliegt, wird dem Nutzer ein Dialog angezeigt, der ihn über die Notwendigkeit der Berechtigung informiert.

5.6.3 Kalender

Beim Anwählen des Kalender-Icons in der Hauptansicht wird der User zur *EventsActivity* navigiert. Die Kalenderfunktion ermöglicht es dem Benutzer, persönliche Ereignisse und Termine direkt in der Applikation zu erstellen und zu verwalten.

Erstellung von Ereignissen

Über die *Ereigniserstellungsansicht* kann der Benutzer neue Termine hinzufügen (Siehe Abbildung 9). In dieser Ansicht gibt es ein Eingabefeld für den Ereignisnamen sowie eine intuitive Kalenderauswahl, in der das gewünschte Datum und die Uhrzeit des Ereignisses festgelegt werden können. Nach der Eingabe wird der Termin durch Drücken des „Add Event"-Buttons in der Datenbank gespeichert (Siehe Abbildung 10).

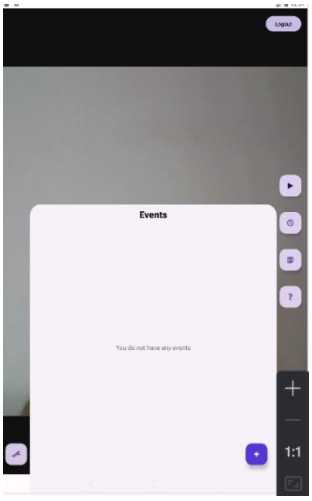


Abbildung 9: Kalender Eventerstellungsansicht

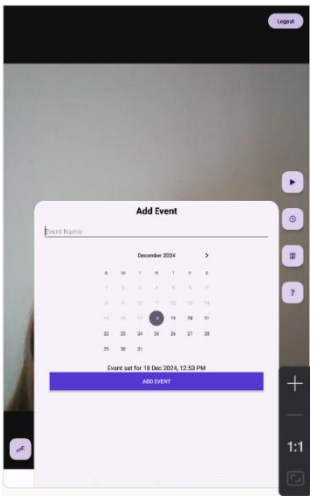


Abbildung 10: Neuer Event im Kalender hinzufügen

Anzeige und Verwaltung von Ereignissen

Die gespeicherten Ereignisse werden in der *Ereignisübersicht* tabellarisch dargestellt. Jeder Termin ist mit dem entsprechenden Namen, Datum und der festgelegten Uhrzeit aufgeführt. Neben jedem Ereignis befindet sich eine Schaltfläche „Delete Event", mit der der Benutzer den Termin bei Bedarf löschen kann. Änderungen werden direkt in der Datenbank reflektiert (Siehe Abbildungen 11 und 12).

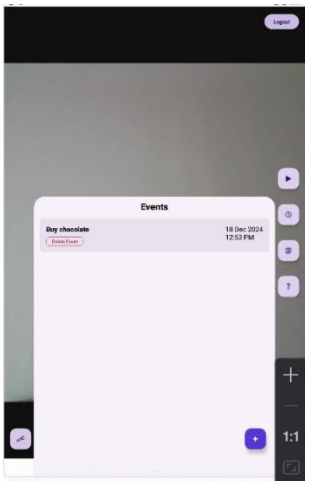


Abbildung 11: Eventansicht im Kalender

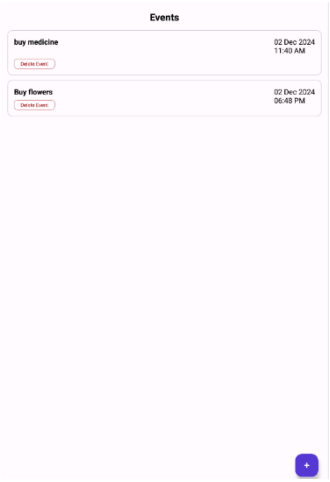


Abbildung 12: mehreren Events im Kalender

5.6.4 Einstellungen

Beim Anwählen des Schraubenschlüssel-Icons in der Hauptansicht wird der User zur *SettingsActivity* navigiert. Die SettingsActivity bietet dem Benutzer verschiedene Funktionen, um seine Einstellungen und persönlichen Daten zu verwalten. Hier kann der Benutzer sowohl seine Kontoinformationen anpassen als auch bestimmte Funktionen der App aktivieren oder deaktivieren.

Benutzerverwaltung

In der Einstellungsansicht gibt es die Möglichkeit, einen neuen Benutzer hinzuzufügen. Durch das Anklicken der Option **„Add User“** kann der Benutzer ein weiteres Profil erstellen. Dadurch können verschiedene Benutzer die App auf dem gleichen Gerät verwenden und weitere später hinzugefügt werden. Jeder Benutzer hat seine eigenen Daten und Einstellungen, die unabhängig voneinander verwaltet werden.

Musiksteuerung

Wie bereits in der *SongSelectionActivity* kann der Benutzer in den Einstellungen die Wiedergabe von Musik aktivieren oder deaktivieren. Anhand eines Toggles sieht der Benutzer direkt den aktuellen Status der Musikwiedergabe und somit, ob die Musik aktiviert oder deaktiviert ist. Mit einem einfachen Klick auf den Toggle-Buttton kann der Benutzer die Musik jederzeit ein- oder ausschalten, ohne dass er die Songauswahl öffnen muss.

Datenverwaltung

Die App bietet den Usern die Möglichkeit ihre persönlichen Daten zu verwalten. Wenn der Benutzer seine Eingaben oder Präferenzen löschen möchte, kann er die Option **„Benutzerdaten löschen“** wählen. Dadurch werden alle persönlichen Einstellungen, wie ausgewählte Songs, Alarme und Kalenderdaten, auf die Standardwerte zurückgesetzt. Ausgenommen davon sind der Benutzername und das Bild, das für die Gesichtserkennung verwendet wird. Diese bleiben erhalten, damit sich der Benutzer weiterhin problemlos einloggen und auf sein Profil zugreifen kann.

Für Benutzer, die nicht nur ihre persönlichen Daten, sondern ihr gesamtes Konto löschen möchten, gibt es die Option **„Konto löschen“**. Diese Funktion entfernt alle vom Benutzer gespeicherten Daten aus der Datenbank. Dies kann sinnvoll sein, wenn der Benutzer die App nicht mehr nutzen möchte oder aus Sicherheitsgründen sämtliche gespeicherte Informationen löschen möchte. Um versehentliche Löschungen zu vermeiden, wird der Benutzer mit einer Sicherheitsabfrage konfrontiert, die ihn fragt, ob er sicher ist, dass er sein Konto löschen möchte. Erst nach einer Bestätigung wird der gesamte Account mit allen zugehörigen Daten unwiderruflich gelöscht.

5.7 Sicherheitsaspekte

Um die Datensicherheit und den Schutz der Privatsphäre der User zu gewährleisten, wurde die «Persönlicher Spiegel»-App so konzipiert, dass sie keine Internetverbindung benötigt. Alle Daten werden lokal auf dem Gerät des Benutzers gespeichert, wodurch das Risiko einer externen Datenübertragung und damit auch potenzieller Sicherheitslücken reduziert wird.

So bleiben alle sensiblen Daten, wie Benutzerinformationen, auf dem Gerät und werden nicht an externe Server gesendet. Das erhöht die Datensicherheit, da der Zugriff auf die Daten nur über das Gerät des Nutzers möglich ist.

Zusätzlich wurde eine Datenschutzrichtlinie und allgemeine Geschäftsbedingungen (AGB) erstellt, die beim ersten Öffnen der App angezeigt werden. In dieser werden Informationen zum Umgang und der Speicherung der persönlichen Daten angezeigt. Der User muss diesen Bestimmungen zustimmen, um die App nutzen zu können (siehe Abbildung 13).

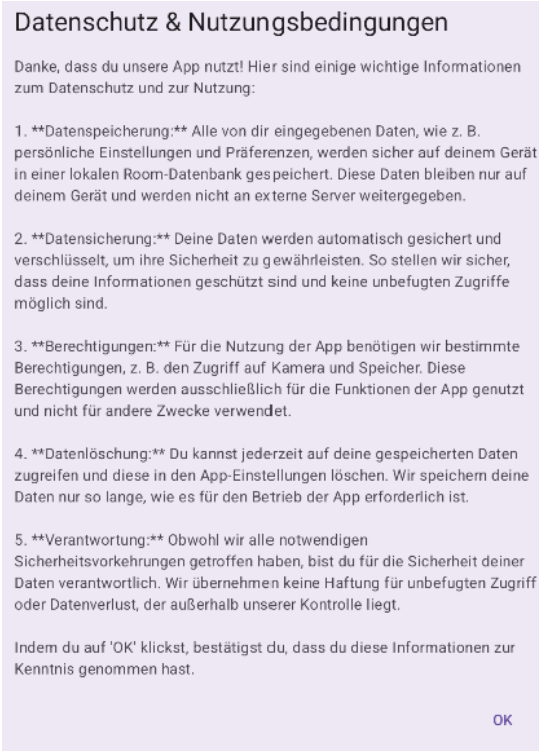


Abbildung 13: Datenschutz & Nutzungsbedingungen der Spiegelapp

5.7.1 Datenverschlüsselung

Ein kritischer Punkt ist der Schutz der Datenbank, die sensible Nutzerinformationen speichert. Ursprünglich wurde die Datenbank unverschlüsselt erstellt, um die Einrichtung und das Debugging zu vereinfachen. Im späteren Entwicklungsprozess wurde jedoch entschieden die Datenbank mit SQLCipher zu verschlüsseln, eine leistungsstarke Open-Source-Erweiterung für SQLite, die umfassende Verschlüsselungsmechanismen bietet. SQLCipher zeichnet sich durch die Verwendung von AES-256-Verschlüsselung aus und bietet Kompatibilität mit der bestehenden SQLite-Schnittstelle, wodurch der Implementierungsaufwand minimiert wurde.

Das Hauptziel der Datenbankverschlüsselung war, die Einsichtnahme in sensible Nutzerdaten durch Entwickler oder Unbefugte zu verhindern. Die Verschlüsselung wurde nachträglich implementiert. Dies erforderte die Migration einer bestehenden unverschlüsselten Datenbank, benannt als „person_database“, in eine neue verschlüsselte Datenbank, genannt „encrypted_person_database“.

5.7.2 Implementierungsschritte

Um SQLCipher zu verwenden, wurden die notwendigen Abhängigkeiten in der build.gradle-Datei hinzugefügt (Abbildung 14). SQLCipher erweitert SQLite um Verschlüsselungsfunktionen, ohne die grundlegende Architektur von SQLite zu verändern. Dies bedeutet, dass bestehende SQLite-Datenbanken mit minimalen Änderungen verschlüsselt werden können. Room ist hierbei die gewählte Datenbanklösung, die eine Abstraktionsebene über SQLite bietet und die Entwicklung und Wartung von Datenbanken erheblich vereinfacht.

```
dependencies {

    //Room dependencies
    implementation("androidx.room:room-runtime:2.6.1")
    implementation("androidx.room:room-ktx:2.6.1")

    // SQLCipher dependency
    implementation("net.zetetic:android-database-sqlcipher:4.5.0@aar")
```

Abbildung 14: Bildschirmfoto build.gradle mit SQLCipher dependencies

Die Sicherheit eines verschlüsselten Systems hängt entscheidend von der Handhabung des Verschlüsselungsschlüssels ab. Ein unsicher gespeicherter Schlüssel könnte die gesamte Verschlüsselung unwirksam machen. Um dieses Problem zu lösen, wurde die Klasse KeystoreManager entwickelt, die den Android **KeyStore** verwendet (Abbildung 15). Der KeyStore ist ein sicherer Bereich im Betriebssystem, der speziell für die Speicherung kryptografischer Schlüssel konzipiert wurde.

```
object KeystoreManager {
    private const val KEYSTORE = "AndroidKeyStore"
    private const val KEY_ALIAS = "EncryptedPersonDBKey" // The unique key alias for your database encryption key

    /**
     * Retrieves the encryption key from the Android Keystore or generates it if it doesn't already exist.
     */
    fun getOrCreateKey(): SecretKey {
        val keyStore = KeyStore.getInstance(KEYSTORE).apply { load(null) }

        // Check if the key alias exists in the Keystore
        if (!keyStore.containsAlias(KEY_ALIAS)) {
            // Generate a new encryption key
            val keyGenerator = KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES, KEYSTORE)
            val keySpec = KeyGenParameterSpec.Builder(
                KEY_ALIAS,
                KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT
            )
                .setBlockModes(KeyProperties.BLOCK_MODE_GCM)
                .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
                .setKeySize(256) // Use a 256-bit key for strong encryption
                .build()

            keyGenerator.init(keySpec)
            keyGenerator.generateKey()
        }

        // Retrieve the encryption key from the Keystore
        return (keyStore.getEntry(KEY_ALIAS, null) as KeyStore.SecretKeyEntry).secretKey
    }
```

Abbildung 15: Bildschirmfoto Klasse KeystoreManager

Ein essenzieller Schritt vor der Migration war die Erstellung eines Backups der bestehenden unverschlüsselten Datenbank. Dieses Backup diente als Fallback-Lösung, um Datenverluste zu vermeiden, falls während der Migration Fehler auftreten sollten. Dies wurde mithilfe der Klasse DatabaseUtils realisiert (Abbildung 16).

```
fun backupDatabase(context: Context): Boolean {
    val databasePath = context.getDatabasePath(DATABASE_NAME)
    val backupPath = File(context.filesDir, BACKUP_NAME)

    return try {
        FileInputStream(databasePath).use { input ->
            FileOutputStream(backupPath).use { output ->
                input.copyTo(output)
            }
        }
        println("Database backup created at: ${backupPath.absolutePath}")
        true
    } catch (e: IOException) {
        e.printStackTrace()
        println("Failed to create database backup: ${e.message}")
        false
    }
}
```

Abbildung 16: Bildschirmfoto Funktion backupDatabase

Die Migration bestand darin, alle Daten aus der unverschlüsselten Datenbank auszulesen und in eine neue verschlüsselte Datenbank zu übertragen. SQLCipher erfordert hierbei die Bereitstellung eines Verschlüsselungsschlüssels, der mit dem Keystore-Manager generiert wurde (Abbildung 17).

```
private fun migrateToEncryptedDatabase() {
    val passphrase = SQLiteDatabase.getBytes(KeystoreManager.getPassphrase())
    val factory = SupportFactory(passphrase)

    CoroutineScope(Dispatchers.IO).launch {
        try {
            // Daten aus der unverschlüsselten Datenbank auslesen
            val allPersons = unencryptedDatabase.uiMirrorDao().getAllPersons()

            // Daten in die verschlüsselte Datenbank einfügen
            encryptedDatabase.uiMirrorDao().insertAll(allPersons)

            // Unverschlüsselte Datenbank löschen
            unencryptedDatabase.close()
            applicationContext.deleteDatabase("person_database")

            Log.i("DatabaseMigration", "Migration erfolgreich abgeschlossen!")
        } catch (e: Exception) {
            Log.e("DatabaseMigration", "Migration fehlgeschlagen: ${e.message}")
        }
    }
}
```

Abbildung 17: Bildschirmfoto Funktion «migrateToEncryptedDatabase»

Die Initialisierung wurde danach so geändert, dass künftig nur noch die verschlüsselte Datenbank verwendet wird.

```
val database by lazy {
    val passphrase = SQLiteDatabase.getBytes(KeystoreManager.getPassphrase())
    val factory = SupportFactory(passphrase)
    Room.databaseBuilder(
        this.applicationContext,
        PersonDatabase::class.java,
        "encrypted_person_database"
    )
        .openHelperFactory(factory)
        .build()
}
```

Abbildung 18: Bildschirmfoto Initialisierung verschlüsselte Datenbank

6 Evaluation und Validation

Zu Beginn des Projekts wurde gemeinsam mit dem Projektbetreuer, Ron Porath, ein Scope definiert, um die Anforderungen und Ziele für die Entwicklung der Applikation klar festzulegen. Dieser Scope diente als Leitfaden während der gesamten Entwicklungsphase und beinhaltete folgende Aspekte:

6.1 Scope des Projekts

Sicherheit:

- **Keine Internetverbindung**: Die Applikation muss lokal funktionieren, um einen hohen Sicherheitsstandard einzuhalten
- **Trennung der Benutzerkonten**: Benutzer A soll keine Einstellungen von Benutzer B einsehen können
- **Einhaltung des Datenschutzes**: Alle relevanten Datenschutzbestimmungen müssen beachtet werden.

OpenCV

- Gesichtserkennung und Analyse muss mit minimaler Verzögerung funktionieren
- ***Nice-To-Have***: Möglichkeit neue User hinzuzufügen

Funktionen

- **Kernfunktion** / minimal Use Case: Alarm-App
- **Ergänzende Funktionen**: persönliche Musikauswahl, Kalender, Notizen

6.2 Projektergebnisse und Umsetzung

Mit Ausnahme der Notizen-Funktion konnten sämtliche Aspekte umgesetzt werden, eine Leistung welche die Erwartungen des Projektteams deutlich übertroffen hat. Diese eher konservative Einschätzung kam zustande, da kein Teammitglied eine Applikation entwickelt hatte oder generell einen Entwickler-Hintergrund hat.

Dass dennoch ein so gutes Resultat zustande kam, liegt vor allem am motivierten Einsatz aller Teammitglieder, die ihre Wissenslücken durch Selbststudium und Experimentieren aufgearbeitet haben. Besonders hilfreich war es, das Projekt in Zwischenziele zu unterteilen, wodurch kleinere, überschaubare Aufgaben fokussiert bearbeitet werden konnten. Somit konnten die Zwischenziele jede Woche jeweils entsprechend auf die Teammitglieder aufgeteilt werden, um einen kontinuierlichen Fortschritt sicherzustellen.

Zu Beginn wurde in Zweierteams gearbeitet, um gegenseitige Unterstützung zu bieten. Später erfolgte die Arbeit überwiegend einzeln, um Konflikte und Überschneidungen zu vermeiden.

6.3 Projektergebnisse

Die Sicherheit der Applikation wurde durch die vollständige lokale Ausführung sowie die Implementierung einer verschlüsselten Datenbank sichergestellt. Auch die Trennung der Benutzerkonten konnte erfolgreich umgesetzt werden, sodass die Privatsphäre und Integrität der Daten gewährleistet sind.

Die verzögerungsfreie Funktion der Kamera und Gesichtserkennung wurde mithilfe der Nutzung effizienter, von Android zur Verfügung gestellten Schnittstellen auf die Hardware implementiert. Zudem wurden, wo es technisch und mit dem verfügbaren Wissensstand möglich war, Ressourcen sparsam eingesetzt. Dazu gehört das Schliessen nicht benötigter Datastreams oder das gemeinsame Nutzen einer Ressource, anstatt mehrere Instanzen zu öffnen.

Auch die Funktionsentwicklung verlief erfolgreich. Die Alarmfunktion wurde als Kernmodul implementiert, und zusätzlich konnten eine Musikwiedergabe und ein Kalender integriert werden. Der Verzicht auf die Notizfunktion war eine bewusste Entscheidung, um sich auf die Optimierung der Kernfunktionen zu konzentrieren, welche jeweils als Minimal Viable Product implementiert wurden.

Durch die Verschlüsselung der Datenbank wurden die Sicherheit und der Schutz sensibler Daten gewährleistet. Allerdings führte dies zu einer leichten Verlangsamung bei Aktualisierungen und Datenzugriffen. Dieses Problem könnte bei einer möglichen Weiterentwicklung der Applikation optimiert werden, beispielsweise durch den Einsatz effizienterer Verschlüsselungsalgorithmen oder durch gezieltes Caching häufig genutzter Daten. Zudem könnten Strategien zur asynchronen Datenverarbeitung implementiert werden, um Verzögerungen weiter zu minimieren und die Benutzererfahrung zu verbessern.

6.4 Arbeitsweise und Workflow

Da der Wissensstand bei jedem Aspekt des Projekts zu Beginn bei null lag, verlief der Workflow in der Regel wie folgt:

Thema recherchieren → Experimentieren → Erstellung eines lokalen Proof-of-Concepts → Funktion im gemeinsamen Repository implementieren → Funktion nach erfolgreichen Tests ins Repository integrieren.

Dieser iterative Ansatz ermöglichte es dem Team, Schritt für Schritt Wissen zu erlangen und gleichzeitig die Applikation kontinuierlich weiterzuentwickeln.

6.5 Herausforderungen

Während der Entwicklung stiess das Projektteam auf verschiedene Herausforderungen. Ein besonders zeitintensiver und anspruchsvoller Aspekt des Projekts war das Debuggen und Testen der Applikation. Hier traten immer wieder Probleme mit dem Emulator auf:

- Bei einigen Teammitgliedern funktionierte der Emulator überhaupt nicht, was die Möglichkeiten zum Testen stark einschränkte.

- Der Emulator benötigte oft sehr lange zum Starten und funktionierte leider nicht immer fehlerfrei. Dies machte es schwer, herauszufinden, ob ein Problem am Code lag oder ob der Emulator selbst nicht richtig funktionierte. Beispielsweise wird die Musik durch den Emulator manchmal nicht abgespielt oder bricht mittendrin ab.
- Da nur ein Teammitglied ein physisches Tablet zur Verfügung hatte, waren die meisten Teammitglieder auf den Emulator angewiesen. Das erschwerte die Tests zusätzlich, da der Emulator häufig mehrmals gestartet werden musste, um sicherzustellen, dass die Fehler reproduzierbar sind und tatsächlich aus dem Code stammen.
- Ein wichtiger Punkt für zukünftige Projekte ist, Sicherheitsfeatures wie die Datenbankverschlüsselung von Anfang an zu berücksichtigen, um den Implementierungsaufwand zu minimieren und potenzielle Risiken frühzeitig zu adressieren.

Diese Probleme führten dazu, dass deutlich mehr Zeit für das Debuggen eingeplant werden musste als ursprünglich gedacht. Trotzdem konnte mit viel Geduld und wiederholten Tests die Funktionalität der App sichergestellt werden.

7 Ausblick

Das Projekt hat insgesamt sehr gute Ergebnisse erzielt, da mehr umgesetzt wurde als ursprünglich geplant – abgesehen von einer optionalen Funktion. Die Erfahrungen aus diesem Projekt können dazu beitragen, den Entwicklungsprozess für zukünftige Android-Applikationen zu verbessern.

Ein wichtiger Faktor für eine schnellere Entwicklung ist die Planung der Applikationsstruktur. Ohne praktische Erfahrung war es anfangs schwierig, diese zu planen. Doch mit den gewonnenen Erkenntnissen können die Struktur von Klassen, Methoden und globalen Instanzen schon vor der Programmierung festgelegt werden, was den Prozess beschleunigt und die Applikation übersichtlicher macht. Auch der Umgang mit Android Studio und Kotlin wird in zukünftigen Projekten schneller und effizienter erfolgen.

Trotz des erfolgreichen Projekts gibt es noch Verbesserungspotential. Die Funktionen könnten weiter getestet und die Benutzererfahrung optimiert werden. Automatisierte Tests könnten helfen, Fehler schneller zu finden. Auch die Gesichtserkennung könnte weiterentwickelt werden, um eine zuverlässigere Anmeldung zu ermöglichen. Dazu wäre eine genauere Untersuchung der Technologie und mehr Zeit für die Umsetzung nötig. Aufgrund der Verschlüsselung der Datenbank kann es zu Verzögerungen bei der Aktualisierung der Einträge kommen. Für zukünftige Arbeiten könnte die Optimierung der Synchronisierung und Datenverarbeitung weiter untersucht und verbessert werden.

8 Abkürzungs-, Abbildungs-, Tabellenverzeichnis

8.1 Abkürzungsverzeichnis

Abkürzung	Bedeutung
DAO	*Data Access Objects*
IDE	integrated development environment
ML Kit Face Detection API	Machine Learning Kit Face Detection Application Programming Interface
OpenCV	Open Source Computer Vision

8.2 Abbildungsverzeichnis

Abbildung 1: Konzept Personal Mirror Begrüssungs-, Haupt- und Kalenderansicht 6
Abbildung 2: Terminplan 9
Abbildung 3: Pool der definierten und terminierten Arbeitsschritte 10
Abbildung 4: Lebenszyklus der Activities in Android Studio 12
Abbildung 6: Ordnerstruktur der Applikation 13
Abbildung 5: Ordnerstruktur der Applikation 13
Abbildung 7: Systemdialog welcher Kameraberechtigungen anfordert 14
Abbildung 8: Ansicht Songauswahl 17
Abbildung 9: Kalender Eventerstellungsansicht 21
Abbildung 10: Neuer Event im Kalender hinzufügen 21
Abbildung 11: Eventansicht im Kalender 21
Abbildung 12: mehreren Events im Kalender 21
Abbildung 13: Datenschutz & Nutzungsbedingungen der Spiegelapp 23
Abbildung 14: Bildschirmfoto build.gradle mit SQLCipher dependencies 24
Abbildung 15: Bildschirmfoto Klasse KeystoreManager 24
Abbildung 16: Bildschirmfoto Funktion backupDatabase 25
Abbildung 17: Bildschirmfoto Funktion «migrateToEncryptedDatabase» 25
Abbildung 18: Bildschirmfoto Initialisierung verschlüsselte Datenbank 26

8.3 Tabellenverzeichnis

Tabelle 1: Übersicht Android Studio vs. Kivy 2
Tabelle 2: Übersicht OpenCV vs. ML Kit Face Detection API 3
Tabelle 3: Übersicht XCamera 4
Tabelle 4: Übersicht Room Datenbank 5

9 Literaturverzeichnis

Android Developers. (2024). *Android Studio und App-Tools herunterladen – Android-Entwickler*. Android Developers. Abgerufen 13. Dezember 2024, von https://developer.android.com/studio/intro?hl=de

Android Developers. (2025). *Daten mit Room in einer lokalen Datenbank speichern*. Android Developers. Abgerufen 04. Januar 2025, von https://developer.android.google.cn/training/data-storage/room?hl=de#kts

Kivy: Cross-platform Python Framework for NUI. (o. J.). Abgerufen 13. Dezember 2024, von https://www.kivy.org/

ML Kit. (2024). *ML Kit*. Google for Developers. Abgerufen 13. Dezember 2024, von https://developers.google.com/ml-kit/guides

OpenCV. (2024). *Home*. OpenCV. Abgerufen 13. Dezember 2024, von https://opencv.org/