OTH REGENSBURG

Spiegel AI

Datenverarbeitung in der Technik

Gruppe 5

Leon Kranner leon.kranner@st.oth-regensburg.de
Marco Kuner marco.kuner@st.oth-regensburg.de
David Vollmer david1.vollmer@st.oth-regensburg.de
Marcel Wagner marcel.wagner@st.oth-regensburg.de

15. Juli 2024

Inhaltsverzeichnis

Einleitung

In der Einleitung stellen wir das Projekt **Spiegel AI** vor. Wir beschreiben die Zielsetzung des Projekts, die Motivation und den allgemeinen Aufbau der Dokumentation. Zudem geben wir einen Überblick über die eingesetzte Hardware und Software sowie die geplanten Anwendungsbereiche.

Zielsetzung

Beschreiben Sie hier die Zielsetzung des Projekts.

Motivation

Erläutern Sie die Motivation hinter dem Projekt.

Überblick

Geben Sie einen Überblick über die Struktur der Dokumentation.

1 Hardware

In diesem Kapitel beschreiben wir die Hardware-Komponenten, die für das Projekt **Spiegel AI** verwendet wurden. Wir gehen auf die Auswahlkriterien, die Installation und die Konfiguration der Hardware ein.

1.1 Komponenten

Beschreiben Sie die einzelnen Hardware-Komponenten und deren Spezifikationen.

1.2 Auswahlkriterien

Erläutern Sie die Kriterien, nach denen die Hardware ausgewählt wurde.

1.3 Installation

Beschreiben Sie den Installationsprozess der Hardware.

1.4 Konfiguration

Erläutern Sie die Konfiguration der Hardware-Komponenten.

2 Display

In diesem Kapitel gehen wir auf das Display ein, das im **Spiegel AI** Projekt verwendet wird. Wir beschreiben die Spezifikationen, die Installation und die Anpassungen, die vorgenommen wurden.

2.1 Spezifikationen

Beschreiben Sie die technischen Spezifikationen des Displays.

2.2 Installation

Erläutern Sie den Prozess der Installation des Displays.

2.3 Anpassungen

Beschreiben Sie etwaige Anpassungen oder Modifikationen am Display.

2.3.1 eventuell HTMl seite und Aufbau oder in Installation

2.3.2 Widget 1

2.3.3 Widget 2

2.3.4 Widget 3

2.3.5 Widget 4

2.3.6 Uhr Widget

Erarbeitet von: Marcel Wagner

Die Implementierung des Uhrzeit Widgets für den Smart Mirror ist ein wichtiger Schritt zur Verbesserung der Funktionalität und Benutzerfreundlichkeit des Geräts. Ziel dieses Widgets ist es, die aktuelle Uhrzeit exakt und zuverlässig anzuzeigen. Wobei die Anzeige in Echtzeit aktualisiert werden muss, um stets die genaue Uhrzeit widerzuspiegeln.

Die Implementierung dieses Widgets basierte auf der Nutzung von JavaScript zur

Echtzeitaktualisierung der Uhrzeit und HTML zur Einbettung des Widgets in die Benutzeroberfläche des Smart Mirrors. Desweiteren wurde CSS benutzt um das Widget zu formatieren. Die JavaScript Funktion sorgt dafür, dass die Uhrzeit jede Sekunde aktualisiert wird, während das HTML Dokument die Struktur definiert. Abschließend definiert die CSS Datei das Styling des Widgets.

Während der Entwicklung des Widgets traten mehrere Herausforderungen auf. Eine der größten Herausforderungen bestand darin, sicherzustellen, dass die Uhrzeit in Echtzeit und ohne Verzögerung aktualisiert wird. Dies war besonders wichtig, um die Genauigkeit der angezeigten Zeit zu gewährleisten. Die Verwendung der 'setTimeout' Funktion in JavaScript ermöglicht eine wiederholte Ausführung der Aktualisierungsfunktion in einem festgelegten Intervall von einer Sekunde, wodurch eine kontinuierliche und genaue Aktualisierung der Uhrzeit sichergestellt wurde. Eine weitere Herausforderung war die exakte Zeitanzeige, insbesondere hierbei ist wichtig die Erwähnung der Formatierung der Uhrzeit, um sicherzustellen, dass Stunden, Minuten und Sekunden stets zweistellig angezeigt werden. Durch die Verwendung der 'padStart' Methode konnten die Zahlen auf eine konstante Länge von zwei Stellen gebracht werden, indem bei Bedarf führende Nullen hinzugefügt werden. Dies gewährleistete eine konsistente und gut lesbare Anzeige.

Die Implementierung des Uhrzeit Widgets verlief erfolgreich und erfüllt die gestellten Anforderungen. Die Uhrzeit wird zuverlässig und exakt in Echtzeit angezeigt. Das Widget integriert sich nahtlos in die Benutzeroberfläche des Smart Mirrors und bietet eine klare und gut lesbare Darstellung der aktuellen Uhrzeit. Insgesamt stellt das Uhrzeit Widget eine wesentliche Funktionalität des Smart Mirrors dar. Der nachfolgenden Abbildung 1 kann das Implementierte Uhrzeit Widget auf der HTML Seite entnommen werden.

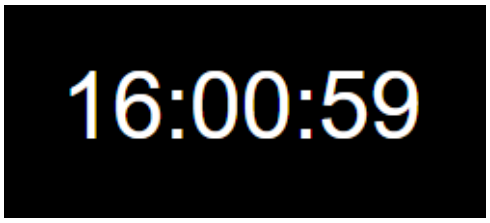

Abbildung 2.1: Uhrzeit Widget
Quelle: eigene Darstellung

2.3.7 Verkehrsinformation

Erarbeitet von: Marcel Wagner

Die Implementierung des Stau-Widgets auf dem Smart Mirror stellt einen wichtigen Schritt dar, um den Nutzern eine umfassende und zuverlässige Quelle für aktuelle Verkehrsinformationen zur Verfügung zu stellen. Das Widget wurde speziell entwickelt, um eine Echtzeitübersicht über die Verkehrslage in Regensburg zu bieten, was insbesondere für Pendler und Reisende von großem Nutzen ist. Durch die Verwendung von JavaScript wurde eine nahtlose Integration mit der OpenStreetMap Overpass API realisiert, die als zuverlässige Datenquelle für Verkehrsdaten dient.

Die Strategie hinter der Implementierung war zweigleisig: Zum einen wurde eine sofortige Aktualisierung der Verkehrsinformationen beim Laden der Seite implementiert, um den Nutzern bei jedem Besuch des Smart Mirrors die aktuellsten Daten bereitzustellen. Zum anderen erfolgt eine regelmäßige automatische Aktualisierung alle fünf Minuten, um sicherzustellen, dass die angezeigten Informationen kontinuierlich aktuell gehalten werden. Dieser Ansatz gewährleistet eine hohe Aktualität und Relevanz der bereitgestellten Verkehrsinformationen.

Während der Entwicklung wurden mehrere Herausforderungen gemeistert, darunter die robuste Fehlerbehandlung, um sicherzustellen, dass Netzwerkprobleme oder API Ausfälle die Funktionalität des Widgets nicht beeinträchtigen. Ein besonderes Augenmerk lag auf der Gewährleistung einer stabilen und zuverlässigen Datenaktualisierung, die für eine nahtlose Benutzererfahrung entscheidend ist.

Das Verkehrs Widget präsentiert die Verkehrslage in einer klaren und intuitiven Benutzeroberfläche. Es informiert die Nutzer klar verständlich darüber, ob derzeit ein Stau vorliegt oder nicht, und bietet gegebenenfalls zusätzliche Informationen über Verkehrshindernisse oder Verkehrswarnungen. Diese klare visuelle Darstellung hilft den Nutzern, schnell zu erfassen, wie die aktuelle Verkehrssituation ihre geplante Route beeinflussen ist.

Insgesamt trägt das Verkehrs Widget erheblich zur Funktionalität und Benutzerfreundlichkeit des Smart Mirrors bei. Es bietet eine unverzichtbare Informationsquelle für die tägliche Routenplanung und unterstützt die Nutzer dabei, ihre Fahrtzeiten effizient zu optimieren. Das Implementierte Verkehrsinformationen Widget kann der nachfolgenden Abbildung entnommen werden. Diese Abbildung zeigt den Fall, dass aktuell gerade kein Stau in den Straßen von Regensburg sind.

Abbildung 2.2: Verkehrsinformations Widget
Quelle: eigene Darstellung

2.3.8 Schlagzeilen

Erarbeitet von: Marcel Wagner

Die Implementierung des Nachrichten Widgets für den Smart Mirror stellt einen wichtigen Schritt dar, um den Nutzern eine aktuelle und relevante Informationsquelle direkt auf seinem Smart Mirror zur Verfügung zu stellen. Das Widget wurde in JavaScript entwickelt und verwendet die 'RSS2JSON-API', um die neuesten Nachrichtenartikel eines ausgewählten RSS Feeds abzurufen und auf dem Smart Mirror anzuzeigen.

Dies ermöglicht eine dynamische und automatische Aktualisierung der Nachrichteninhalte, sobald der Nutzer den Spiegel nutzt.

Ein zentrales Element der Implementierung ist die Verwendung des 'DOMContentLoaded' Events, das sicherstellt, dass das Widget erst aktiv wird, nachdem die gesamte Seite vollständig geladen ist. Dadurch wird sichergestellt, dass alle notwendigen Ressourcen und Elemente bereitstehen, bevor die Datenabfrage und die Darstellung der Nachrichten beginnen.

Die Funktionalität des Widgets umfasst die Asynchronität der Datenabfrage über die Fetch API, die die RSS Feeds von Nachrichtenquellen in ein JSON Format umwandelt, das vom JavaScript Code weiterverarbeitet werden kann. Dies ermöglicht eine schnelle und effiziente Bereitstellung der neuesten Nachrichteninhalte direkt auf dem Smart Mirror, ohne dass der Nutzer zusätzliche Schritte unternehmen muss, um sich auf dem Laufenden zu halten.

Eine besondere Herausforderung während der Implementierung war die unterschiedliche Verfügbarkeit von RSS Feeds bei verschiedenen Nachrichtenseiten. Viele führende Nachrichtenagenturen und Zeitungen bieten zwar RSS Feeds an, einige jedoch nicht oder beschränken den Zugang zu ihren Inhalten über diese Schnittstelle. Dies erforderte eine sorgfältige Auswahl geeigneter RSS Feeds, die eine kontinuierliche und zuverlässige Datenversorgung gewährleisten konnten. Die Ausgegeben Nachrichten dieses Widget entspannen der Frankfurter Allgemeinen Zeitung

Um die Benutzerfreundlichkeit zu maximieren, wurde die Benutzeroberfläche des Widgets bewusst einfach und intuitiv gestaltet. Die angezeigten Nachrichten werden in einer geordneten Liste präsentiert.

Zusammenfassend bietet das Nachrichten Widget einen bedeutenden Mehrwert für den Smart Mirror, indem es den Nutzern eine einfache und effektive Möglichkeit bietet, sich über aktuelle Ereignisse zu informieren. Die Implementierung war erfolgreich in Bezug auf die gesetzten Ziele. Der Nachfolgenden Abbildung kann das implementierte Widget auf dem Smart Mirror entnommen werden.

Abbildung 2.3: News Widget
Quelle: eigene Darstellung

2.3.9 Tankstellen

Erarbeitet von: Marcel Wagner

Die Implementierung des Tankstellen Widgets für den Smart Mirror verfolgt das Ziel,

den Nutzern eine praktische und zeitnahe Information über den günstigsten Kraftstoffpreise einer Tankstelle in der Nähe zu bieten. Diese Funktionalität wurde durch die Integration von JavaScript und die Nutzung der Tankerkoenig API realisiert, die speziell auf die Abfrage von Tankstellenpreisen und Tankstelleninformationen ausgerichtet ist.

Zu Beginn des Implementierungsprozesses wird der 'DOMContentLoaded' Eventlistener verwendet, um sicherzustellen, dass sämtliche Inhalte der Webseite geladen sind, bevor die Datenabfrage gestartet wird. Dies gewährleistet eine stabile und zuverlässige Performance des Widgets auf dem Smart Mirror. Die API Anfrage erfolgt unter Verwendung eines spezifischen API Schlüssels, der die Authentifizierung gegenüber der Tankerkoenig API ermöglicht. Der Standortbezug erfolgt für die Stadt Regensburg mit definierten geografischen Koordinaten und einem Suchradius von 5 Kilometern, um die Tankstellen in unmittelbarer Umgebung zu erfassen.

Die Datenabfrage wird asynchron durchgeführt, um eine reibungslose Interaktion mit der API zu gewährleisten. Nachdem die Daten abgerufen wurden, erfolgt eine Überprüfung auf erfolgreiche Antwort und die Verfügbarkeit von Tankstelleninformationen. Falls die API Daten erfolgreich zurückgegeben werden und Tankstelleninformationen vorhanden sind, wird die günstigste Tankstelle ermittelt. Dies geschieht durch einen Vergleich der Kraftstoffpreise der abgerufenen Tankstellen, wobei die preisgünstigste Option ausgewählt und deren Informationen weiterverarbeitet werden.

Ein zentraler Aspekt der Implementierung ist die robuste Fehlerbehandlung, die sicherstellt, dass der Nutzer bei Problemen wie Netzwerkfehlern oder unerwarteten API Antworten angemessen informiert wird.

Die Darstellung der Tankstelleninformationen auf dem Smart Mirror erfolgt in einer klar strukturierten Form. Dies umfasst den Namen der Tankstelle, die vollständige Adresse inklusive Straße, Hausnummer, Postleitzahl und Ort sowie den aktuellen Preis pro Liter Kraftstoff. Diese Informationen sind leicht zugänglich und ermöglichen es dem Nutzer, schnell die wichtigsten Details zu erfassen und eine informierte Entscheidung zu treffen.

Die Implementierung des Tankstellen Widgets erweitert somit die Funktionalität des Smart Mirrors erheblich, indem sie eine praktische Lösung für die Überwachung und Optimierung der Kraftstoffkosten bietet.

Abbildung 2.4: Tankstellen Widget
Quelle: eigene Darstellung

2.3.10 Test Verfahren

Erarbeitet von: Leon Kranner und Marcel Wagner

Für die implementierten Widgets auf dem Smart Mirror wurden umfangreiche Testverfahren angewendet, die sowohl die Funktionalität als auch die Benutzererfahrung der einzelnen Widgets sicherstellen sollen. Diese unterschiedlichen Testverfahren werden nun im Folgenden genauer beschrieben.

Funktionalitätstests: Dieser Test konzentrierten sich auf die grundlegenden Aufgaben jedes Widgets. Das Uhrzeitwidget wurde auf seine Fähigkeit getestet, die aktuelle Uhrzeit präzise anzuzeigen. Außerdem wurde sichergestellt, dass die Darstellung formatiert und korrekt aktualisiert wird. Beim News Widget lag der Fokus auf der korrekten Abrufung und Darstellung aktueller Nachrichten, wobei sichergestellt wurde, dass die Informationen stets aktuell und relevant sind. Das Tankstellenwidget durchlief API Integrationstests, um sicherzustellen, dass die Kraftstoffpreise korrekt von der Tankerkoenig API abgerufen und in einem klaren Format angezeigt werden. Das Verkehrsinformations Widget wurde auf seine Fähigkeit geprüft, Verkehrsinformationen zeitnah abzurufen und zuverlässig darzustellen, um Nutzer vor aktuellen Verkehrsbehinderungen zu warnen.

Benutzererfahrungstests: Diese waren entscheidend, um sicherzustellen, dass die Widgets intuitiv sind. Hierbei halfen Usability Tests, diese bewerteten die Widgets auf Benutzerfreundlichkeit der Benutzeroberfläche. Dabei wurde besonders darauf geachtet, dass die Widgets übersichtlich gestaltet sind und Nutzer schnell die benötigten Informationen finden können.

Performance- und Lasttests: Diese Testverfahren waren ebenfalls Teil der Teststrategie, um sicherzustellen, dass die Widgets unter verschiedenen Bedingungen effizient arbeiten. Ladezeittests wurden durchgeführt, um sicherzustellen, dass die Widgets schnell genug reagieren und Daten effizient verarbeiten. Skalierbarkeitstests wurden genutzt, um sicherzustellen, dass die Widgets auch bei erhöhtem Datenverkehr stabil bleiben und keine übermäßigen Ressourcen verbrauchen, was besonders wichtig für die Langzeitnutzung ist.

Integrationstest: Dabei wurden Kompatibilitätstests durchgeführt, um sicherzustellen, dass die Widgets reibungslos mit anderen Komponenten des Smart Mirrors interagieren. Systemtests prüften die Gesamtfunktionalität des Smart Mirrors unter verschiedenen Betriebsbedingungen, um sicherzustellen, dass alle Widgets harmonisch zusammenarbeiten und die Gesamtleistung des Systems nicht beeinträchtigen werden.

Diese umfassenden Testverfahren stellen sicher, dass die implementierten Widgets nicht nur funktional sind, sondern auch eine qualitativ hochwertige Benutzererfahrung bieten und unter allen Bedingungen zuverlässig arbeiten.

3 Spiegel AI Remote

Erarbeitet von David Vollmer.

Im folgenden wird die **Spiegel AI Remote** App - auch **Remote App** genannt - beschrieben. Es handelt sich dabei um eine mobile Anwendung, dessen Hauptaufgabe die Fernsteuerung des Smart Mirrors ist.

3.1 Die Flutter™ SDK

Für die Entwicklung einer mobilen Applikation gibt es heutzutage viele Tool-Kits, die verwendet werden können.

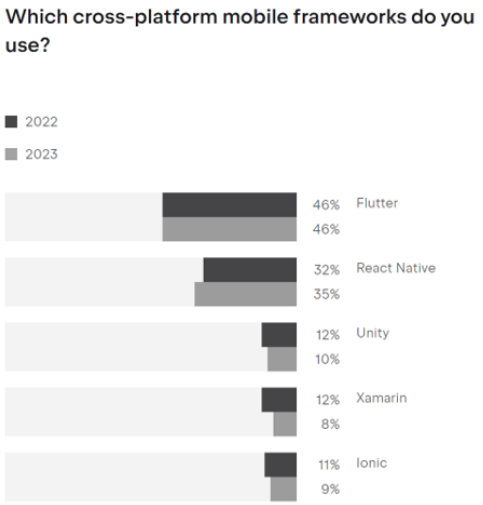

Abbildung 3.1: Laut dieser von JetBrains durchgeführten Umfrage war Flutter im Jahr 2023 das am häufigsten verwendete mobile, plattformübergreifende Framework.[1]

Zusätzlich zu ihrer Popularität ist die von Google entwickelte Flutter SDK in der Lage, mittels AOT-Compiler Programme direkt für die Zielplattform zu kompilieren. Dabei wird die Programmiersprache Dart verwendet.[2] Flutter unterstützt unter anderem die Entwicklung auf den Plattformen Android SDK, iOS, Windows, macOS und Web.[3] Für die Umsetzung der Fernsteuerungs-App wurde insbesondere mit den Plattformen Android und iOS entwickelt und getestet.

3.2 Funktionen

Um das Display des Spiegel AIs fernzusteuern, müssen einige Hauptfunktionalitäten vorhanden sein. Die Remote App muss in der Lage sein, mit dem Spiegel zu kommunizieren, die Anzeige der Widgets auf dem Display zu ändern, verfügbare Widgets auszuwählen und Profile zu verwalten. Mit Ausnahme der ersten Anforderung, welche in der **Implementierung** und im Kapitel **Schnittstelle** näher beschrieben wird, werden all diese Punkte in sogenannten Ansichten (englisch: views) behandelt. Diese kann der Nutzer mithilfe einer Navigationsleiste auswählen.

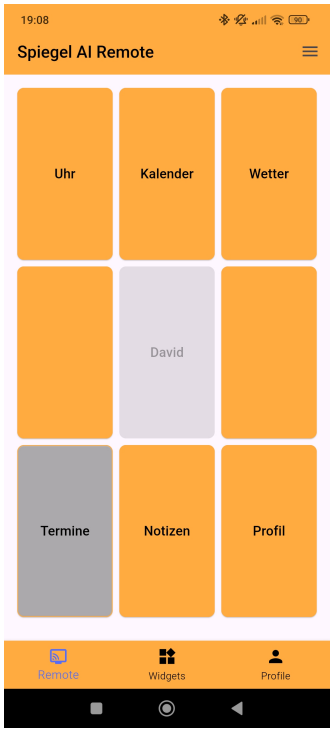

Abbildung 3.2: Remote Ansicht

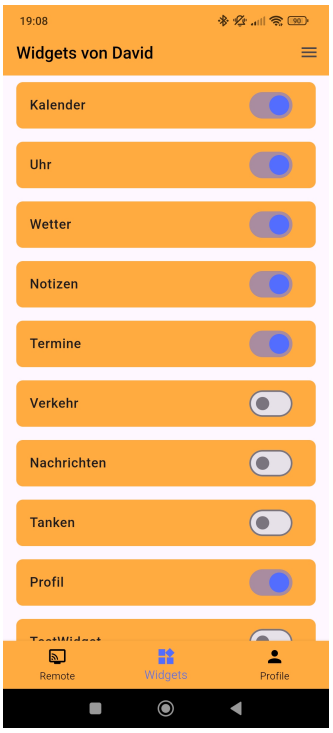

Abbildung 3.3: Widgets Ansicht

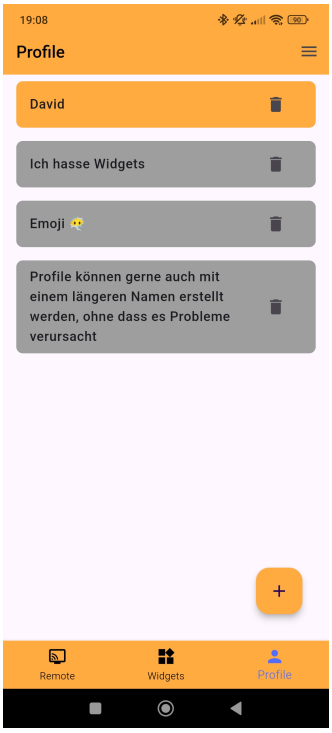

Abbildung 3.4: Profile Ansicht

3.2.1 Remote View

Die Remote View, welche die Standardansicht nach Öffnen der App ist, bietet die Möglichkeit, den Status der Displayanzeige am Spiegel zu ändern. In der Mitte wird der Name des gerade ausgewählten Profils angezeigt. Dieses Feld lässt keinerlei Interaktion zu, da das zentrale Feld der Spiegelanzeige frei bleibt. Das bedeutet, dass bis zu acht Widgets angezeigt und geändert werden können. Mit einem Klick auf einen Button wird das jeweilige Widget aus- oder eingeblendet. Ein Feld in grauer Farbe bedeutet, dass das Widget vom Spiegel AI Display nicht angezeigt wird. Zieht man ein Widget über ein anderes, werden ihre Positionen getauscht. Die Felder, die keinen Text enthalten, haben die selben Interaktionsmöglichkeiten wie die anderen. Sie sind Platzhalter für Widgets, die hinzugefügt werden können. Falls kein Profil ausgewählt ist, wird im Remote View eine Standardeinstellung angezeigt und jegliche Interaktion der Buttons ist ausgestellt. Bei einem Versuch, ohne Profilselektion eine Änderung

vorzunehmen, wird eine Snackbar angezeigt, welche darauf verweist, dass ein Profil geladen sein muss.

3.2.2 Widgets View

In der Widgets Ansicht können für das ausgewählte Profil Widgets ausgewählt werden. Diese View bietet die Widgets Kalender, Uhr, Wetter, Notizen, Termine, Verkehr, Nachrichten, Tanken, Profil und TestWidget an. Bei letzterem handelt es sich um einen Platzhalter, welcher zum Testen der Widgetfunktionalitäten verwendet wurde, aber auch zukünftig mit einem neuen Widget ersetzt werden kann. Die Anwendungen der restlichen Widgets sind im Kapitel **Display** beschrieben. Mithilfe eines Toggle-Buttons werden bis zu acht Widgets selektiert. Beim Versuch, ein neuntes Widget auszuwählen, schlägt dies fehl und eine Snackbar benachrichtigt über die Obergrenze erlaubter Widgets. Auf die Änderung eines Widgets, ohne ein Profil geladen zu haben, folgt ebenfalls eine dementsprechende Fehlermeldung. Wird ansonsten ein Widget ausgeschaltet, dann wird das im Toggle-Button signalisiert und in der Remote View wird der Name des Widgets mit einem leeren Feld ersetzt. Wenn ein ausgeschaltetes Widget ausgewählt wird, aktualisiert sich auch da der Toggle-Schalter und in der Remote Ansicht wird das erste Feld ohne Textinhalt mit dem Namen des Widgets versehen.

3.2.3 Profile View

Die letzte navigierbare Ansicht ist die Profile View. Hier findet die Verwaltung der gespeicherten Profile statt. Die Profile werden aufgelistet und können mit einem Klick ausgewählt werden. Hält man ein Profil für eine kurze Zeit gedrückt, kann man diese in ihrer Position in der Auflistung ändern, indem man sie an die gewünschte Stelle zieht. Löschen kann man einen Eintrag, indem auf das Mülleimer-Icon geklickt wird. Darauf öffnet sich ein sogenanntes Alert-Dialog, welches das Abbrechen oder Bestätigen der Löschung durchführt. Ein neues Profil kann erstellt werden, indem auf ein Button, welches sich in der Ansicht rechts unten befindet und mit einem '+'-Symbol gekennzeichnet ist, gedrückt wird. Es erscheint ebenfalls ein Alert-Dialog, welches mithilfe eines Texteingabefeldes einen Profilnamen geben kann. Dieser Prozess kann auch abgebrochen oder bestätigt werden. Falls bei Bestätigung der Name des Profils leer oder schon vergeben ist, wird unterhalb des Textfeldes eine entsprechende Fehlermeldung ausgegeben. Wenn das Erstellen des Profils erfolgreich ist, wird das neue Profil direkt ausgewählt und bekommt die ersten acht Widgets in der Widgets View zugeordnet. Sie werden dementsprechend in der Remote Ansicht angezeigt. Alle Anpassungen, die in diesen beiden Ansichten getätigt werden, werden in den jeweilig ausgewählten Profilen gespeichert.

3.3 Implementierung

Der Dart-Code, welcher die Codebase für die Kompilierung des Programms darstellt, befindet sich in einem Flutter-Projekt im Verzeichnis mit dem Namen „lib“. Der Websocket wird in der *websocket_manager.dart* verwaltet. Die Funktionalitäten der drei Views sind in den Dateien *remote_content.dart*, *widgets_content.dart* und *profile_content.dart* implementiert.

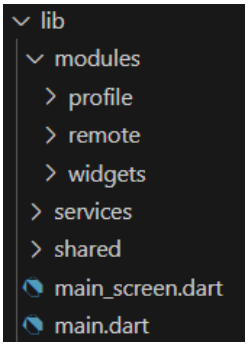

Abbildung 3.5: Verzeichnisstruktur der „lib“ im Spiegel AI Remote Projekt.

3.3.1 Websocket

Um die Kommunikation zwischen der Remote App und Spiegel AI sicherzustellen, muss der Websocket in der App richtig verwaltet werden. Der sogenannte „WebSocketManager“ bewältigt dies mithilfe von Bibliotheken, die die Flutter-Umgebung zur Verfügung stellt. Er ermöglicht, dass eine Verbindung zum Websocket-Server hergestellt und abgebrochen werden kann und erlaubt das Empfangen und Senden von Daten. Die App empfängt Daten als Strings vom Webserver, jedoch werden nur jene mit bestimmten Eigenschaften auch verarbeitet. Der zu empfangende String muss in ein JSON-Format dekodiert werden können. Dann wird geprüft, ob der Wert des ersten Schlüssels mit der Bezeichnung „sender“ den Wert „mirror“ hat. Dies prüft, ob die Nachricht des Servers ursprünglich vom Spiegel AI gesendet wurde. In diesem Fall werden die Werte des Keys mit der Bezeichnung „profiles“ lokal in der App gespeichert. In einem ähnlichen Stil werden Nachrichten versendet. Der einzige Unterschied ist hierbei, dass dabei der „sender“ den Wert „remote“ bekommt, um zu signalisieren, dass die Quelle der Nachricht die mobile Applikation ist. Gesendet werden diese immer, nachdem eine Änderung der Profile stattfindet. Es gibt jedoch eine weitere Nachricht, die die Remote App versendet. Jedes mal, nachdem eine Verbindung mit dem Websocket aufgebaut wurde, sendet sie einen String mit dem Inhalt „fetch“. Auf die Nachricht folgt, dass der Spiegel AI seinen aktuellen Stand der Profile sendet. Dies ist wichtig, damit die lokalen Profildaten der App synchronisiert werden, bevor sie in der Lage ist, Änderungen vorzunehmen. Ansonsten kann es dazu führen, dass Profildaten des Smart Mirrors mit veralteten Daten der App überschrieben werden. Um sich mit dem Websocket-Server zu verbinden, muss die IP-Adresse und der Port des Servers im Format „ws://<server-ip>:<server:port>“ angegeben werden. Der Websocket schließt, sobald die App entweder geschlossen oder in den Hintergrund laufen gelassen wird. Sobald die App wieder geöffnet wird, wird auch die Verbindung zum Websocket hergestellt und sendet den „fetch“-String an den Server.

3.3.2 Remote

Die Remote App ist zuständig für die Änderung einer Profileigenschaft mit dem Namen **state**. Jedes Profil hat einen State, welcher die Positionierung (oder „index“), ID und Anzeigestatus aller ausgewählten Widgets angibt. Mithilfe dieses Status wird die Anzeige des Spiegels festgelegt. Die Änderungen des States in der Remote Ansicht sind nur am ausgewählten Profil möglich. Ist der Wert der Angegebenen ID *-1*, dann handelt es sich hierbei um ein Feld ohne Widget. Das heißt, es wird an der Stelle kein

Name angezeigt und am Spiegel erscheint an dieser Position kein Widget.

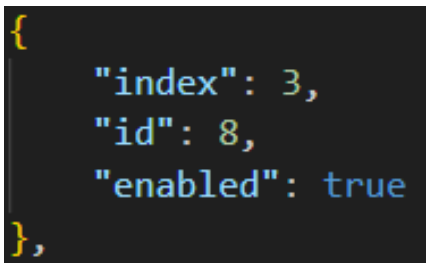

Abbildung 3.6: Beispiel eines State-Eintrags im JSON-Format

Satz

3.4 Testen

Testen

4 Gesichtserkennung

In diesem Kapitel beschreiben wir das Gesichtserkennungssystem, das im **Spiegel AI** Projekt integriert ist. Wir gehen auf die verwendeten Algorithmen, die Trainingsdaten und die Implementierung ein.

4.1 Algorithmen

Beschreiben Sie die Algorithmen, die für die Gesichtserkennung verwendet werden.

4.2 Trainingsdaten

Erläutern Sie die Quelle und Vorbereitung der Trainingsdaten.

4.3 Implementierung

Beschreiben Sie die Implementierung der Gesichtserkennung.

5 Schnittstelle

In diesem Kapitel wird die Schnittstelle des **Spiegel AI** Projekts beschrieben. Wir erläutern die verschiedenen Schnittstellen, die verwendet werden, sowie deren Implementierung und Nutzung.

5.1 Überblick

Geben Sie einen Überblick über die verwendeten Schnittstellen.

5.2 Implementierung

Beschreiben Sie die Implementierung der Schnittstellen.

5.3 Nutzung

Erläutern Sie, wie die Schnittstellen genutzt werden.

5.3.1 Dynamische Widget Anordnung

Erarbeitet von: Marcel Wagner

Die Entwicklung eines Smart Mirrors mit dynamisch anpassbarer Display Anordnung stellt eine technische Herausforderung dar. Das Ziel dieses Bereiches war es, eine benutzerfreundliche und flexible Oberfläche zu schaffen, die sich den individuellen Präferenzen der Benutzer anpasst. Dieser Abschnitt beschreibt ausführlich die Methodik der Implementierung der dynamischen Display Anordnung sowie die dabei aufgetretenen Probleme und deren Lösungen.

Ein essenzieller Bestandteil des Systems war die kontinuierliche Überwachung der 'profiles.json' Datei. Wie diese Datei implementiert wird kann dem Abschnitt (ergänzen) entnommen werden. Hierzu wurde ein periodischer Abrufmechanismus implementiert, der alle 200 Millisekunden die Datei abfragte. Bei jedem Abfrage wurde der aktuelle Inhalt der Datei mit dem vorherigen Zustand verglichen, um Änderungen zu erkennen. Diese Methode stellte sicher, dass Anpassungen in den Benutzerprofilen zeitnah detektiert und umgesetzt wurden.

Aus dieser Datei wird beginnend nach dem aktuellen ausgewählten Profil gesucht.

Zur Identifikation des aktuell ausgewählten Profils wurde eine spezielle Funktion entwickelt. Diese Funktion durchsuchte die Liste der Profile nach dem als ausgewählt markierten Profil und gibt diese zurück. Ist kein Profil ausgewählt, wurde null zurückgegeben, was die Anwendung des Standardzustands ermöglichte. Die Fähigkeit, das aktive Profil zu identifizieren, war entscheidend für die Anpassung der Display Anordnung und stellte sicher, dass die Benutzereinstellungen korrekt umgesetzt wurden.

Als nächster Schritt war ein weiterer zentraler Aspekt der Implementierung die Ermittlung aktuellen Zustands der Widgets. Hierfür wurde eine Funktion entwickelt, die den Standardzustand der Widgets zurückgab, falls kein spezifisches Profil ausgewählt ist. Diese Funktion generierte eine vordefinierte Anordnung der Widgets, die als Ausgangspunkt diente. Ist hingegen ein Profil ausgewählt, wurde der Zustand dieses Profils verwendet. Diese flexible Handhabung ermöglichte es, die Anzeige dynamisch an die individuellen Präferenzen der Benutzer anzupassen.

Abschließend mussten noch auf Basis dieser Daten eine Dynamische Anpassung der HTML Seite vorgenommen werden. Basierend auf dem aktuellen Zustand der Widgets wurden die entsprechenden HTML Elemente ein- oder ausgeblendet und in der gewünschten Reihenfolge angeordnet. Diese Anpassungen wurden durch die Funktion 'updateState' gesteuert, die die Widgets gemäß den Benutzereinstellungen neu positionierte. Die Funktion arbeitete folgendermaßen: Zunächst wird der Container, der die Widgets enthielt, geleert. Anschließend werden die Widgets gemäß der im Profil definierten Reihenfolge wieder hinzugefügt. Dabei wird auch die Sichtbarkeit jedes Widgets entsprechend dem enabled Status beachtet. Widgets, die nicht aktiviert waren, werden ausgeblendet, während aktivierte Widgets sichtbar blieben. Diese dynamische Anpassung ermöglichte es, die Widgets je nach Benutzerprofil in der gewünschten Anordnung und Sichtbarkeit darzustellen.

Aufgetretene Probleme und deren Lösung

Cache Verwaltung: Das am häufigsten aufgetretene Problem war das der Cache Verwaltung des Browsers. Dies stellte eine besondere Herausforderung dar, da durch die regelmäßigen Anfragen an die 'profiles.json' Datei wurde oft veralteter Inhalt aus dem Cache verwendet, anstatt die neuesten Daten abzurufen. Dieses Problem wurde durch gezielte Deaktivierung des Caches für die betreffenden Anfragen gelöst. Der HTTP-Header Cache-Control wurde entsprechend konfiguriert, um sicherzustellen, dass die Anfragen stets frische Daten zurücklieferten. Dies gewährleistete, dass immer die aktuellste Version der profiles.json-Datei abgerufen und verarbeitet wurde, was eine zuverlässige Aktualisierung der Anzeige ermöglichte.

CORS Beschränkung Ein weiteres Problem, welches während der Implementierung aufgetreten ist, war im Zusammenhang mit der Same Origin Policy des Browsers. Diese Sicherheitsrichtlinie verhinderte den Abruf der profiles.json-Datei von einem anderen Ursprung, was die Aktualisierung der Profile erschwerte. Um dieses Problem zu umgehen, wurde ein lokaler Server mit Python erstellt, der die Datei auslieferte. Dieser Server ermöglichte es, die Datei lokal zu hosten und somit die CORS Beschränkungen (Cross-Origin Resource Sharing) zu umgehen. Durch den Zugriff auf diesen lokalen Server konnte die Datei problemlos und sicher abgerufen werden, was die zuverlässi-

ge Aktualisierung der Profile gewährleistete.

6 Ergebnisse

In diesem Kapitel fassen wir die Ergebnisse des Projekts **Spiegel AI** zusammen. Wir gehen auf die erreichten Ziele, die Herausforderungen und die zukünftigen Arbeiten ein.

6.1 Erreichte Ziele

Beschreiben Sie die Ziele, die im Rahmen des Projekts erreicht wurden.

6.2 Herausforderungen

Erläutern Sie die Herausforderungen, die während des Projekts aufgetreten sind.

6.3 Zukünftige Arbeiten

Beschreiben Sie mögliche zukünftige Arbeiten oder Erweiterungen des Projekts.

Stundenliste

Stundenliste Leon Kranner

Kalenderwoche	Stunden	Aufgabe
12	3	Einführungfsveranstaltung
13	3 3	GANNT Diagramm Teambesprechung
14	2 2	Planung mit Hardware-Team Teambesprechung
15	3 1 2	Postererstellung und Hw Displaymessung + postervorstellung Teambesprechung
16	4 2	Display-Projekt aufsetzen Baumarkt Materialien erkunden
17	2 2	Umstrukturierung des Display-Projekts Neues Widget erstellen
18	6	Weitere Widgets und Änderungen an alten Widget
19	3 1	Einkerbungen fräsen MDF Platte auf Maß schneiden
20	2 1	Teambesprechung Umstrukturierung des Stundenplans
21	2	Teambesprechung
22	2	Teambesprechung
23	2	Automatische Aktualisierung der Widgets
24	2 2	Teambesprechung Besprechung Schnittstellen
25	2 1	Teambesprechung Plexiglas überarbeiten
26	2 3 1	Teambesprechung Austausch der Spiegel Folie, Aufbau der Spiegels, Hardware installieren Testen des Displays mit Aufgebauten Spiegels
27	2	Powerpoint erstellung

Fortsetzung auf nächster Seite

Tabelle 6.1 – Fortsetzung von vorheriger Seite

Kalenderwoche	Stunden	Aufgabe
	3	HMTL neu anordnen auf Basis von Json Datei
	2	Besprechung profiles sync

Gesamtstunden: 109

Stundenliste Marco Kuner

Kalenderwoche	Stunden	Aufgabe
12	3	Einführungfsveranstaltung
13	3	GANNT Diagramm
14	3 2 3	Teileliste / GANNT Diagramm Teambesprechung Inventur
15	10 3 2	Technologie-Recherche + Postererstellung Posterdemütigung ertragen und HW Teambesprechung
16	2 8	Teambesprechung Erster Prototyp mit HAAR Cascades
17	2 8	Teambesprechung Neue Version mit DLIB Bibs geschrieben
18	2 2 4	Teambesprechung Recherche über facial Landmark Storage Neue Iteration mit Storage Technologie
19	10 2	Troubleshoot da extrem langsam Teambesprechung
20	2 4	Teambesprechung Recherche zu geeigneter Schnittstelle und Format der Profilerstellung mit profile landsmarks
21	2	Teambesprechung
22	2	Teambesprechung
23	2	Teambesprechung
24	2	Teambesprechung
25	2 6 2	Teambesprechung Implementieren einer Lösung zur automatischen Erkennung eines neuen Gesichts und output der Daten in .json Schnittstellen Thinktank mit David
26	2 5 6	Teambesprechung Ausgabe und automatische Aktualisierung einer genormten profiles.json Implementierung eines neuen Websockets zwischen Raspi und Android in Vorbereitung zur Synchronisation

Fortsetzung auf nächster Seite

Tabelle 6.2 – Fortsetzung von vorheriger Seite

Kalenderwoche	**Stunden**	**Aufgabe**
	2	Recherche zu Technologien zur Synchronisation zwischen Raspi und Android (inotify?)
	2	Besprechung mit remote app Spezialist bzgl. Synchronisationsproblemen
27	4	Vor- und Aufbereiten der Präsentation
	2	Verbessern der readability des Algorithmus
	8	Implementation des Websockets mitsamt Logik für andauernder Synchronisation
	4	Troubleshooting: Gesichtserkennung stürzt ab auf Raspi

Gesamtstunden: 130

Stundenliste David Vollmer

Kalenderwoche	Stunden	Aufgabe
12	3	Einführungsveranstaltung
	4	Setup Gitlab und Drafts
13	4	Erstellung GANNT Diagramm und Lastenheft
	2	Teambesprechung
14	5	Abgabevorbereitung GANNT und Lastenheft
	2	Teambesprechung
	3	Hardwarediskussion und -suche
	2	Überarbeitung GANNT und Lastenheft
15	4	Postererstellung
	3	Vostellung Poster und Hardwaresuche
	6	Setup Flutter und Frontend dev
	2	Teambesprechung
	2	Frontend dev (Navigation)
16	2	Teambesprechung
	3	Frontend dev
17	2	Setup Raspberry Pi
	2	Teambesprechung
18	2	Teambesprechung
19	2	Teambesprechung
	1	Frontend dev (Widget buttons)
20	2	Teambesprechung
21	2	Teambesprechung
22	2	Teambesprechung
	4	Troubleshooting Android SDK
23	2	Teambesprechung
24	2	Besprechung Schnittstellen
	4	Frontend dev (Widgets final)
	2	Teambesprechung
	2	Konfiguration Raspberry Pi
	8	Konfiguration Schnittstellen (Flutter + Server)
	5	Konfiguration Schnittstellen (Spiegel + Server)
25	2	Teambesprechung
	3	Konfiguration Raspberry Pi wifi
	5	Troubleshooting + Testing Websocket
	2	Anpassung Android und iOS (icon, splash, usw.)
	8	Anpassung Datenspeicher, Profile und Websocket

Fortsetzung auf nächster Seite

Tabelle 6.3 – Fortsetzung von vorheriger Seite

Kalenderwoche	Stunden	Aufgabe
	3	Speichern von Widget- und Remotestatus in Profilen
	3	Code Refactoring und Bugfixing
26	1	Besprechung Schnittstellen Profile
	1	Überarbeitung Poster
	2	Teambesprechung
	1	Überarbeitung Websocket-Message
	2	Troubleshooting selected Widgets
	2	Konfiguration Raspberry Pi Gesichtserkennung
	6	Erstellung Powerpoint
27	2	Besprechung profiles sync
	1	Refactoring File Reader
	2	Implementierung profiles sync
	3	Schreiben des Präsentationsskripts
	2	Übung Präsentation
	5	Testen der Gesichtserkennung am Websocket
	3	Testen und Korrigieren profiles sync

Gesamtstunden: 150

Stundenliste Marcel Wagner

Kalenderwoche	Stunden	Aufgabe
12	3	Einführungfsveranstaltung
13	3	GANNT Diagramm
	2	Teambesprechung
	2	Vorbereitung Template
14	2	Teambesprechung
	3	Planung und Hardware Suche
	3	Setup CAD und ersten Entwurf zeichnen
15	3	Poster Erstellung
	2	Besprechung Spiegelrahmen
	2	Detaillierung der CAD Datei
	1	Baumarkt
	2	Teambesprechung
16	3	Einführung Display Programmierung und erste Ansätze
	2	Planung und Besprechung für den Bilderrahmen
	2	Materialien im Baumarkt suchen
	2	Teambesprechung
17	4	Weitere Setup für Display Programmierung
	4	Projekt Besprechung und weitere Programmierung
	4	Weitere Widget Programmierung und Bug Fixing
	2	Teambesprechung
18	2	Display Programmierung (Fertigstellung des Verkehrsinformations Widget)
	2	Teambesprechung
	4	Holz auf Maß schneiden und hobeln
	3	Vorbohren und Bilderrahmen zurechtlegen
19	3	Einkerbungen fräsen
	1	MDF Platte auf Maß schneiden
	1	In MDF Platte Löcher bohren für Befestigung
	1	Zwischenstücke vorne anschrauben
	2	Teambespechung
	2	Holz zusammenleimen und trocknen lassen
20	2	Teambesprechung
	1	Bug Fixing von älteren Widgets
21	2	Teambesprechung

Fortsetzung auf nächster Seite

Tabelle 6.4 – Fortsetzung von vorheriger Seite

Kalenderwoche	Stunden	Aufgabe
	1	Plexiglas auf Maß schneiden
22	2	Teambesprechung
23	3	Erstellung weiter Widgets
	2	Teambespechung
24	2	Teambesprechung
	2	Besprechung Schnittstellen
	2	Bugfixing für das News Widget
25	2	Teambesprechung
	1	Bugfixing der Widget Ansicht
	1	Plexiglas überarbeiten
	2	Löcher bohren und Plexiglas festschrauben
	1	Spiegelfolie aufbringen und Kabel Loch bohren
	5	Schnittstelle zwischen Gesichtserkennung und Display die Grundlagen auf Seite des Displays aufsetzen
26	2	Teambesprechung
	1	Fehler korrigieren am Spiegel
	1	Bugfixing Widget
	3	Austausch der Spiegel Folie, Aufbau der Spiegels, Hardware installieren
	1	Testen des Displays mit Aufgebauten Spiegels
	1	Vorbereitung Präsentation
	2	Vorbereitung Präsentation (Bilder und Aufbau Finalisieren)
	2	Raspberry Pi Gesichtserkennung Testen
	5	Profile aus der Gesichtserkennung auslesen und Speicherin in Raspberry
	2	Bugfixing Browser Problem
27	2	Powerpoint erstellung
	3	HMTL neu anordnen auf Basis von Json Datei
	2	Troubleshooting Chache Probleme
	1	nicht ausgewählte Widgets ausblenden
	2	Vorbereitung Präsentation
	3	Schreiben des Präsentationsskripts
	2	Teambespechung
	1	Studenliste in Dokumentation eintragen

Fortsetzung auf nächster Seite

Tabelle 6.4 – Fortsetzung von vorheriger Seite

Kalenderwoche	**Stunden**	**Aufgabe**
	4	Dokumentation für Widgets schreiben
	1	Dokumentation für Testverfahren schreiben
	3	Verschieden Testvarianten bei den Widgets durchführen
	2	Dokumentation der Dynamischen Widget Anordnung / Überarbeitung Layout

Gesamtstunden: 151

Literaturverzeichnis

[1] JetBrains. „The State of Developer Ecosystem 2023.“ (2023), Adresse: https://www.jetbrains.com/lp/devecosystem-2023/development/ (Zugriff: 30. 06. 2024).

[2] Google LLC. „Dart: The platforms.“ (2024), Adresse: https://dart.dev/overview#platform (Zugriff: 07. 07. 2024).

[3] Google LLC. „Supported deployment platforms.“ (2024), Adresse: https://docs.flutter.dev/reference/supported-platforms (Zugriff: 07. 07. 2024).

Abbildungsverzeichnis