Blink Detector

Seminararbeit

vorgelegt von

Jan-Nicolas Weider, Felix Tischler und Mattis Dietrich Dietrich

angefertigt am

Lehrstuhl für Digitale Bildverarbeitung
Fakultät für Mathematik und Informatik
Friedrich-Schiller-Universität Jena

Seminar: Seminar Rechnersehen
Betreuer: Niklas Penzel
Datum: 17. September 2023

Inhaltsverzeichnis

1 Einführung

2 Theorie

2.1 Gesichtserkennung

2.2 Blinzeldetektion

2.3 Features

2.4 Klassifizierung

2.5 App Entwicklung

Unser Projekt hatte zum Ziel, eine Anwendung zu entwickeln, die die Müdigkeitserkennung durchführt. Diese App soll auf verschiedene Plattformen und Geräte zugänglich sein und es den Benutzern ermöglichen, ihre Müdigkeit in Echtzeit zu überwachen. Die App sollte einfach zu bedienen, effizient und zuverlässig sein. Zusätzlich war es von entscheidender Bedeutung, dass die App auch offline reibungslos funktioniert, da sie in Situationen eingesetzt werden soll, in denen möglicherweise keine kontinuierliche Internetverbindung verfügbar ist.

2.5.1 Technologieauswahl

Bei der Auswahl der Technologien für unser Projekt haben wir uns intensiv mit den verschiedenen Optionen auseinandergesetzt. Eine entscheidende Überlegung war die Notwendigkeit der Offline-Funktionalität, da unsere App in Umgebungen zum Einsatz kommen sollte, in denen eine dauerhafte Internetverbindung nicht immer gewährleistet ist. Aus diesem Grund haben wir uns bewusst gegen die Realisierung einer API mit React und Flask entschieden.

Die Wahl der Programmiersprache fiel auf Python, aufgrund seiner Vielseitigkeit und der umfangreichen Unterstützung in der Entwicklergemeinschaft. Python ermöglichte es uns, schnell Prototypen zu erstellen und den Entwicklungsprozess effizient zu gestalten.

Die Entscheidung für das Kivy-Framework als Frontend-Framework war sorgfältig abgewogen. Kivy bietet eine benutzerfreundliche Schnittstelle für die Entwicklung plattformübergreifender Apps, die auf verschiedenen Betriebssystemen und Geräten laufen.

Da unsere App auf einer breiten Palette von Geräten funktionieren sollte, war dies von entscheidender Bedeutung.

Die Begründung für die Wahl dieser Technologien ergibt sich aus ihrer Kombination von Flexibilität und Leistungsfähigkeit. Python ist in der Lage, die erforderliche Logik effizient umzusetzen, während Kivy die Erstellung einer ansprechenden Benutzeroberfläche ermöglicht. Die Möglichkeit, Python-Bibliotheken wie OpenCV und Mediapipe in unser Projekt zu integrieren, stellte sicher, dass wir die Müdigkeitserkennung auf höchstem Niveau durchführen können.

Diese Technologieauswahl ermöglichte es uns, unsere App effektiv zu entwickeln und ihre Ziele zu erreichen. In den folgenden Abschnitten werden wir auf den Entwicklungsprozess, die Implementierung und die Herausforderungen bei der Bereitstellung näher eingehen.

2.5.2 Entwicklungsprozess

Der Entwicklungsprozess unserer App war ein sorgfältig durchdachter Prozess, der auf die Schaffung einer zuverlässigen und benutzerfreundlichen Anwendung abzielte. Dieser Prozess begann mit der Festlegung auf Python als Programmiersprache. Python wurde aufgrund seiner Vielseitigkeit und seiner breiten Akzeptanz in der Entwicklergemeinschaft ausgewählt, was uns eine solide Grundlage für unser Projekt verschaffte.

Als nächstes stand die Frage im Raum, wie wir überhaupt eine App entwickeln können. Wir analysierten verschiedene Ansätze und untersuchten Tools und Frameworks, die unsere Anforderungen erfüllen könnten. Dabei wurden verschiedene Optionen wie Streamlit, Docker und React in Betracht gezogen. Allerdings stellten wir fest, dass diese Tools eher auf die Entwicklung von APIs oder onlinebasierten Anwendungen ausgerichtet waren und nicht unseren Anforderungen an Offline-Funktionalität entsprachen.

Unser Durchbruch kam, als wir das Kivy-Framework entdeckten. Kivy unterstützte Python und erwies sich als ideal für unsere Zwecke. Es ermöglichte uns nicht nur die Integration der benötigten Bibliotheken wie OpenCV und Mediapipe, sondern auch die Entwicklung einer benutzerfreundlichen Benutzeroberfläche.

Die Integration der Python-Funktionalitäten zur Müdigkeitserkennung in die App er-

forderte eine sorgfältige Planung und technische Umsetzung. Wir konzentrierten uns darauf, die Funktionalitäten effizient und zuverlässig in die App zu integrieren, um eine genaue Müdigkeitserkennung sicherzustellen.

Während des Entwicklungsprozesses arbeiteten wir intensiv an der Desktop-basierten Entwicklung der App. Dies ermöglichte es uns, das Layout und die Funktionalitäten der App effizient zu entwickeln und zu optimieren. Dabei standen die Benutzerfreundlichkeit und die Gewährleistung der Offline-Funktionalität stets im Mittelpunkt unserer Bemühungen.

Im Lessons Learned-Teil werden wir näher darauf eingehen, warum wir uns gegen die Fortsetzung der Bereitstellung bis zur Erstellung einer APK-Datei entschieden haben. Dabei werden wir auf die Herausforderungen und Lernprozesse eingehen, die diese Entscheidung beeinflusst haben.

2.5.3 Deployment-Strategie

Die Bereitstellung unserer App auf verschiedenen Plattformen und Geräten war ein wichtiger Teil unseres Projekts. Hier sind einige Aspekte unserer Deployment-Strategie:

- Zielplattformen und -geräte: Wir konzentrierten uns auf die Zielplattformen Android und iOS, da dies die am weitesten verbreiteten mobilen Betriebssysteme sind. Unser Ziel war es, die App auf einer breiten Palette von Geräten nutzbar zu machen.
- Deployment-Tools und -Methoden: Wir verwendeten spezielle Tools und Methoden, um die App erfolgreich zu bereitstellen. Dabei standen die Integration von OpenCV und Mediapipe sowie die plattformübergreifende Kompatibilität im Vordergrund.
- Herausforderungen bei der Bereitstellung: Während des Bereitstellungsprozesses traten einige Herausforderungen auf, insbesondere in Bezug auf die plattformübergreifende Kompatibilität und die Gewährleistung der Offline-Funktionalität. Wir waren jedoch in der Lage, diese Hindernisse erfolgreich zu bewältigen.

Die Erfahrungen, die wir während des Entwicklungsprozesses und der Bereitstellung gesammelt haben, haben unser Verständnis für die Entwicklung plattformübergreifender

Apps vertieft und uns wertvolle Erkenntnisse geliefert, die wir in zukünftigen Projekten anwenden können.

3 Datensatz

4 Evaluation

4.1 Das Interface/Die App

4.1.1 Realisierung

Die App bietet eine Benutzeroberfläche mit verschiedenen Bildschirmen: Dies ist der

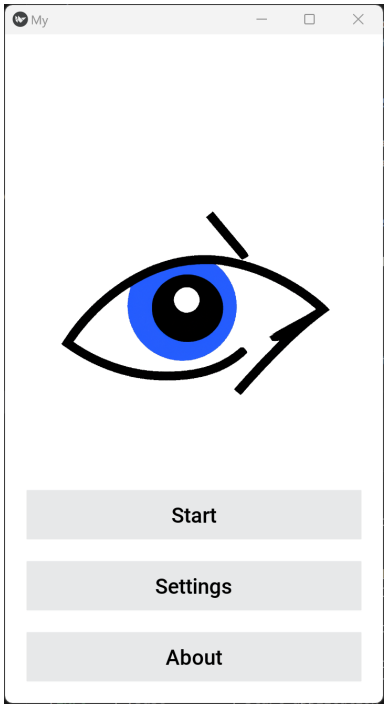

Abbildung 1: Hauptbildschirm (MainScreen)

Einstiegsbildschirm der App. Hier erhalten Benutzer eine kurze Einführung in die App und können die Müdigkeitserkennung starten oder auf die Einstellungen zugreifen. Hier findet die eigentliche Müdigkeitserkennung statt. Die Kamera zeigt eine Echtzeitansicht, und erkannte Gesichtsmerkmale wie die Augen werden markiert. Bei Erkennung von Müdigkeitsmerkmalen kann die App Warnungen anzeigen oder Alarmtöne abspielen. Benutzer können hier verschiedene Einstellungen für die Benutzeröberfläche anpassen.

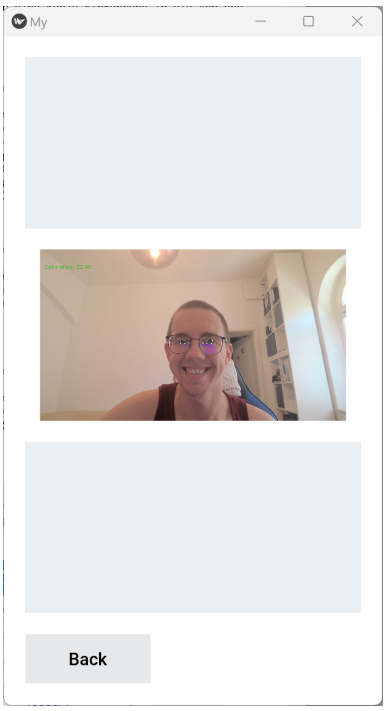

Abbildung 2: Erkennungsbildschirm (DetectionScreen)

Dieser Bildschirm bietet detaillierte Informationen zur Verwendung der App und ihrer Funktionen.

5 Lessons Learned

5.1 Herausforderungen in der App-Entwicklung

Unsere Reise in der App-Entwicklung war von zahlreichen Herausforderungen und Lernprozessen geprägt, die unser Verständnis für die Entwicklung von plattformübergreifenden Anwendungen vertieft haben. Eines der wichtigsten Erkenntnisse, die wir gewonnen haben, war die Bedeutung einer umfassenden Betrachtung aller Faktoren, bevor wir uns auf eine bestimmte Technologie festlegen. In unserem Fall war es ein Fehler, sich frühzeitig auf Python als Programmiersprache festzulegen, ohne die Alternativen zu prüfen. Eine gründliche Analyse aller relevanten Faktoren, einschließlich der Eignung der Programmiersprache, hätte uns möglicherweise zu besseren Entscheidungen geführt.

Wir haben erkannt, dass Java und Kotlin als Programmiersprachen besser zu unseren Anforderungen gepasst hätten. Beide Sprachen bieten nicht nur eine reibungslose

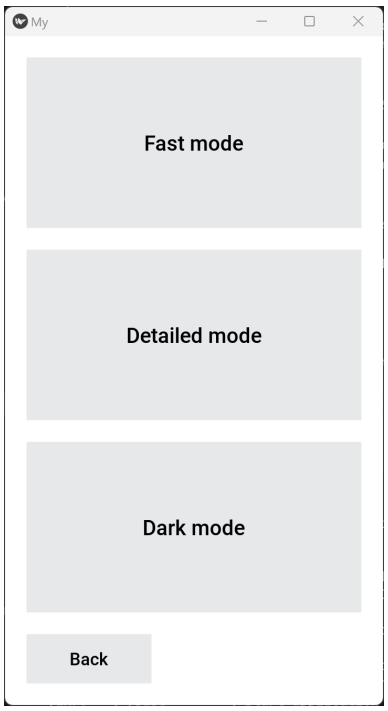

Abbildung 3: Einstellungsbildschirm (SettingsScreen)

Integration der benötigten Bibliotheken wie OpenCV und Mediapipe, sondern sind auch in Bezug auf die App-Entwicklung insgesamt charmanter. Diese Erkenntnis hat uns gezeigt, wie wichtig es ist, die Flexibilität bei der Wahl der Technologie zu wahren und die Optionen offen zu halten.

Ein weiterer wesentlicher Lernpunkt war die Priorisierung von Design und Usability. In unserem Fall lag die Hauptaufmerksamkeit zu stark auf der reinen Integration der Müdigkeitserkennungslogik und dem Versuch, die App zu deployen. Dabei hätten wir mehr Ressourcen und Aufmerksamkeit auf das Design und die Benutzerfreundlichkeit der App und des Interfaces lenken können. Wir haben gelernt, dass eine ansprechende Benutzeroberfläche und eine benutzerfreundliche Interaktion entscheidend für den Erfolg einer App sind.

Darüber hinaus haben wir erkannt, dass die App-Entwicklung noch tiefer gehen kann, insbesondere im Hinblick auf die Nutzung der Hardware des Endgeräts. Wir könnten spezielle Techniken zur Verbesserung der Stabilität und Geschwindigkeit der App erforschen. Dies umfasst die effiziente Nutzung von Ressourcen wie Kamera und Prozessor, um eine reibungslose Erfahrung für die Benutzer sicherzustellen.

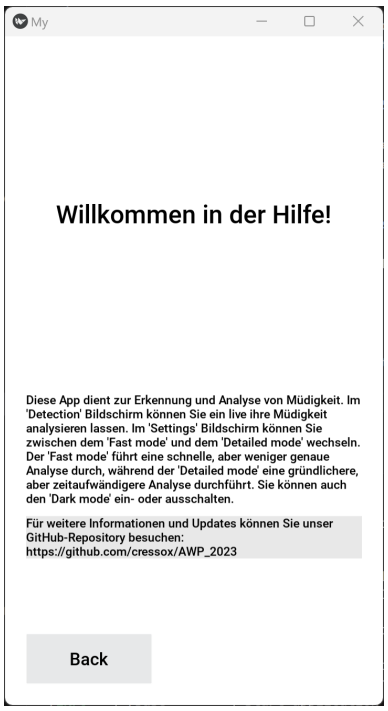

Abbildung 4: Hilfebildschirm (HelpScreen)

Insgesamt haben diese Herausforderungen und Lernprozesse unser Verständnis für die Entwicklung von Apps erweitert und uns dabei geholfen, fundierte Entscheidungen zu treffen. Diese Erkenntnisse werden uns in zukünftigen Projekten dabei helfen, die Qualität unserer Apps weiter zu verbessern und die Anforderungen unserer Benutzer optimal zu erfüllen.

5.2 Deployment-Herausforderungen

Eine der herausforderndsten Phasen unseres Projekts war zweifellos der Versuch, die App als APK-Datei bereitzustellen. Diese Herausforderungen ergaben sich aus unserer anfänglichen Entscheidung, Python und das Kivy-Framework zu verwenden, was sich als eine limitierende Faktoren für das Deployment herausstellte.
Ein bedeutendes Problem war die Unausgereiftheit des Build-Prozesses mit Kivy im Vergleich zu etablierten Alternativen wie Android Studio in Verbindung mit Java oder Kotlin. Der Build-Prozess mit Kivy war anfälliger für Fehler und Schwierigkeiten. Je mehr Abhängigkeiten und Bibliotheken in unserer Logik verwendet wurden, desto geringer war die Wahrscheinlichkeit, dass der Build-Prozess ohne Fehler durchlief.

Wir haben schmerzlich erfahren, dass die Wahl von Kivy für unser Projekt die Möglichkeit, die App als APK-Datei bereitzustellen, stark eingeschränkt hat. Dies führte zu erheblichem Zeitaufwand und frustrierenden Versuchen, Lösungen zu finden. Wir unternahmen eine Vielzahl von Rettungsversuchen, darunter das Deployment über Docker und die Implementierung in einem Container. Wir gingen sogar so weit, die grundlegende Funktionsweise von Kivy beim Kompilieren und Builden tiefgehend zu analysieren.
Die Zeit, die in diese Bemühungen investiert wurde, hätte stattdessen genutzt werden können, um die Potenziale zu erforschen, die bereits in Kapitel 3 erwähnt wurden. Eine verstärkte Fokussierung auf die Verbesserung des Designs, der Usability und der Effizienz der App sowie die tiefere Integration der Hardware des Endgeräts hätte unsere App insgesamt verbessern können.
Diese Deployment-Herausforderungen haben uns wertvolle Einsichten vermittelt, darunter die Notwendigkeit, den Build-Prozess frühzeitig zu evaluieren und eine gut durchdachte Wahl der Entwicklungsumgebung zu treffen. Sie haben uns auch gezeigt, wie wichtig es ist, die Abhängigkeiten und Anforderungen der ausgewählten Technologie sorgfältig zu berücksichtigen, um unerwartete Schwierigkeiten im späteren Entwicklungsprozess zu vermeiden.

6 Zusammenfassung

Literatur

Abbildungsverzeichnis