**DIPLOMARBEIT**

DermaAI: Intelligente Hautanalyse

Gesamtprojekt

Entwicklung eines KI-Modells zur Detektion und Klassifikation pigmentierten Hautläsionen und dessen Integration in eine Mobile App

Diplomarbeitsnummer  
5AHINF-24/25-DA11

**Datenbank, Verwaltung der medizinischen Daten**

Jonas Maier 5AHINF Betreuer: Dipl.-Ing. Dr.

Gerhard Gaube

**Mobile Integration, Frontend**

Jonas Bogensberger 5AHINF Betreuer: Dipl.-Ing. Dr.

Michael Wimmer

**KI-Modell in Python, API und Appanbindung**

Daniel Jessner 5AHINF Betreuer: Dipl.-Ing. Dr.

Gerhard Gaube

Schuljahr 2024/25

Abgabevermerk:

Datum: TT.MM.JJJJ übernommen von:

# Allgemeines / Projektübersicht

## Projektbeschreibung

## Projekt und Schnittstellen

## Funktionale Anforderungen

## Use Cases

## Nicht Funktionale Anforderungen

## Projektplanung

## Variantenbildung

## Machbarkeitsstudie

## Projektziele

## Benötigte Ressourcen

## Entwicklungsmethodik

## Kommunikations- und Berichterstattungsstrategie

## Softwarearchitektur

## Aktivitätsdiagramme

## Sequenzdiagramme

## Komponentendiagramme

## Verteilungsdiagramme

## Softwarediagramme / Komponenten

## SW Programme

## SW Komponenten

# Projektdurchführung

## Webscraping & labeln der Daten

### Main.py

Das mit Abstand wichtigste Thema bei diesem Projekt ist die Bildbeschaffung und Bildbearbeitung, deshalb ist es auch das, was mich am längsten zum Implementieren gebraucht hat. Um ein zufriedenstellendes Ergebnis zu erhalten habe ich die bekanntesten und besten medizinischen Datenbanken durchsucht und bin letztendlich auf das Isic-Archive gekommen. Das Isic-Archive ist eine Kollaboration medizinischen Bildern von diversen Krankheiten und Erscheinungen und da es eine gute Dokumentation zum öffentlichen Endpunkt gibt, habe ich schnell meine Wahl getroffen. Meine Wahl für diesen Anwendungsfall fiel schnell auf die Programmiersprache **Python**, da diese viele Bibliotheken für diese Situation bereitstellt und einfach und schnell erstellt ist. Die wichtigsten Packages, welche ich für das main script verwendet habe, sind die folgenden:

* **BeautifulSoup4 – 4.12.3 🡪** Bibliothek für das sammeln von Informationen aus dem Internet.
* **Re** 🡪 Prüft Text auf RegularExpressions um nach bestimmten Sachen zu filtern.
* **Subprocess** 🡪 Starten eines neuen Prozesses um den **Python GIL** zu umgehen.
* **Json 🡪** ermöglicht mir es JSON-strings zu parsen.
* **Time 🡪** für Diagnostik und Zeitmessung
* **Os 🡪** simple Befehle für Informationsbeschaffung zum OS
* **Sys** 🡪 Zugang zu bestimmen Objekten vom OS
* **Requests 🡪** ermöglicht http requests

#### Fetch\_from\_isic\_archive(…)

Wir starten mit dem entrypoint des mainscripts:

A screen shot of a computer code

Description automatically generated

Bevor die Funktion **fetch\_from\_isic\_archive(…)** aufgerufen wird halten wir mithilfe vom **time.perf\_counter()** die Zeit zum Zeitpunkt der Exekution des Scriptes fest. Nachdem die Funktion ausläuft oder man sie beendet, wird noch einmal die Zeit festgehalten und letztendlich subtrahiert, um die Zeit seit dem Start des Skriptes zu erhalten. Wir werden alle genannten und verwendeten Funktionen nach der Reihe nach in dieser Dokumentation aufarbeiten.

A close up of a code

Description automatically generated

Funktionen, auf welche später genaueres eingegangen wird:

* fetch\_html 🡪 gibt den HTML-Code des bereitgestellten Links zurück
* to\_json 🡪 gibt den bereitgestellten string als JSON zurück
* convert\_data 🡪 normalisiert den Inhalt des strings

Wichtige Variablen für diese Funktion:

* data 🡪 für uns Interessanter Inhalt der gesammelten Daten
* next\_link 🡪 Link zur nächsten Page der Datenbank
* content 🡪 normalisierter Inhalt der gesammelten Daten

Gedankengang hinter den gezeigten Zeilen ist, die API von ISIC mithilfe des **links** abzufragen und in eine Zwischenvariable zu speichern. Damit wir einfach auf die Inhalte der Request zugreifen können, habe ich das erhaltene Objekt zu einem JSON konvertiert. Mithilfe der JSON können wir uns nun den Link zur nächsten Page der API zwischenspeichern und letztendlich am Ende der Funktion wird diese Zwischenvariable wieder verwendet.

A screenshot of a computer program

Description automatically generated

Funktionen zur Datensammlung:

* find\_regex(…) 🡪 Gibt das Muster zurück, fall es existiert
* download\_images(…) 🡪 Startet neuen subprocess
* fetch\_from\_isisc\_archive(…)🡪 Rekursiver Aufruf der derzeitigen Funktion

Wichtige Variablen:

* diagnosis 🡪 Diagnose erhalten aus dem Objekt gefiltert mittels RegEx
* picture 🡪 Bildobjekt enthält Diagnose und Bilddaten als Variablen

Der Code hier ist im Endeffekt simpel und leicht zu verstehen, jedoch schadet eine Erklärung zum Gedankengang nie. Wir beginnen den Code mit einer Schleife, welche jedes Objekt in dem in den vorherigen Schritten erzeugten Array durchläuft. Da ein Absturz des Programms natürlich unerwünscht ist, habe ich einen try-catch Block um den folgenden Programmcode gelegt. Wie bereits erklärt suchen wir in der Liste von Objekten nach einer Diagnose und Bild, falls dieser Schritt nicht erfolgreich ist, wird diese Iteration in der Schleife übersprungen. Wenn wir Diagnose und Downloadlink zum Bild haben, geben wir das Bildobjekt an eine weiter Funktion weiter, auf welche später eingegangen wird. Sobald alle Objekte der Liste durchlaufen wurden verwenden wir den bereits erhaltenen Link zur nächsten Page und beginnen den Schritt von vorne.

#### download\_images(…)



Diese Funktion ermöglicht mir es das Bildobjekt in als JSON-Text in das nächste Python Skript asynchron weiterzugeben. Als Argumente für diese Funktion habe ich eine Liste definiert, welche am ersten Platz den **Python-Interpreter** weitergibt. Das zweite Argument ist das nächste Skript, welches mit den im 3. Argument bereitgestellten Daten die Bilder herunterlädt, jedoch dazu mehr im nächsten Kapitel. Nach diesem Funktionsaufruf kehrt der Interpreter wieder zu **fetch\_from\_isic\_archive()** zurück.

#### RegEX

RegEx ist ein wichtiger Bestandteil dieser Datei, da ohne diese keiner Filterung nach den gewünschten **Links** möglich wäre. Das erstellen von RegEx ist mittels Internetseiten und künstlicher Intelligenz zwar vereinfacht, jedoch ist es wichtig sich selbst im klaren zu sein was passiert. Die verwendete Regular Expression sieht wie folgt aus:

outcome\_diagnosis\_regex\_ENG\_ISIC = r'\'outcome\_diagnosis\':\s\*\'([^\']+)'

**Bestandteile der RegEx:**

**\'outcome\_diagnosis\'**

* Was wird gesucht?  
  Der Schlüssel 'outcome\_diagnosis', der in einfachen Anführungszeichen (') steht.
* **Details:**
  + **\':** Ein einzelnes Hochkomma. Der Backslash (\) dient als Escape-Zeichen, damit das Hochkomma als Literal interpretiert wird.
  + **outcome\_diagnosis:** Der Name des Schlüssels.
  + **\':** Das abschließende Hochkomma.

**2. \s\***

* **Was wird gesucht?  
  Beliebig viele Leerzeichen (einschließlich gar keiner) nach dem Doppelpunkt :.**
* **Details:**
  + **\s: Repräsentiert ein Leerzeichen, Tab oder Zeilenumbruch.**
  + **\*: Gibt an, dass 0 oder mehr Leerzeichen erlaubt sind.**

**3. \'([^\']+)'**

* **Was wird gesucht?  
  Der Wert, der in einfachen Hochkommas (') steht und zu 'outcome\_diagnosis' gehört.**
* **Details:**
  + **\': Ein öffnendes Hochkomma, das den Start des Wertes markiert.**
  + **([^\']+):**
    - **[^\']: Jedes Zeichen, das kein Hochkomma ist (^ bedeutet "nicht").**
    - **+: Ein oder mehrere solcher Zeichen.**
    - **(...): Die runden Klammern definieren eine Gruppe, die den gesuchten Wert enthält.**
  + **\': Ein schließendes Hochkomma, das das Ende des Wertes markiert.**

### DownloadImage.py

Dieses Skript dient dazu, das vorher bereitgestellte Objekt zu verarbeiten und letztendlich in ein weiteres Skript zu geben, wobei der Austausch der Daten etwas komplexer als bei der vorherigen Datei ist.

Die Liste der benötigten Packages ist in diesem Skript so lange wie bei keinem anderem aber sie sind überschaubar und haben alle ihren Nutzen gefunden.

Packages:

* **Os 🡪** simple Befehle für Informationsbeschaffung zum OS
* **Sys** 🡪 Zugang zu bestimmen Objekten vom OS
* **subprocess** 🡪 Starten eines neuen Prozesses um den **Python GIL** zu

umgehen.

* **json 🡪** ermöglicht mir es JSON-strings zu parsen.
* **requests** 🡪 ermöglicht http requests
* **multiprocessing 🡪** ermöglicht uns shared\_memory zu erstellen
* **pickle** 🡪 besserer Json-serializer
* **cv2 – 4.10.0.84 🡪** mächtige Bildbearbeitungs library
* **numpy** 🡪 erleichtert Handhabung von Vectorn, Arrays usw.

Das Skript beginnt mit einem einfachen Check ob Argumente für das Skript vorhanden sind, falls dies nicht der Fall ist wird ein „AssertionError“ geworfen und es wird nicht mehr fortgefahren. Die Logik dieser Datei kümmert sich im Endeffekt um die Datei welche aus dem Internet heruntergeladen wird und dann gibt diese sie serialisiert weiter in das nächste Skript. **A computer screen shot of a program

Description automatically generated**

Funktionen:

* replace 🡪 falls noch Inkonsistenzen der Daten vorhanden sind
* loads 🡪 erstellt ein JSON-Objekt aus einem JSON-string
* get 🡪 macht eine POST-Anfrage auf den bereitgestellten Link
* frombuffer 🡪 kopier das Bild von binärer zu einem Numpy-Array
* imdecode 🡪 konvertiert Numpy-Array zu Image-Matrix für OpenCV
* Picture.Picture 🡪 erstellen das „TransferObjekt“ und serialiseren es
* SharedMemory 🡪 Erstellt einen Buffer für Transport zwischen Skripten
* shm.buf 🡪 Schreibt die Daten in den geteilten Speicher
* array\_shape 🡪 Speicher von Metadaten des geteilten Speichers

****

Wie bereits in den vorherigen Skripten gibt diese Zeile die wichtigen Daten zum Finden des Buffers weiter und startet somit das find\_spot.py Skript.

### find\_spot.py

Dieses Skript ist das Kernstück der Diplomarbeit, neben der Künstlichen Intelligenz. Da wir inzwischen bereits die Bilder und die Diagnose als Objekt in dieses Skript als Parameter übergeben bekommen haben, bleibt nur noch der Schritt zur Erkennung der Hautläsionen. Auch mit Packages wie OpenCV, welche schon seit langem ihren Weg in den Standard der Bildverarbeitung gefunden hat, hat sich diese Aufgabe schwerer als gedacht herausgestellt. Wichtige Abhängigkeiten sind die folgenden:

* cv2 – Wichtiges Paket zur Verarbeitung und Manipulation

von Bildern

* shared\_memory - Erstellt einen Buffer für Transport zwischen Skripten
* picke – Zum serialisieren des fertigen Objektes

Diesen Teil der des Codes werde ich nur noch erklären, jedoch kein Bildmaterial zur Verfügung stellen. Der Grundgedanke dieses Skripts liegt bei der automatisierten Erkennung von Hautläsionen, jedoch ist der Vorgang ziemlich kompliziert und derzeit noch mit Problemen versehen.

Durch die Inkonsistenz der Bilddaten und fehlender Norm habe ich mich nur auf Bilder spezifiziert, welche die Hautläsion zentriert und gut erkennbar haben. Das heißt im Klartext, dass der Algorithmus sich einen Wert für die Mitte des Bildes berechnet und in diesem vordefinierten Bereich nach der „**Region of Interest“** sucht. Bevor das Bild weiter verarbeitet werden kann wird es noch in **Graustufen**, mittels **cvtColor(.., cv2.COLOR\_BGR2GRAY)**, umgewandelt werden. Durch diesen Arbeitsschritt haben wir die **Farbtiefe** des Bildes von 24-Bit auf nur noch 8-Bit herunterskaliert. Damit sich der Algorithmus der OpenCV-Bibliothek zuverlässigere Antworten gibt, wurde auch noch eine weiter Effekt angewandt.

Damit Konturen und Farbunterschiede einfacher erkennbar sind ist es sinnvoll, die sogenannte Gaußsche Unschärfe anzuwenden, welcher am Ende weiter behandelt wird. Mit diesen Schritten ist es der Funktion von OpenCV möglich eine Kopie des Originalbildes mit nur noch 2 Farben zurückzugeben, das wird anschließend mittels der **findContours(…)** gefiltert und gibt den gewünschten Bereich zurück. Wie bereits erwähnt ist dieser Algorithmus noch nicht perfekt aber er ist im Stande, Bilder welche unter akzeptablen Umständen gemacht wurden auf die Hautläsion zu suchen. Da nun unser Objekt wie gewünscht gespeichert wurde, ist es Zeit dieses Bild nun in das nächste Skript zu geben und als Teil der Datenvorbereitung für das Trainieren der KI in die Datenbank zu speichern. Wegen der fehlenden Genauigkeit der Bilder der Datenbank, ist noch eine manuelle Bewertung von Nöten, was heißt, dass die Bilder derzeit in ein temporäres Verzeichnis gespeichert wird. Temporär, weil die verbleibenden Bilder, welche nicht gelöscht wurden, werden in das **Download**-**Skript** geladen und anschließend vom Betriebssystem gelöscht.

**Änderungen in neuen Versionen:**

Aufgrund der schwachen Leistung des älteren Ansatzes wurde dieses Skript überarbeitet und mithilfe von **FastAPI** mittels Requests zu einem Microservice umdesigned.

Die erste Änderung betrifft schon die Übergabeparameter der Funktion, in der alten Version mittels geteilten Speicher und in der neuen simpel per Bytearray. Die initiale Vorgehensweise hat sich nicht wirklich verändert, was heißt, dass das Bild noch immer mittels Filter und Unschärfefilter versehen wird. Der Unterschied zwischen den Versionen liegt bei der Auswahl der Hautläsion. Damals in einem Durchgang, nun wird mittels Iterationen wird der Bereich indem gesucht wird stetig vergrößert und wenn keine Veränderung der Größe vor liegt zurück gegeben. Eine Änderung, welche die Perfomance noch weiters verbessert wäre mithilfe von Änderung des Kontrasts des Bildes.

### OpenCV

OpenCV (Open Source Computer Vision Library) ist eine umfangreiche Bibliothek, die speziell für die Entwicklung von Computer-Vision-Anwendungen und maschinellem Lernen entwickelt wurde. Sie ist Open-Source und unterstützt eine Vielzahl von Programmiersprachen, darunter C++, Python und Java. OpenCV bietet eine Vielzahl von Funktionen und Werkzeugen, die es ermöglichen, Bild- und Videoverarbeitung effizient durchzuführen, um visuelle Informationen zu extrahieren, zu analysieren und zu verarbeiten. Die Bibliothek enthält über 2500 Algorithmen, die in zahlreichen Anwendungen in Bereichen wie maschinelles Sehen, Robotik, Automobilindustrie, Medizin und Sicherheit eingesetzt werden.

Hauptfunktionen und Einsatzmöglichkeiten von OpenCV:

**Bildverarbeitung:**

OpenCV ermöglicht grundlegende Bildoperationen wie das Laden, Speichern und Bearbeiten von Bildern, z. B. Helligkeit, Kontrast und Filter. Transformationen wie Skalierung, Rotation und Verzerrung. Erkennung von Kanten, Ecken und Konturen in Bildern.

**Objekterkennung und Tracking:**

OpenCV bietet Methoden zur Gesichtserkennung, Bewegungserkennung, Objekterkennung und Verfolgung in Echtzeit. Dies ist nützlich für Sicherheitsanwendungen oder bei der Entwicklung autonomer Systeme wie Drohnen oder selbstfahrende Autos.

**Maschinelles Lernen:**

OpenCV enthält auch Funktionen, die maschinelles Lernen und tiefes Lernen unterstützen. So können Modelle für die Klassifikation und Erkennung von Mustern trainiert werden. Die Bibliothek ist kompatibel mit anderen tiefen Lernbibliotheken wie TensorFlow und PyTorch.

**3D-Bildverarbeitung:**

OpenCV ermöglicht die Arbeit mit 3D-Daten, etwa durch Stereo- oder Tiefenkarten. Dies ist besonders relevant für Anwendungen in der Robotik oder Augmented Reality.

**Videoanalyse:**

OpenCV kann zur Analyse von Videostreams verwendet werden, um z. B. Bewegungen zu erkennen, Objekte zu verfolgen oder sogar die Gesichts- oder Körperhaltung zu überwachen.

**Technische Merkmale:**

Plattformübergreifend: OpenCV ist auf verschiedenen Plattformen wie Windows, Linux, macOS, Android und iOS verfügbar.

**Optimierung:**

Die Bibliothek ist hoch optimiert für Geschwindigkeit, z. B. durch die Verwendung von Multi-Core-Prozessoren und GPU-beschleunigten Operationen.

**Erweiterbarkeit:**

OpenCV unterstützt die Integration mit anderen Bibliotheken und Frameworks, sodass es an die spezifischen Anforderungen einer Anwendung angepasst werden kann.

OpenCV ist aufgrund seiner Vielseitigkeit, Leistungsfähigkeit und der aktiven Entwicklergemeinschaft eine der am weitesten verbreiteten und beliebtesten Bibliotheken für Bild- und Videoverarbeitung.

Fazit: OpenCV bietet ein äußerst umfangreiches Set an Werkzeugen und Funktionen, die für verschiedenste Anwendungen in der Computer Vision und dem maschinellen Lernen genutzt werden können. Es ist eine unverzichtbare Ressource für die Entwicklung moderner, visuell orientierter Technologien und hat sich als Standardbibliothek in der Branche etabliert.

### FastAPI

FastAPI ist ein modernes, schnelles (high-performance) Web-Framework für die Entwicklung von APIs mit Python. Es basiert auf Python 3.6+ und nutzt die neuesten Features der Sprache, wie Typannotation und asynchrone Programmierung. FastAPI wurde mit dem Ziel entwickelt, eine möglichst einfache und dennoch leistungsstarke Möglichkeit zur Erstellung von Web-APIs zu bieten, und hat sich besonders aufgrund seiner Geschwindigkeit, Benutzerfreundlichkeit und der hervorragenden Unterstützung für automatische API-Dokumentation einen Namen gemacht.

Hauptfunktionen und Merkmale von FastAPI:

**Geschwindigkeit:**

FastAPI ist eines der schnellsten Python-Frameworks zur Erstellung von APIs. Es verwendet moderne Features von Python und ist stark auf Leistung optimiert. Durch den Einsatz von asynchronen Endpunkten und die Verwendung von Starlette als Web-Framework für das Routing und das Event-Loop-Modell, erreicht FastAPI beeindruckende Geschwindigkeiten, die mit anderen etablierten Frameworks wie Node.js konkurrieren können.

**Automatische API-Dokumentation:**

FastAPI erzeugt automatisch eine benutzerfreundliche API-Dokumentation mit OpenAPI und Swagger UI, was die Entwicklung und Nutzung von APIs deutlich vereinfacht. Entwickler müssen keine separate Dokumentation schreiben, da die API-Dokumentation automatisch aus den Typanmerkungen des Codes generiert wird.

Ein weiteres nützliches Feature ist die interaktive Dokumentation, bei der API-Endpunkte direkt im Browser getestet werden können.

**Typisierung und Validierung:**

FastAPI verwendet Python-Typanmerkungen und Pydantic-Modelle zur Validierung von Daten. Dies führt zu einer klareren und sichereren API-Entwicklung, da Eingabe- und Ausgabedaten automatisch validiert werden. Wenn die Eingabedaten nicht den erwarteten Typen entsprechen, gibt FastAPI automatisch Fehler zurück, was die Fehlerbehandlung vereinfacht.

Durch diese Typisierung ist der Code leicht verständlich und hilft, Fehler frühzeitig zu erkennen.

**Asynchrone Programmierung:**

FastAPI unterstützt vollständig asynchrone Programmierung mit async und await, was die Performance für I/O-gebundene Anwendungen (wie Datenbankabfragen oder Webanfragen) erheblich verbessert.

Dies macht FastAPI besonders geeignet für den Aufbau von APIs, die viele gleichzeitige Anfragen bearbeiten müssen, wie bei Echtzeitkommunikationssystemen oder Microservices.

**Einfache Integration von Datenbanken und externen Systemen:**

FastAPI lässt sich problemlos mit Datenbanken (z. B. SQLAlchemy, Tortoise ORM) und externen APIs integrieren. Es bietet Unterstützung für gängige Datenbankoperationen, die mit asynchroner Programmierung optimiert werden können.

**Sicherheit:**

FastAPI enthält Features zur Handhabung von Sicherheitsmechanismen, wie OAuth2, JWT (JSON Web Tokens) und grundlegender Authentifizierung. Dies erleichtert die Implementierung sicherer APIs und die Verwaltung von Benutzerrechten und -rollen.

**Technische Merkmale:**

**Leistung:** FastAPI ist extrem schnell, da es auf Starlette aufbaut und modernste Optimierungen wie asynchrone Programmierung nutzt.

Einfachheit: Trotz seiner vielen fortschrittlichen Funktionen bleibt FastAPI einfach zu bedienen, und Entwickler profitieren von einer kurzen Lernkurve, dank der klaren und gut strukturierten API-Dokumentation und -Beispielen.

**Kompatibilität:** FastAPI arbeitet hervorragend mit gängigen Python-Paketen und -Bibliotheken zusammen, einschließlich Pydantic für Datenvalidierung und SQLAlchemy oder Tortoise ORM für die Datenbankintegration.

Automatische Validierung und Fehlerbehandlung: Die automatische Eingabedatenvalidierung und die detaillierte Fehlerbehandlung verbessern die Zuverlässigkeit und Wartbarkeit der Anwendungen.

Beispiele für Anwendungsbereiche:

**Fazit:**

FastAPI hat sich als eine der besten Wahlmöglichkeiten für die Entwicklung von modernen, performanten APIs in Python etabliert. Es bietet eine Vielzahl von Funktionen wie automatische Dokumentation, hohe Geschwindigkeit, asynchrone Programmierung und starke Typisierung, die zusammen eine hervorragende Entwicklererfahrung bieten. Durch die einfache Integration mit verschiedenen Datenbanken, die Unterstützung für Sicherheitsmechanismen und die Möglichkeit, mit asynchronen Prozessen zu arbeiten, ist FastAPI besonders gut für anspruchsvolle, skalierbare Web- und Microservices-Anwendungen geeignet. Entwickler können sich darauf verlassen, dass FastAPI nicht nur eine schnelle und effiziente API-Entwicklung ermöglicht, sondern auch ein robustes Framework zur Verfügung stellt, das die Entwicklung und Wartung von APIs vereinfacht.

* + 1. **Datenbank-Backend**
       1. **Datenmodell und Struktur**

Als Datenbank haben wir eine No-SQL Datenbank verwendet, da es für unser Vorhaben perfekt in unser Bild gepasst hat. Durch die Funktionsweise sind wir nicht an ein reguläres Schema gebunden, sondern können einfach mittels JSON-Objekte dynamisch eine persistente Speicherung der Daten erzielen. Das Backend meiner API verwendet jedoch doch ein Schema, welches ein Bild und die Diagnose in einem JSON-Dokument haben will. Mit dieser Konfiguration ist es uns möglich eine einheitliche persistente Speicherung der Daten zu erzielen, was uns später vieles vereinfachen wird. Eine Implementation von Usern ist nicht Teil unserer Muss-Ziele, jedoch wollen wir, wenn genug Zeit vorhanden ist ein solches System wie in unseren Soll-Zielen umsetzten und somit weitere Funktionalität zu unserer Applikation bringen.

**Änderung in neuen Versionen:**

Mithilfe der in **AdonisJS** implementieren Funktionalitäten ist es mir gelungen, ein Schema für Benutzer im Backend zu etablieren und mittels Hashing-Algorithmen zu verschlüsseln. Das Frontend sendet eine Anfrage an meinen Endpunkt mit den bereits verschlüsselten Daten an meinen bereitgestellten Endpunkt, anschließend durchläuft dieses Objekt eine Reihe von Verschlüsselungen im Backend. Diese Daten werden nun persistent im Backend gespeichert, was zu einem gesteigertem Benutzererlebnis führt, da diese nun ihre bereits eingescannten Bilder in einem Verlauf zur Verfügung gestellt bekommen.

* + - 1. **Technologien und Tools**

Um uns das bestmögliche Backend zu machen habe ich auf modernste Technologien wie das Typescript-Framework [**Adonisjs**](https://adonisjs.com) gesetzt. Um eine Verbindung zur Datenbank aufzubauen habe ich das Package Mongoose verwendet, welches mich nun bereits durch das 3 Projekt im schulischen Umfeld begleitet. Der Techstack von Adonisjs ist nicht gering, bietet jedoch für sehrviele Situationen wichtige Funktionalität, welche das Entwickeln um einiges einfacher machen. Mithilfe der gut formulierten und instandgehaltenen Dokumentation ist das Einleben in das Framework besonders leicht gemacht. Zum Einsatz sind diese Funktionalitäten bereits beim Verschlüsseln der Benutzerdaten und erhalten der etwas größeren Bilddaten gekommen.

* + - 1. **Datenbankoperationen**
      2. **Verschlüsselung der gespeicherten Daten**

Eine wichtige und meist unterschätze Funktionalität eines gut implementieren Backends, ist das Verschlüsseln der Daten mittels verschiedensten zur Verfügung stehenden Technologien. Aufgrund des breit gestreuten Techstacks von AdonisJS, ist das Verschlüsseln der Daten ein Kinderspiel und kann mittels einer simplen Funktion garantiert werden. Das hashen der Daten wurde mittels **SHA-256** umgesetzt welcher als derzeitiger Standard angesehen wird. Eine optimale Verschlüsselung würde auch noch einen gebrauch vom **Salting**, also eine bewusste Zufälligkeit zum Hashing selbst bringen, jedoch fehlt mir für diese Implementierung einfach die Zeit und das Wissen. Die Passwörter werden mehrfach Verschlüsselt und letztendlich auf die Datenbank persistent gespeichert. Dies verhindert aber das Einsehen des Passworts und schränkt den Benutzer in der Hinsicht ein, ihr Passwort im Klartext zu betrachen.

### Befüllen der Datenbank mit Testdaten

Aufgrund der beschränkten Leistung meiner Hardware, ist kein direktes **Multi-Threading** möglich sondern eine optimiertere Art von synchronen Programmierung. Eine erheblicher Leistungsanstieg könnte man durch das Auslagern der einzelnen Funktionen auf verschiedene Geräte erreichen, wie man es von einer Microserice Architektur gewohnt ist. Trotz der technischen Einschränkungen könnte man bereits die Software mittels **Docker-Container** auf verschiedenen Maschinen hosten und somit das Ergebnis, welches mich ganze 3 Stunden gebraucht hat, innerhalb von wahrscheinlich einer Stunde erzielen. Eine Menge von 500 Datensätzen serialisiert und persistent in der Datenbank lokal auf meinem Rechner zu speichern beansprucht 3 Stunden. In vorherigen Versionen wurde wegen mangelnder Leistung des Webscraping Skriptes der Datensatz nur auf ~250 Bilder beschränkt. Durch diesen beschränkten Datensatz war die künstliche Intelligenz nur im Stande eine durchschnittliche Genauigkeit beim Analysieren der Bilder von ~50% zu erreichen, was nicht gerade ein vielversprechendes Ergebnis ist. Aufgrund der Verdopplung der Daten zum Trainieren und Testen des Modells ist eine Erhöhung der Zuverlässig- und Genauigkeit zu erwarten.

## Datenbank-API

#### Test und Validierung

AdonisJS stellt bereits ein Framework zum Schreiben und ausführen von Tests bereit, was heißt, dass mittels CLI-Befehlen einfach Tests erstellen und ausführen kann. Eine wichtige Unterscheidung zwischen Tests gibt es jedoch, welche sich von den herkömmlichen unit-tests unterscheiden.

Unit-Tests prüfen wie gewohnt einzelne Komponenten und Funkionen des Codes, während Funktionale-Tests einen größeren Abschnitt der Applikation abdeckt und diese aus der Sicht des Benutzers ausgeführt werden. Um die tests laufen zu lassen muss man im vorhinein die Packages mithilfe von **npm i** runterladen und anschließend mit ***node ace test*** aktivieren, dies startet die Tests und gibt eine formatierte Ausgabe für den Benutzer in der CLI wieder.

# Projektabschluss

## Installation / Software deployment

# Literaturverzeichnis

# Abbildungsverzeichnis