**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:

Inhalt

# 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

### Datenquelle und Beschaffung

#### 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()

* + - 1. **Datenbereinigung und Vorverarbeitung**

Zum Webscrapen, damit wir die Bilddaten für unser Projekt verwenden können, habe ich mithilfe von verschiedenen Python Modulen gearbeitet. Der erste Schritt war die Links und die Diagnose mittels RegEx herauszufiltern und mittels Request an den Link des Bildes letztendlich das Bild selbst zu erhalten. Im nächsten Schritt war es vonnöten diese Bilder zuzuschneiden und für die KI vorzubereiten. Einfach gesagt habe ich mittels einer OpenCV-Funktion die Anomalien in einem Bild ausgeschnitten und zu Graustufen verändert. Das führte letztendlich zu inkonsistenten Bildern in Graustufen, welche jedoch aufgrund von fehlender Skalierung nicht für das Machine-Learning verwendbar sind. Eine weiter wichtige Information für diesen Abschnitt ist auch, dass ich für Performancezwecke auch die Scripte aufgeteilt habe und deshalb auf einen „shared Memory“, auch geteilten Speicher, zugreifen musste. Dieser geteilte Speicher ermöglichte es mir einfach große Datenmengen, wie die Bilddaten selbst, zwischen Scripten auszuteilen.

* + - 1. **Labeln der Daten**

Das Labeln der Daten habe ich mittels JSON-Objekts umgesetzt, was heißt, dass das es ein Attribut Picture gibt, welches logischerweise auch ein Bild als Numpyarray repräsentiert. Das zweite Attribut der JSON ist die Diagnose, welches die Diagnose des Arztes selbst ist. Wichtig hinzuzufügen ist auch, dass ich einen großen Teil meiner Zeit in das Zuschneiden der Bilder auf das nötigste gelegt habe.

* + 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 an die Datenbank senden. 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.

* + - 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 auf das Package Mongoose gesetzt, 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.

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

Mithilfe der Hashing-Funktionen SHA-256 und „salting“ werden Passwörter auf dem Backend gespeichert, was heißt, dass es nicht möglich ist Passwörter anzeigen zu lassen. Datentransfer zwischen Client und Server wird auch mittels eines gehashten Wertes ermöglicht.

* + - 1. **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.

* + 1. **Vorbereitung**
    2. **Aktionsphase**
    3. **Ende**

1. **Installation / Software deployment**
2. **Projektabschluss**

# 

# Literaturverzeichnis

# Abbildungsverzeichnis