Technische Universität Braunschweig

IAS | INSTITUTE FOR APPLICATION SECURITY

# Software-Entwicklungspraktiktum (SEP)

## ARID - Augmented Reality in Disguise

Software-Entwicklungspraktikum (SEP)
Sommersemester 2024

## Fachentwurf

Auftraggeber:
Technische Universität Braunschweig
Institut für Anwendungssicherheit (IAS)
Prof. Dr. Martin Johns
Mühlenpfordstraße 23
38106 Braunschweig

Betreuerin: Alexandra Dirksen

Auftragnehmer:


| Name | E-Mail-Adresse |
|---|---|
| Amir Fakhim Hashemi | a.fakhim-hashemi@tu-braunschweig.de |
| Ibrahim Abdullah | i.abdullah@tu-braunschweig.de |
| Jadon-Kim Fischer | jadon-kim.fischer@tu-braunschweig.de |
| Mohamed Ali Mrabti | m.mrabti@tu-braunschweig.de |
| Tim Kütemeyer | t.kuetemeyer@tu-braunschweig.de |
| Vyvy Nguyen | Vyvy.nguyen@tu-braunschweig.de |


Braunschweig, 19. Juni 2024

# Bearbeiterübersicht

| **Kapitel** | **Autoren** | **Kommentare** |
|---|---|---|
| 1 | Vyvy | — |
| 1.1 | Vyvy, Amir | — |
| 2 | Ibrahim | — |
| 2.1 | Jadon | — |
| 2.2 | Jadon | — |
| 2.3 | Jadon | — |
| 2.4 | Jadon | — |
| 2.5 | Ibrahim | — |
| 2.6 | Ibrahim | — |
| 2.7 | Ibrahim | — |
| 3 | Tim, Amir | — |
| 4 | Mohamed Ali | — |
| 5 | Sämtliche | Fortlaufend ergänzt |

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# 1 Einleitung

Im Fachentwurf werden die Funktionen und dessen Anforderungen des Produktes dem Auftraggeber fachlich modelliert und erläutert. Es soll also eine übersichtliche graphische und verbale Darstellung der Funktionen erstellt werden. Im folgendem Kapitel wird ein Zustandsdiagramm über die Arbeitsweise des gesamten Systems gezeigt, zudem sind noch für einzelne Funktionen Akitvitätsdiagramme aufzufinden und außerdem wird auf die Erstellung der versteckten QR-Codes eingegangen. In Kapitel Zwei wird das Verhalten der wichtigsten Funktionen als Sequenzdiagramm dargestellt und analysiert. Da das Produkt keine Daten speichert wird Kapitel drei leer gelassen. Im Kapitel der Konfiguration wird eine Übersicht des Codes gegeben und dessen Konfiguration.

Bei dem Projekt ARID - Augemented Reality in Disguise ist die Hauptfunktion das Entschlüsseln der versteckten QR-Codes mithilfe des Monocle. Der User, der das Monocle trägt, bedient das vorhandene Touchpad um ein Bild aufzunehmen. Dieses Bild wird dem Host weitergeleitet, dort wird die Nachricht angezeigt und dann entschlüsselt. Diese Nachricht wird dem Host und auf dem Monocle angezeigt. Mögliche Host sind Computer, Laptops sowie mobile Endgeräte.

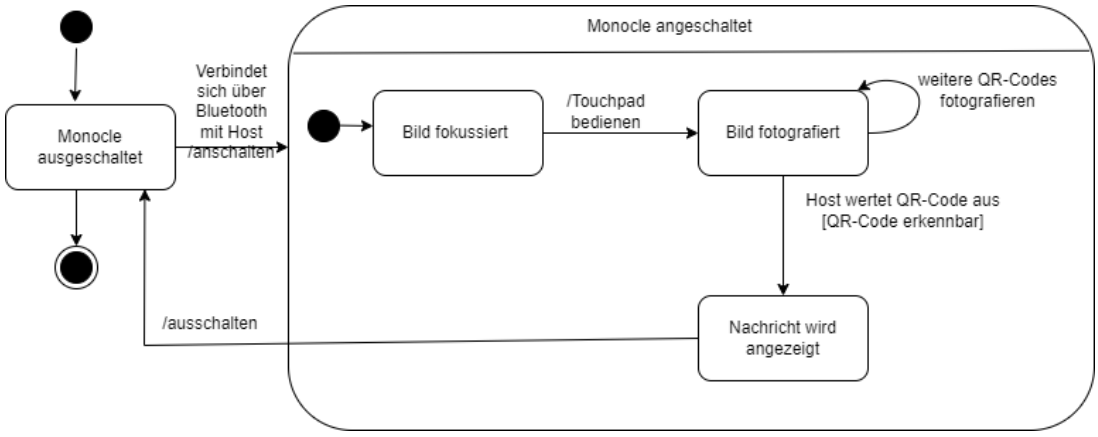


Abbildung 1.1: Zustandsdiagramm des Anwendungsprozesses

In dem Zustandsdiagramm ist ein Überblick des Systems zusehen. Das Monocle wird eingeschaltet und über Bluetooth kann sich dieser mit dem Host verbinden. Wenn das Monocle eingeschaltet ist, fokussiert dieser sich auf den QR-Code, um ein verschwommenes Bild zu vermeiden. Dann kann das Bild fotografiert werden, indem das Touchpad bedient wird. Das Bild wird dem Host geschickt. Ist das Bild scharf genug, wird dieser dann entschlüsselt und die Botschaft wird

dem Monocle gesendet, welcher dann die Nachricht anzeigt. Daraufhin kann das Monocle wieder ausgeschaltet werden.

## 1.1 Projektdetails

In diesem Abschnitt werden einige interessante Sachverhalte mit Hilfe von Aktivitätsdiagrammen genauer erläutert:

- QR-Code erstellen und verstecken
- QR-Code erkennen und anzeigen
- Ver- und Entschlüsselung

### 1.1.1 QR-Code erstellen und verstecken

Der Python-Code 'qrcode-encode' generiert aus der verschlüsselten Botschaft einen QR-Code. Mithilfe einer Deep-Learning KI namens Stable Diffusion wird ein Bild generiert, in dem der QR-Code mittels Steganographie versteckt wird. Es existieren verschiedene WebUIs mit denen die Bilder anhand von text-to-image oder image-to-image generiert werden können. Grundlage für das zu entstehende Bild sind Prompts, schriftliche Eingabeaufforderung, die der KI instruieren, wie das zu generierende Bild auzusehen hat. Ein wichtiger Punkt der zu beachten gilt, ist die Relation zwischen QR-Code und Kunst. Der QR-Code muss insofern in dem Bild integriert werden, so dass es vom Monocle erkannt werden kann, er aber für den bloßer Betrachter nicht zu erkennen ist. Um diese Gewichtung zu steuern, existieren verschiedene Parameter.

### 1.1.2 QR-Code erkennen und anzeigen

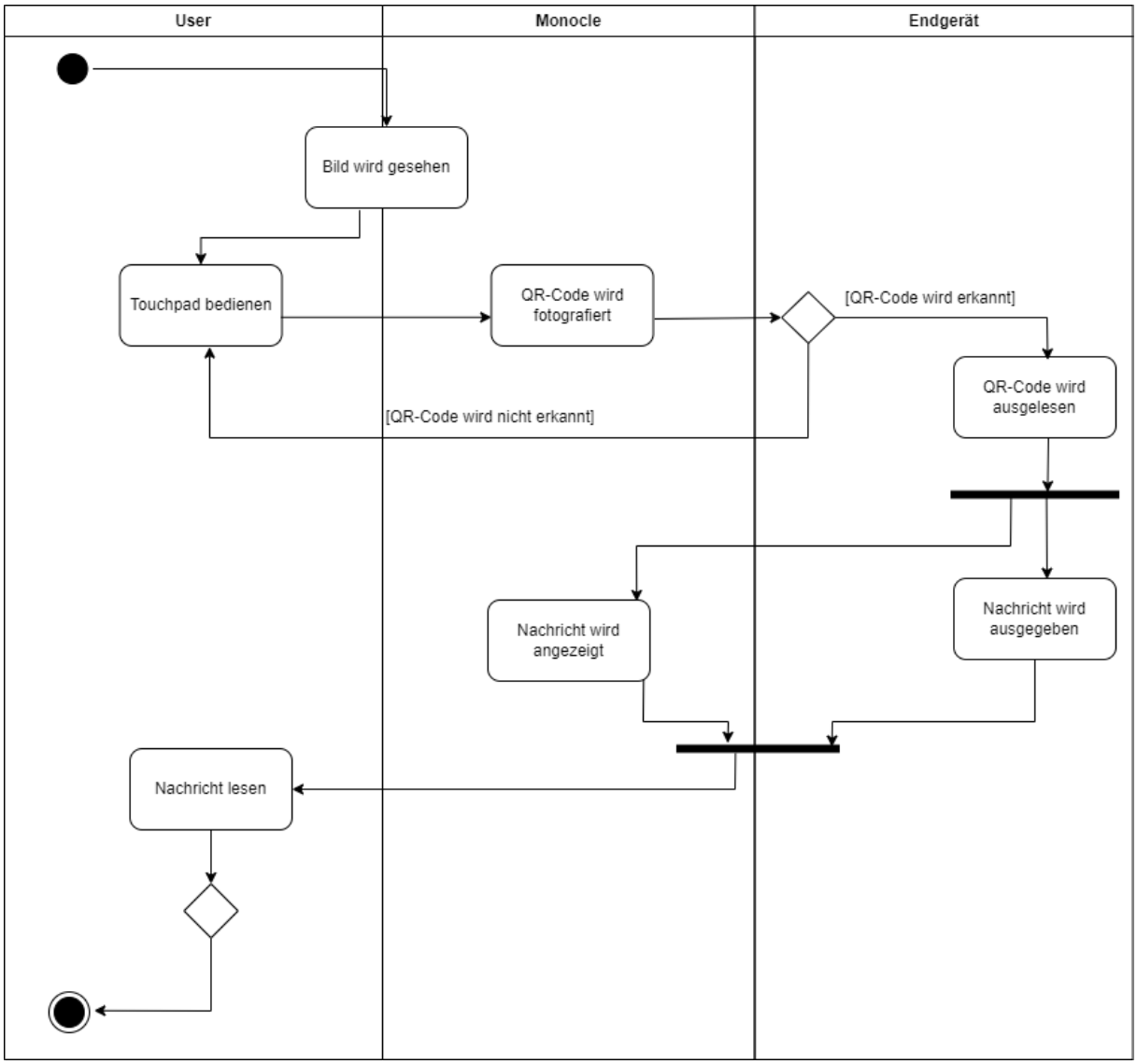


Abbildung 1.2: Aktivitätsdiagramm erkennen und anzeigen von QR-Codes

Der User bedient das Touchpad des Monocles um damit den QR-Code abzufotografieren. Ist das Bild verschwommen oder nicht lesbar, muss der QR-Code erneut abfotografiert werden. Wenn es jedoch auswertbar ist, wird das fotografierte Bild dem Host geschickt und daraufhin ausgelesen. Die Botschaft wird dann sowohl dem Monocle-Display angezeigt als auch dem Host-Endgerät. Anschließend kann die Nachricht dann vom Verwender gelesen werden.

### 1.1.3 Ver- und Entschlüsselung

Das Monocle kann verschiedene Standard-QR-Codes auslesen, jedoch können auch speziell verschlüsselte und versteckte QR-Codes ausgelesen werden. Für die verschlüsselten Botschaften wurde der Advanced Encryption Standard in der Betriebsart Cipher Block Chaining verwendet. Der Key wird dabei "hard gecoded". Da es sich bei der AES Verschlüsselung um eine symmetrische Verschlüsselung handelt, ist der Key für Ver- und Entschlüsselung derselbe. AES wird weltweit als sehr sicher angenommen, so kann eine vertrauchliche Verschlüsselung der Botschaft gewährleistet werden.

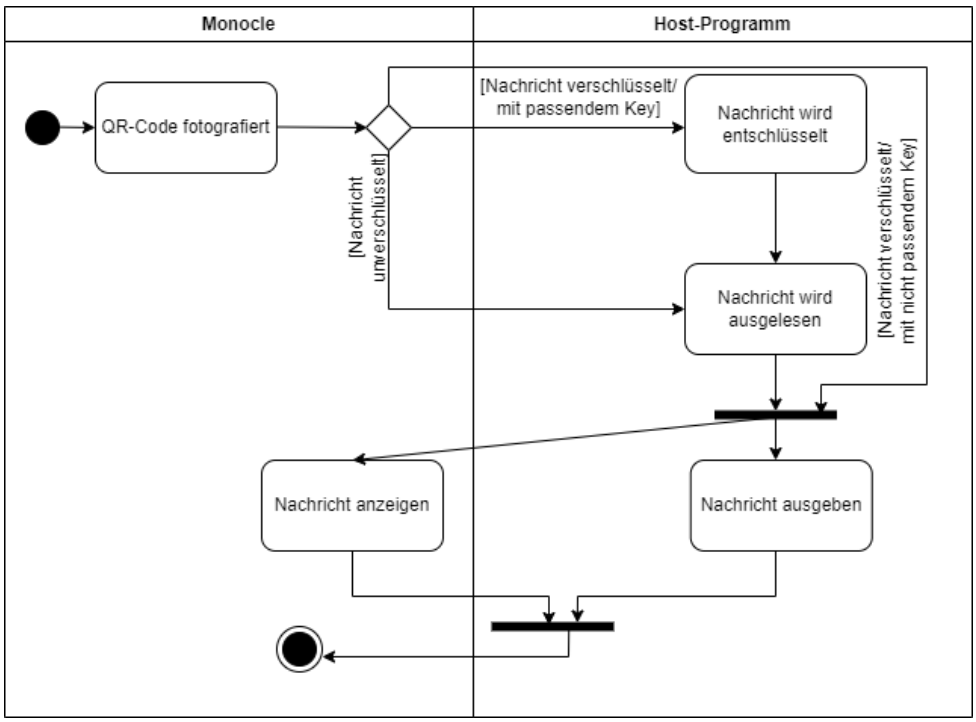


Abbildung 1.3: Aktivitätsdiagramm QR-Codes entschlüsselte Nachricht anzeigen

Die QR-Codes werden anhand des Monocle abfotografiert. Daraufhin entschlüsselt der Host die Nachricht und zeigt dann die Botschaft an. Das Monocle kann jedoch auch unverschlüsselte Botschaften anzeigen lassen.

# 2 Analyse der Produktfunktionen

In diesem Kapitel analysieren wir die im Pflichtenheft beschriebenen Funktionen und werden diese mithilfe von Sequenzdiagrammen visualisieren. Jede Funktion wird dabei auf ihre Anforderungen und ihre Rolle innerhalb der Gesamtsysteme untersucht. Diese Diagramme sollen dabei helfen, zu verstehen, welche Komponenten beteiligt sind und wie die einzelnen Schritte zwischen den verschiedenen Komponenten ablaufen.

## 2.1 Analyse von Funktionalität F10: Automatisches Ein- und Ausschalten

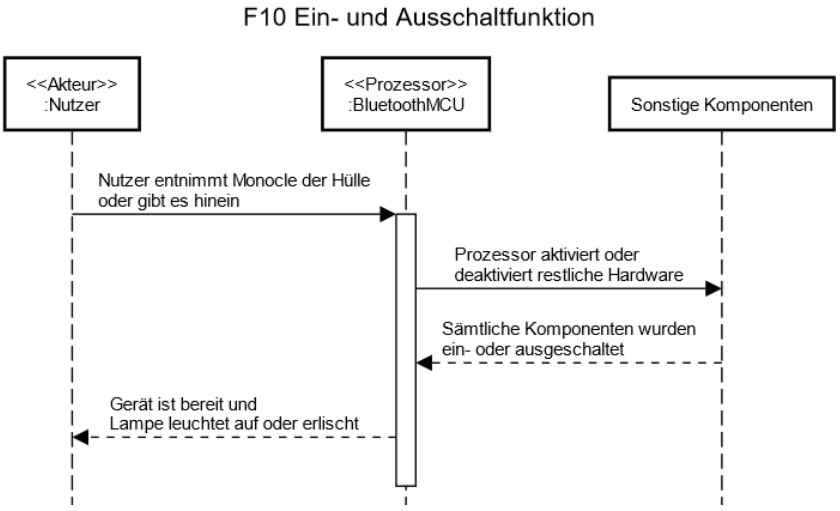


Abbildung 2.1: F10 als Sequenzdiagramm

Das Monocle-Gerät ist in der Lage sich automatisch ein- und auszuschalten. Um diese Funktionalität zu realisieren erkennt das Gerät per Sensor, ob es sich in dem für das Gerät vorgesehenen Gehäuse befindet. Bei dem Ein- und Ausschalten spielt sowohl der Nutzer und spätere Träger des Monocle eine Rolle, als auch der Prozessor, als zentrale Hardware, sowie weitere im Pflichtenheft aufgeführte Hardware-Komponenten. Entnimmt der Nutzer nun das Monocle-Gerät aus seinem Gehäuse, schlägt der dafür konzipierte Sensor aus und benachrichtigt den im Standby befindlichen Bluetooth-MCU-Prozessor. Dieser fährt daraufhin sämtliche Komponenten, wie

etwa die Omnivision OV 5640-Kamera hoch und wartet auf dessen Rückmeldung. Sind sämtliche Rückmeldungen bei dem Prozessor eingegangen, meldet der Prozessor die Bereitschaft des Monocle-Geräts über eine kleine Lampe außen am Prozessor zurück. Der Ausschalte-Vorgang verhält sich hierbei analog.

## 2.2 Analyse von Funktionalität F20: Kommunikation zwischen Endgerät und Monocle

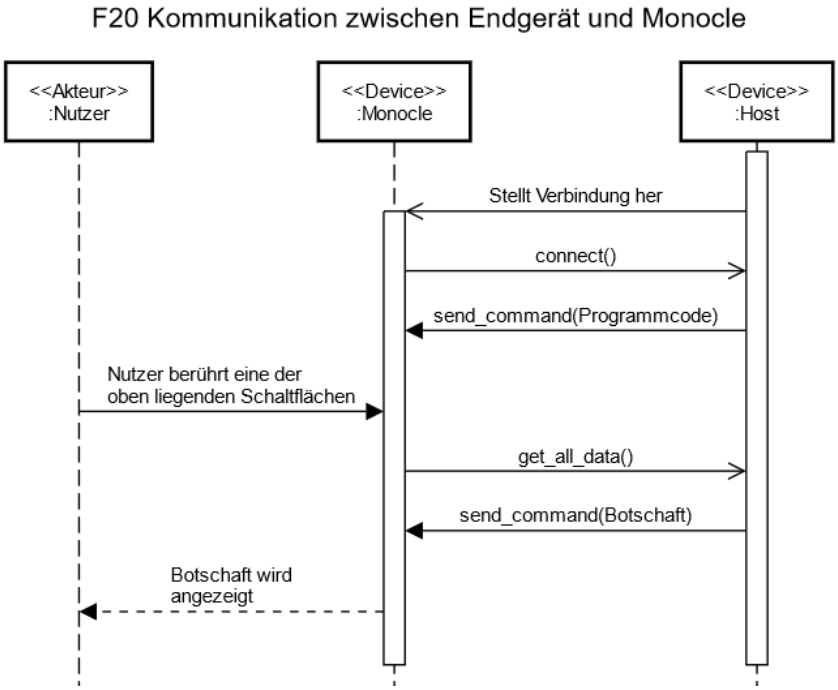


Abbildung 2.2: F20 als Sequenzdiagramm

Um den Anforderungen unserer Monocle-Anwendung gerecht zu werden, müssen das Monocle und das Host-Gerät beim Starten der Anwendung, sowie bei jedem Einlesevorgang in der Lage sein, Daten untereinander auszutauschen. An dieser Stelle kommt die im Pflichtenheft beschriebene Bluetooth Low Energy-Kommunikation (BLE) in das Spiel. Die Akteure der BLE-Kommunikation sind dabei wie folgt miteinander in Verbindung zu setzen: Sobald das Monocle-Gerät wie in 2.1 beschrieben hochgefahren ist und die Anwendung auf dem Host-Gerät ausgeführt wird, unternimmt der Host einen Verbindungsversuch zum Monocle und der auf dem Monocle auszuführende Programmcode wird mithilfe der Funktion „send_command()“ versucht zu übermitteln. Dieses erwidert bei funktionierender BLE-Kommunikation den so genannten Handshake mittels der asynchronen „connect()“-Funktion, wodurch die Verbindung erfolgreich hergestellt wird. Bei jeder weiteren Berührung durch eines der Touchpads auf dem Monocle, unternimmt dieses wie unten Beschrieben einen Einlesevorgang. Innerhalb dieses Einlesevorgangs kommunizieren Monocle und Host abermals miteinander, um erst das eingelesene Bild und später die entschlüsselte Nachricht miteinander auszutauschen. Dabei kommen die Funktionen „get_all_data()“ zum asynchronen Senden einer Fotoaufnahme an das Host-Gerät und „send_command()“ zum übermitteln der entschlüsselten Botschaft zum Tragen. Die übertragene Nachricht wird zuletzt auf dem OLED-Display des Monocle dem Träger angezeigt, wie Sie der

Analyse 2.4 entnehmen können.

## 2.3 Analyse von Funktionalität F30: Einlesen per Touchpads

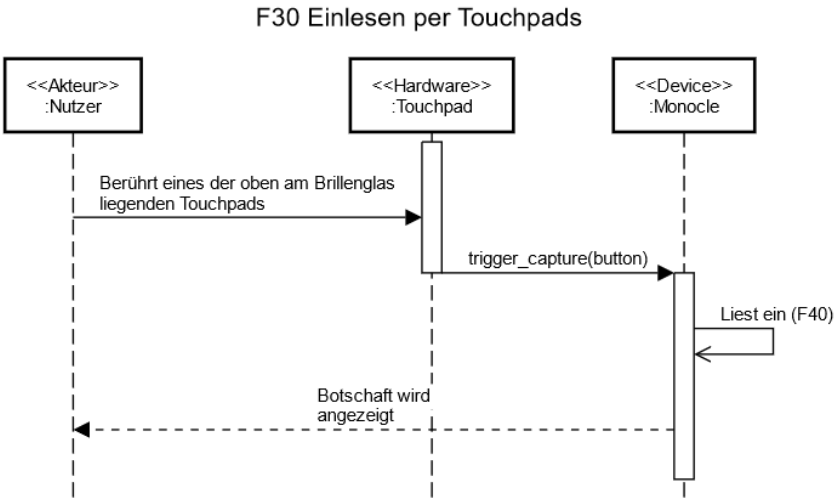


Abbildung 2.3: F30 als Sequenzdiagramm

Der in F40 beschriebene Einlesevorgang eines QR-Codes findet nur auf Nachfrage des Trägers statt. Dieser Kommuniziert der Anwendung seine Absicht über eines der beiden oberhalb des Brilenglases liegenden Azoteq IQS620A Touch Controllern. Hierzu müssen die Touchpads wie in F10 beschrieben von dem Prozessor des Monocle hochgefahren worden sein. Ein weiterer Akteur bei dem Einlesen des QR-Codes per Touchpad ist unsere auf dem Monocle-Gerät ausgeführte Anwendung. Diese wird per „trigger_capture(button)“-Funktion über die Berührung des Trägers über eines der Touchpads informiert. Zur Folge hat der Funktionsaufruf schließlich, dass der in Analyse 2.4 beschriebene asynchrone Einlesevorgang in Kraft tritt. Der mit der Berührung der Touchpads initialisierte Vorgang wird durch das Anzeigen der Nachricht im Display des Monocle abgeschlossen, wie Ihnen die Analyse der Funktionalität 2.6 erläutert.

## 2.4 Analyse von Funktionalität F40: Einlesen von QR-Codes

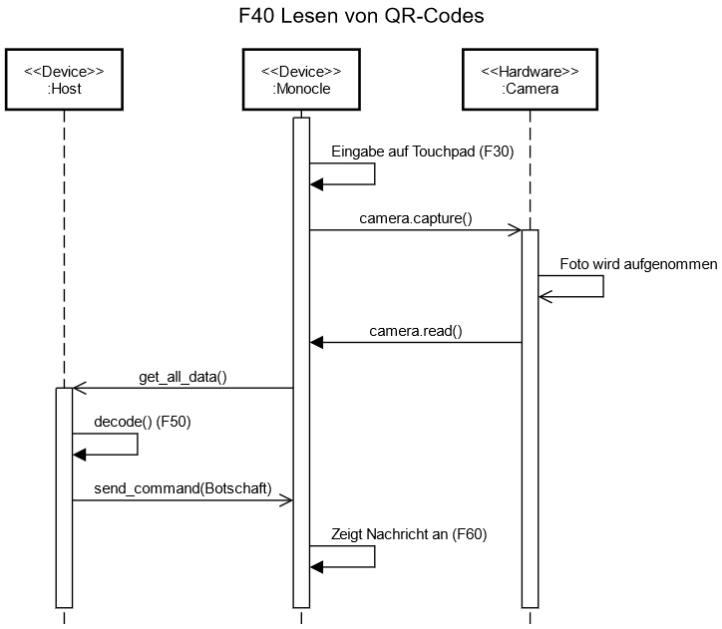


Abbildung 2.4: F40 als Sequenzdiagramm

Bei dem Einlesevorgang von QR-Codes durch das Monocle-Gerät handelt es sich um eine Kombination der vorangegangenen Funktionalitäten, dessen maßgebliche Akteure neben dem Träger auch das Host-Gerät, das Monocle selbst, sowie dessen Kamera sind. Das Verhalten des Monocle wird hierbei natürlich durch unsere Anwendung definiert. Empfängt das Monocle eine Eingabe auf einem der Touchpads, wie in Funktionalität 2.3 beschrieben, wird diese als Anfrage des Trägers zur Erkennung eines QR-Codes interpretiert. Unmittelbar nach der Berührung eines der Touchpads wird deshalb über die interne Funktion „capture()“ der Kamera, eine asynchrone Foto-Aufnahme derselben in Gang gesetzt. Ist das Foto aufgenommen, lässt es sich per „read()“-Funktion aus dem Zwischenspeicher der Kamera auslesen. Diese Foto-Aufnahme wird anschließend jedoch noch an das Host-Gerät per „get_all_data()“ asynchron weitergeleitet, wo die in Analyse 2.5 erläuterte Verschlüsselung durchgeführt wird. Zuvor wird die verschlüsselte Botschaft des QR-Codes über die „decode()“-Funktion aus der für die Extrahierung des QR-Codes vorgesehene Bibliothek entnommen. Die entschlüsselte Botschaft wird zuletzt von dem Host-Gerät an das Monocle zurückgegeben, welches die Botschaft gemäß Funktionalität 2.6 über das Display dem Träger des Geräts zugänglich macht. Genauer wird über die aus Funktionalität 2.2 bekannte „send_command()“-Funktion der gesamte Befehl zum Anzeigen einer Nachricht auf dem Monocle übermittelt

## 2.5 Analyse von Funktionalität F50: Entschlüsselung von Nachrichten

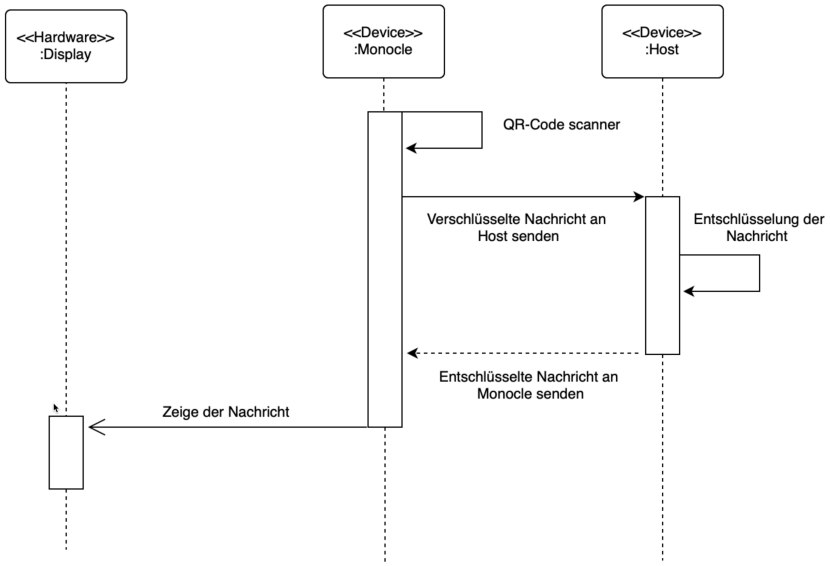


Abbildung 2.5: F50 als Sequenzdiagramm

Im Sequenzdiagramm (2.8) wird die Funktionalität beschrieben, wie Nachrichten, die im QR-Code versteckt sind, entschlüsselt werden. Zuerst sendet das Monocle das aufgenommene Bild an den Host. Der Host durchsucht das Bild nach einem QR-Code. Sobald er den QR-Code findet, entschlüsselt der Host ihn. Dieser Prozess umfasst das Scannen des QR-Codes, das Dekodieren der darin enthaltenen verschlüsselten Informationen mit unserer Entschlüsselungsfunktion und die Umwandlung dieser Daten in eine lesbare Nachricht. Es ist wichtig zu beachten, dass die Nachricht zuvor verschlüsselt und mithilfe unserer Verschlüsselungsfunktion im QR-Code eingebettet wurde. Das entschlüsselte Ergebnis wird dann zurück an das Monocle gesendet. Schließlich zeigt das Monocle die entschlüsselte Nachricht auf dem Display an.

## 2.6 Analyse von Funktionalität F60: Anzeigen der Nachrichten

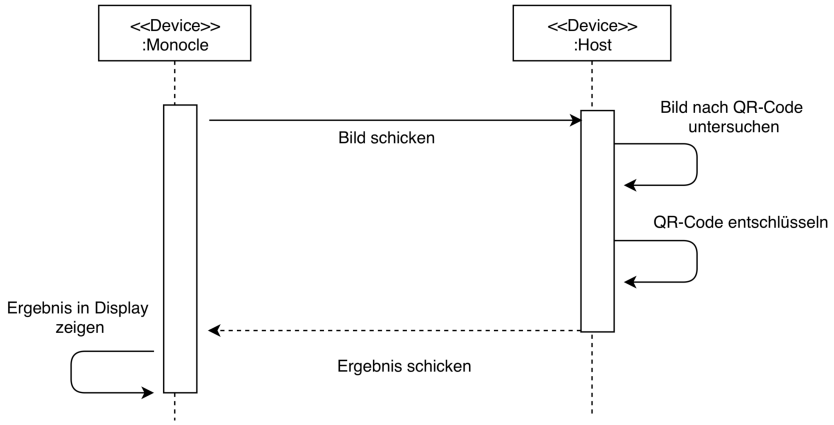


Abbildung 2.6: F60 als Sequenzdiagramm

Bei der Anzeige der Nachricht sollte bereits ein Foto aufgenommen und an den Host gesendet worden sein, wo die Nachricht entschlüsselt wird. Die entschlüsselte Nachricht wird anschließend dann an das Monocle übertragen. Wenn dieser Prozess korrekt durchgeführt wurde, zeigt das Monocle die entschlüsselte Nachricht auf dem Display an. Dieser Ablauf stellt sicher, dass die Nachricht zuverlässig und korrekt verarbeitet wird.

## 2.7 Analyse von Funktionalität F70: Funktionalität der Kamera

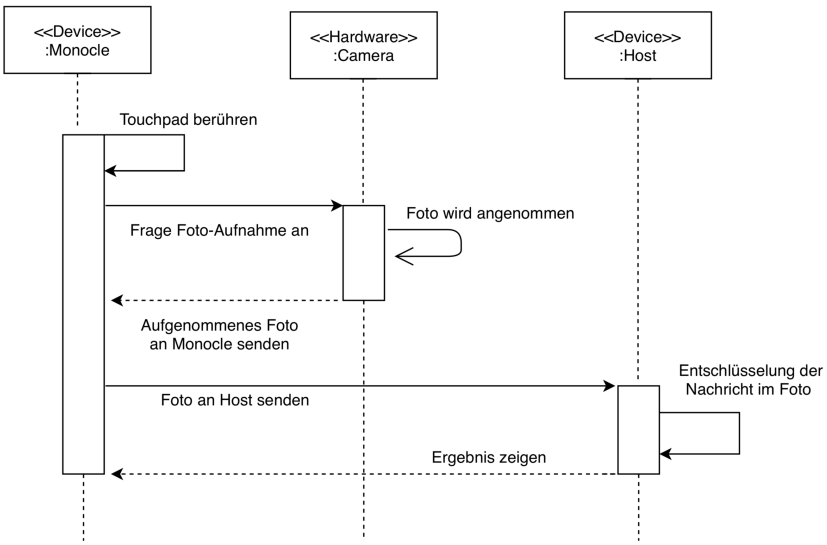


Abbildung 2.7: F70 als Sequenzdiagramm

Im Sequenzdiagramm (2.7) wird die Funktionalität der Kamera des Monocle beschrieben. Zuerst berührt man auf das Touchpad, um mit der ‘capture()‘-Funktion ein Foto aufzunehmen. Die Kamera nimmt dann ein Foto auf und sendet die Bilddaten an das Monocle. Das Monocle leitet diese Bilddaten anschließend an den Host weiter. Der Host verarbeitet das Bild und entschlüsselt die Nachricht darin. Wie in Funktionalität 2.4 beschrieben, sucht er nach einem QR-Code und dekodiert die Nachricht. Schließlich zeigt das Monocle das entschlüsselte Ergebnis an. Diese Reihenfolge von Aktionen stellt sicher, dass die Bilddaten korrekt verarbeitet werden und die enthaltenen Daten erfolgreich gelesen werden können.

# 3 Datenmodell

In diesem Projekt werden keine Daten dauerhaft gespeichert, die eine ausführliche Darstellung von Entities und Beziehungen erfordern würden. Das System speichert weder Nutzerdaten noch anderweitige Informationen in einer relationalen Datenbank, XML-Dateien oder ähnlichen Formaten.

Das Projekt konzentriert sich auf die Erkennung und Dekodierung von QR-Codes in Bildern mittels Augmented Reality, ohne dabei Daten persistent zu speichern. Sämtliche verarbeiteten Informationen sind flüchtig und werden ausschließlich im Arbeitsspeicher gehalten, um nach erfolgreicher Entschlüsselung der QR-Codes sofort genutzt und anschließend verworfen zu werden.

Die Haupt Funktionalitäten unseres Projekts beinhalten:

- Erkennung von QR-Codes in Bildern
- Dekodierung der QR-Codes
- Erstellen verschlüsselter QR-Codes für die Einbettung in Kunstwerke
- Einbettung von QR-Codes in AI Kunstwerken

Diese Funktionalitäten erfordern keine dauerhafte Speicherung der Daten, da alle Schritte direkt im Speicher ablaufen und keine Notwendigkeit besteht, die Daten nach der Verarbeitung zu bewahren.

Aus diesem Grund ist das Kapitel “Datenmodell“ in unserem Projekt nicht relevant.

# 4 Konfiguration

In diesem Abschnitt wird erläutert wie Konfiguration der Bearbeitung und der Anwendung vorgenommen wird. Da die Konfigurationen schon direkt im Code gechrieben sind, haben wir keine Separate config-Dateien. Unser Code besteht aus zwei Python-Dateien, und das heißt, dass wir zwei Konfigurationen haben:

- **Konfiguration für brilliant.py** : brilliant.py, ist ein Skript, das die Verbindung und die Kommunikation mit dem Monokel ermöglicht. Alle relevanten Einstellungen werden direkt im Code festgelegt und in Abbildung 4.1 dargestellt. Um die Konfiguration anzupassen, können Sie den Quellcode von "brilliant.py"mit einem Texteditor Ihrer Wahl öffnen, und da direkt bearbeiten. Die wichtige Parametern in brilliant.py sind:
  - **UART_SERVICE_UUID**: Die UUID des UART-Dienstes, der für die Kommunikation mit der Monocle verwendet wird.
  - **UART_RX_CHAR_UUID**: Die UUID des Empfangscharakteristikums für den Datenübertragungsdienst.
  - **UART_TX_CHAR_UUID**: Die UUID des Übertragungscharakteristikums für den Datenübertragungsdienst.
  - **DATA_SERVICE_UUID**: Die UUID des Datenübertragungsdienstes der Monocle.
  - **DATA_RX_CHAR_UUID**: Die UUID des Empfangscharakteristikums des Datenübertragungsdienstes.
  - **DATA_TX_CHAR_UUID**: Die UUID des Übertragungscharakteristikums des Datenübertragungsdienstes.
- **Konfiguration für main.py:** : Wie beim brilliant.py, main.py ist auch ein Skript, und die Konfiguration ist in der Code geschrieben. Main.py enthält die Steuerlogik und die anwendungsspezifischen Funktionen. Für die Anpassung der Konfiguration, ist es genau sowei bei brilliant.py, der Quellcode mit einem Texteditor öffnen, und die Konfiguration ändern. Die wichtigste Einstellungen sind auch im Code festgelegt und in Abbildung 4.2 erklärt.

```
# brilliant.py

class Monocle:
    UART_SERVICE_UUID = "6e400001-b5a3-f393-e0a9-e50e24dcca9e"
    UART_RX_CHAR_UUID = "6e400002-b5a3-f393-e0a9-e50e24dcca9e"
    UART_TX_CHAR_UUID = "6e400003-b5a3-f393-e0a9-e50e24dcca9e"

    DATA_SERVICE_UUID = "e5700001-7bac-429a-b4ce-57ff900f479d"
    DATA_RX_CHAR_UUID = "e5700002-7bac-429a-b4ce-57ff900f479d"
    DATA_TX_CHAR_UUID = "e5700003-7bac-429a-b4ce-57ff900f479d"
```

Abbildung 4.1: Darstellung

```
# "Waiting" anzeigen auf dem Display
text = display.Text('Waiting...', 100, 0, display.WHITE, justify=display.TOP_LEFT)
display.show(text)

# Auslösen der Bildaufnahme
def trigger_capture(button):
    # Maximale Länge für Bluetooth-Daten
    len = bluetooth.max_length()
    # "Capturing" anzeigen, wenn ein Bild erfasst wird
    text = display.Text('Capturing...', 100, 0, display.WHITE, justify=display.TOP_LEFT)
    display.show(text)
    # Erfassen eines Bildes mit der Kamera
    camera.capture()
    time.sleep_ms(100)
    # Lesen und Senden der Daten mit der Hilfe von Bluetooth
    while data := camera.read(bluetooth.max_length() - 4):
        led.on(led.GREEN)
        while True:
            try:
                # Senden der Daten über Bluetooth
                bluetooth.send((b"img:" + data)[:len])
            except OSError:
                continue
            break
    led.off(led.GREEN)
    # Senden einer Abschlussnachricht über Bluetooth
    bluetooth.send(b'end:')
    # "Done" auf dem Display anzeigen wenn alles fertig ist
    done = display.Text('Done', 100, 0, display.WHITE, justify=display.TOP_LEFT)
    display.show(done)
# Callback-Funktion für das Auslösen der Bildaufnahme bei Berührung
touch.callback(touch.EITHER, trigger_capture)
```

Abbildung 4.2: Darstellung

# 5 Glossar

(Alphabetisch und absteigend sortiert)

AES - Steht für Advanced Encryption Standard, welches eine Blockchiffre ist, die zum Ver- und Entschlüsseln von Informationen verwendet wird.

Sequenzdiagramm - UML-Diagramm, das seinen Fokus auf die Kommunikation zwischen Komponenten der Hardware und Software legt, insbesondere auf deren genauen Ablauf.

Stable Diffusion - Open Source Deep Learning KI mit der Bilder erstellt werden können.

UUID - Steht für Universally Unique Identifier (universell eindeutiger Bezeichner).

UART - Steht für Universal Asynchronous Receiver-Transmitter.

WebUI - Webbrowser mit denen eine Software angewendet werden kann ohne diese dabei herunterzuladen.

Zustandsdiagramm - UML-Diagramm, das sich auf die Zustände eines Produkts oder dessen Software konzentriert, sowie deren Übergänge und Subzustände.