**DIPLOMARBEIT**

DermaAI: Intelligente Hautanalyse

Gesamtprojekt

Entwicklung von KI-Modellen zur Detektion und Klassifikation pigmentierter Hautläsionen und deren 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: MSC

Michael Prader

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

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

Gerhard Gaube

Schuljahr 2024/25

Abgabevermerk:

Datum: TT.MM.JJJJ übernommen von:

(VORHER THEORIE DANN UMSETZUNG)

(Theorie in Projektplanung)

# Allgemeines & Projektübersicht

## Projektbeschreibung

## Projektteam und Schnittstellen

## Funktionale Anforderungen

## Use Cases

## Nicht Funktionale Anforderungen

## Projektplanung

### Recherche / Vorarbeit

#### Jonas Maier (Datenbank)

##### Recherche Backend & Datenbank

Im Rahmen dieser Diplomarbeit war es notwendig uns über die bestmöglichen Technologien, Techniken der Datenverarbeiten und Frameworks für das Machine-Learning zu informieren. Durch die Notwendigkeit von robusten und, falls von Nöten, auch skalierbaren Komponenten haben wir uns eine Zeit zum Recherchieren von zwei Wochen eingeteilt. Während dieser zwei Wochen habe ich die Möglichen Methoden abgewägt und habe mich letztendlich für folgende Frameworks und Methoden entschieden:

**Backend**

Beim Thema Backend haben wir bereits eine Vielzahl von Möglichkeiten in der Schule durchgearbeitet. Trotz der großen Auswahl an Sprachen und Möglichkeiten habe ich mich nicht für ein bereits gelerntes Framework entschieden, sondern kurzerhand für AdonisJS entschieden. Mein Gedanke bei dieser Entscheidung obliegt bei der Komplexität, geringen Overhead und großartigen Ausstattung des Frameworks.

**„*Create bespoke backend applications in record time***

***AdonisJS is a TypeScript-first web framework for building web apps and API servers. It comes with support for testing, modern tooling, an ecosystem of official packages, and more.*“** (AdonisJS Team, 2025)

Da die Community hinter Node und dem Paketmanager NPM eine Vielzahl an Mitwirkern hat ist die Wahrscheinlichkeit auf gute Pakete durchaus hoch. In der Kombination mit den bereits zur Verfügung gestellten Funktionalitäten stellt dieses Framework eine solide Basis für ein solides und auch skalierbares Backend.

**Datenbank**

Das Herzstück jedes Backends ist wahrscheinlich die Datenbank, um die Daten persistent zu hinterlegen, um zu einem späteren Zeitpunkt wieder mit zu Interagieren. Die Auswahl der Datenbank können wir grob in zwei große Klassen einteilen, einerseits die SQL-Datenbanken, welche Datenkonsistenz und komplexe JOINs ermöglichen, andererseits die No-SQL-Datenbanken. Diese sind für unser Vorhaben wie gemacht, indem die uns mit den schnellen Lese- und Schreibgeschwindigkeiten, als auch ein Schemaloses per Beispielsweise JSON-Daten zur Verfügung stellen kann. Wenn wir die Soll-Ziele in Betracht ziehen, wäre MongoDB ein guter Ansatz, da wir bereits des öfteren damit gearbeitet haben, wenn wir das nicht tun und rein die Bilder für das trainieren der KI brauchen, gibt es bessere und optimiertere Datenbanken. Amazon bietet eine Vielzahl von Diensten für die Speicherung von blobs, oder auch **binary large object**, welche leider Großteils entgeltlich zur Verfügung gestellt werden.

**Datenquelle der Trainingsdaten ect.**

„***KI ist nur so gut wie die Daten, die sie antreiben***“ (Kurian, 2023)

Ein Zitat wie es im Bezug auf künstliche Intelligenz nicht richtiger sein könnte, deshalb ist die Quelle und Verarbeitung der Daten einer der wichtigsten Schritte um eine funktionale KI zu erschaffen.

Die Auswahl medizinischer Archive, die unserem Anforderungsprofil entsprechen, ist nicht leicht. Doch schon nach einer einzigen Google-Suche findet man das ISIC-Archiv – eine Sammlung medizinischer Daten nach einer weltweiten Norm, perfekt für unser Projekt. Eine öffentliche Dokumentation zum Erreichen der Endpunkte und genauste Beschreibung, welche Daten man für seine Anfragen erhält.

##### Webscraping & Datenverarbeitung

Im Rahmen dieser Diplomarbeit werden medizinische Bilder aus einer externen Medizinischen Datenbank extrahiert, verarbeitet und anschließend in einer eigenen Datenbank gespeichert. Um einen zuverlässigen Algorithmus zu erschaffen wurden die Arbeitsschritte in viele kleinere Teile aufgeteilt, welche wie folgt aussehen:

1. **Definition der Datenquelle**

* Die Wahl der Datenquelle stellt sich als besonders wichtig heraus, da es notwendig ist, Bilder mit ähnlichen Verhältnissen für eine einfachere und zuverlässigere Datenverarbeitung zu haben.
* Der Zugriff auf die Schnittstelle der medizinischen Datenbank muss gut dokumentiert sein, damit Abfragen mittels APIs und Webscraping effektiv sein können.

1. **Datenextraktion durch Webscraping**

Nachdem die benötigte Information der Datenquelle gesammelt wurden, ist nun die Methode um die gefragten Daten zu sammeln gefragt, dazu gibt es folgende Methoden.

* API-basierte Datenextraktion:

Extern zur Verfügung gestellte Daten können mittels API-Anfragen abgefragt werden, was man dazu benötigt, ist der API-Endpunkt und die benötigte Formatierung der Anfrage. Authentifizierungsmechanismen wie Tokens, API-Keys, OAuth usw. müssen in der Dokumentation aufgewiesen werden.

* Webscraping-Techniken:

Falls keine API zur Verfügung steht, muss man auf sogenannte Webscraping-Methoden zurückgreifen. Hierbei werden HTML-Strukturen und Elemente analysiert um relevante Bild-URLs zu extrahieren und schlussendlich automatisierte Download-Mechanismen implementieren zu können.

1. **Bildverarbeitung und Optimierung**

Da die extrahierten Bilder nun heruntergeladen und zwischen gespeichert wurden, kann die Datenverarbeitung anfangen. Auch hier gibt es wieder verschiedenste Methoden und Ansätze dieses Problem zu lösen.

* Formatkonvertierung

Falls die Bilder in unterschiedlichen Formaten vorliegen, werden sie in ein einheitliches Format überführt, um spätere Verarbeitungen zu erleichtern.

* Bildoptimierung

Techniken wie Rauschreduktion, Kontrastanpassung oder Skalierung werden angewendet, um die Bildqualität zu verbessern. Dabei werden Methoden aus der Bildverarbeitung genutzt, z.B. OpenCV oder Pillow.

* Automatisierte Qualitätskontrollen

Um sicherzustellen, dass die extrahierten Bilder vollständig und fehlerfrei sind, werden automatische Prüfmechanismen implementiert. Dazu gehören Bildformat-Validierungen, Auflösungsprüfungen und Metadaten-Checks.

1. **Automatische Erkennung von Hautläsionen**

* Mittels bereitgestellter Funktionen die Hautläsion ausfündig machen, eventuell weitere Optimierung des Bildes notwendig. ROI (Region of Interest) definieren, damit nicht auf dem gesamten Bild gesucht werden muss.

1. **Speicherung auf eigener Datenbank**

* Datenbankschema

Wie werden die Bilder und eventuell weiter Informationen gespeichert?  
Möglichkeiten wie BLOBs, „Bytearray“ stehen zur Verfügung.

* Auswahl Datenbank

Differenzierung von Datenbanken zwischen Rational und No-SQL-Datenbanken, das eine robust und zuverlässig, während das andere eine gute Leistung aufzeigt.

1. **Skalierung und Automatisierung des Workflows**

Effizient ist ein wichtiges Stichwort, wenn es um die Datenverarbeitung geht. Um die Effizienz zu steigern gibt es folgende Methoden:

* **Batch-Verarbeitung:** Statt Bilder einzeln zu extrahieren und zu verarbeiten, werden sie in Gruppen verarbeitet, um die Effizienz zu steigern.
* **Parallele Verarbeitung:** Nutzung von paralleler Bildbearbeitung durch Multi-Threading oder verteilte Systeme, um die Rechenzeit zu reduzieren.
* **Automatisierte Workflows:** Implementierung von Pipelines mit Tools wie Apache Airflow oder Prefect, um den gesamten Ablauf (Extraktion, Verarbeitung, Speicherung) zu automatisieren.
* **Monitoring & Fehlerbehandlung:**

Logging-Mechanismen zur Nachverfolgung des Verarbeitungsstatus.

Alerts und Benachrichtigungssysteme zur Erkennung und Behebung von Fehlern.

**Fazit**

Durch die Kombination aus Webscraping, Bildverarbeitung und effizienter Speicherung wird eine zuverlässige und skalierbare Infrastruktur für die Verwaltung medizinischer Bilddaten geschaffen. Dies ermöglicht eine einfache Integration in weiterführende Analyse- und Machine-Learning-Systeme sowie die langfristige Nutzung der Daten unter Einhaltung geltender Datenschutzrichtlinien.

##### Datenbank & Persistente Speicherung

Ein wesentlicher Bestandteil moderner Softwareanwendungen ist die strukturierte und dauerhafte Speicherung von Daten. Datenbanken spielen hierbei eine zentrale Rolle, da sie eine effiziente, sichere und konsistente Verwaltung von Informationen ermöglichen. Die Wahl der geeigneten Datenbanktechnologie hängt von verschiedenen Faktoren ab, darunter die Art der zu speichernden Daten, die Zugriffsmuster und die Anforderungen an Skalierbarkeit und Performance.

**Relationale vs. NoSQL-Datenbanken**

Grundsätzlich lassen sich Datenbanksysteme in zwei große Kategorien einteilen: relationale und nicht-relationale (NoSQL) Datenbanken.

* Relationale Datenbanken basieren auf einem tabellenbasierten Modell und verwenden die relationale Algebra sowie die Structured Query Language (SQL) zur Abfrage und Manipulation der Daten. Sie sind besonders geeignet für Anwendungen, bei denen Konsistenz und Integrität der Daten eine hohe Priorität haben. Bekannte relationale Datenbanksysteme sind beispielsweise MySQL, PostgreSQL und Microsoft SQL Server.
* NoSQL-Datenbanken hingegen bieten mehr Flexibilität in der Datenmodellierung, da sie nicht auf einem starren tabellenbasierten Schema beruhen. Sie werden häufig in modernen Webanwendungen, Big-Data-Anwendungen und verteilten Systemen eingesetzt. Je nach Typ können NoSQL-Datenbanken als dokumentenorientierte (MongoDB), key-value-basierte (Redis), spaltenorientierte (Apache Cassandra) oder graphenbasierte (Neo4j) Systeme organisiert sein.

**Persistenzmechanismen**

Die Persistenz von Daten bezeichnet die Fähigkeit eines Systems, Informationen dauerhaft zu speichern und nach einem Neustart wieder bereitzustellen. Hierfür gibt es verschiedene Mechanismen:

* Dateisystembasierte Speicherung: In manchen Anwendungen reicht es aus, Daten in einfachen Dateien oder strukturierten Formaten wie JSON, XML oder CSV abzulegen. Diese Methode wird häufig für Konfigurationsdateien oder temporäre Daten genutzt.
* Datenbanken: Für komplexe Anwendungen sind relationale oder NoSQL-Datenbanken die bevorzugte Lösung, da sie effiziente Abfragen, Transaktionssicherheit und Mehrbenutzerzugriff ermöglichen.
* Caching-Technologien: Um die Performance zu steigern, können häufig genutzte Daten temporär in In-Memory-Datenbanken wie Redis oder Memcached zwischengespeichert werden.

**ACID- und BASE-Prinzipien**

Beim Umgang mit persistenten Daten spielen Konsistenz- und Verfügbarkeitsaspekte eine wichtige Rolle. Relationale Datenbanken orientieren sich am ACID-Prinzip, das für Atomicity (Atomarität), Consistency (Konsistenz), Isolation und Durability (Dauerhaftigkeit) steht. Diese Prinzipien stellen sicher, dass Transaktionen zuverlässig ausgeführt werden.

NoSQL-Datenbanken hingegen folgen oft dem BASE-Modell (Basically Available, Soft State, Eventually Consistent). Dabei wird eine höhere Verfügbarkeit auf Kosten der sofortigen Konsistenz in Kauf genommen, was für verteilte Systeme und Cloud-Anwendungen vorteilhaft ist.

**Auswahl der passenden Speicherlösung**

Die Wahl der richtigen Speicherlösung hängt stark von den Anforderungen der Anwendung ab:

* Hohe Transaktionssicherheit erforderlich? → Relationale Datenbanken
* Große, flexible Datenmengen mit schwacher Struktur? → NoSQL-Datenbanken
* Schnelle Lesezugriffe mit geringem Speicherbedarf? → Caching-Systeme
* Einfache Konfigurations- oder Log-Daten? → Datei- oder JSON/XML-Speicherung

In der Praxis setzen viele Anwendungen auf hybride Ansätze, um die Vorteile verschiedener Technologien zu kombinieren. So kann beispielsweise eine relationale Datenbank für die Kerninformationen genutzt werden, während eine NoSQL-Datenbank oder ein Caching-System für schnelle Lesezugriffe oder unstrukturierte Daten verwendet wird.

Durch eine gezielte Auswahl und Kombination von Datenbank- und Persistenztechnologien kann die Effizienz, Skalierbarkeit und Zuverlässigkeit eines Systems erheblich verbessert werden.

##### Mögliche Backend Frameworks

Die Entwicklung moderner Webanwendungen erfordert eine effiziente, sichere und skalierbare Backend-Architektur. Backend-Frameworks bieten hierfür eine strukturierte Grundlage, indem sie wesentliche Funktionen wie Routing, Datenbankanbindung, Authentifizierung und API-Management bereitstellen. Die Wahl des richtigen Frameworks hängt von verschiedenen Faktoren ab, darunter die verwendete Programmiersprache, die Skalierbarkeitsanforderungen und die Präferenz des Entwicklungsteams.

**Kriterien für die Auswahl eines Backend-Frameworks**

Ein gutes Backend-Framework sollte mehrere wichtige Anforderungen erfüllen:

* **Leistungsfähigkeit & Skalierbarkeit**: Das Framework sollte in der Lage sein, große Mengen an Anfragen effizient zu verarbeiten und sich leicht auf verteilte Systeme erweitern lassen.
* **Flexibilität & Erweiterbarkeit**: Es sollte einfach sein, zusätzliche Module oder externe Bibliotheken zu integrieren.
* **Sicherheit**: Funktionen wie Authentifizierung, Autorisierung und Schutz vor gängigen Bedrohungen (z. B. SQL-Injections, CSRF, XSS) sollten unterstützt werden.
* **Community & Support**: Eine aktive Community, gute Dokumentation und regelmäßige Updates sind entscheidend für die langfristige Wartbarkeit und Weiterentwicklung.

**Beliebte Backend-Frameworks nach Programmiersprachen**

**JavaScript/Node.js**

* **Express.js**: Minimalistisches, schnelles und flexibles Framework, das häufig für REST-APIs und serverseitige Anwendungen genutzt wird.
* **NestJS**: Ein TypeScript-basiertes, modular aufgebautes Framework, das auf Prinzipien von Angular basiert und sich für große Enterprise-Anwendungen eignet.
* **Fastify**: Ein sehr performantes Framework für Node.js, das sich besonders für Anwendungen mit hohen Anfragenraten eignet.

**Python**

* **Django**: Ein leistungsstarkes Full-Stack-Framework mit integriertem ORM (Object-Relational Mapping), Authentifizierung und vielen Sicherheitsfunktionen.
* **Flask**: Ein leichtgewichtiges Framework für kleinere Anwendungen und Microservices, das viel Freiheit in der Architektur bietet.
* **FastAPI**: Ein modernes, asynchrones Framework für APIs, das auf Pydantic und OpenAPI basiert und besonders für leistungsfähige Anwendungen geeignet ist.

**Java**

* **Spring Boot**: Ein sehr verbreitetes, produktionsreifes Framework für Unternehmensanwendungen mit umfangreicher Unterstützung für Microservices und Cloud-Deployments.
* **Quarkus**: Ein neues, auf Performance optimiertes Framework, das besonders für Cloud-native Anwendungen geeignet ist.
* **Micronaut**: Entwickelt für Microservices und Serverless-Umgebungen mit schnellem Startverhalten und geringem Speicherverbrauch.

**PHP**

* **Laravel**: Ein beliebtes Full-Stack-Framework mit vielen eingebauten Funktionen wie Eloquent ORM, Blade Templating und Authentifizierungssystemen.
* **Symfony**: Modulares, flexibles Framework, das sich gut für komplexe Anwendungen eignet.
* **CodeIgniter**: Ein leichtgewichtiges Framework, das für schnelle und einfache Webanwendungen gedacht ist.

**Ruby**

* **Ruby on Rails**: Ein sehr produktives Framework, das auf Konvention über Konfiguration setzt und eine starke Community sowie zahlreiche vorgefertigte Module bietet.

**Go (Golang)**

* **Gin**: Ein performantes, minimalistisches Framework für den schnellen Aufbau von Webanwendungen und APIs.
* **Fiber**: Inspiriert von Express.js, aber optimiert für die Performance von Go.

**C# (.NET)**

* **ASP.NET Core**: Ein leistungsfähiges, plattformübergreifendes Framework für Web-APIs und Anwendungen mit hoher Skalierbarkeit.

**Fazit**

Die Wahl des richtigen Backend-Frameworks hängt stark von den individuellen Anforderungen eines Projekts ab. Während einige Frameworks wie Django oder Laravel eine vollständige Lösung mit vielen eingebauten Funktionen bieten, setzen andere wie Flask oder Express.js auf Minimalismus und Flexibilität. Moderne Anwendungen verwenden häufig auch mehrere Technologien in Kombination, z. B. NestJS für eine API und Redis für das Caching.

#### Daniel Jessner (KI & Python)

#### Jonas Bogensberger (Frontend)

## 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 *Verfasst von Jonas Maier*

### 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.

Maier Jonas

### 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.

Maier Jonas

* + 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.

Maier Jonas

* + - 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.

Maier Jonas

* + - 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.

### 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 Skript 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

Maier Jonas

#### Fetch\_from\_isic\_archive(…)

Wir starten mit dem entrypoint des mainscripts:

1. if \_\_name\_\_ == '\_\_main\_\_':  
2. start\_time = time.perf\_counter()

3. fetch\_from\_isic\_archive(connection\_string)  
4. end\_time = time.perf\_counter()  
5. elapsed\_time = end\_time - start\_time

6 print(f"Elapsed time: {elapsed\_time: .1f} seconds\n")

Bevor die Funktion **fetch\_from\_isic\_archive(…)** aufgerufen wird halten wir mithilfe vom **time.perf\_counter()** die Zeit zum Zeitpunkt der Exekution des Skriptes 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.

1. fetched\_data = fetch\_html(link)  
2. data = to\_json(fetched\_data)['results']  
3. next\_link = to\_json(fetched\_data)['next']  
4. content = convert\_data(data)

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.

1. for i in content:  
2. try:

3. diagnosis = find\_regex(outcome\_diagnosis\_regex\_ENG\_ISIC, str(i)).group(1)  
4. picture = find\_regex(pic\_regex\_ENG\_ISIC, str(i)).group(1)  
5.  
6. if diagnosis is None or picture is None:  
7. print("Iteration skipped!")  
8. continue  
9.  
10. if len(diagnosis) < 0 or len(picture) < 0:  
11. print("Iteration skipped!")  
12. continue  
13.  
14. print(f'Diagnosis: {diagnosis}')  
15. print(f'Pictures: {picture}')  
16. download\_images(Picture.Picture(picture, diagnosis))  
17.  
18. except AttributeError as e:  
20. print(f'AttributeError: {e}', )  
21.  
22. except Exception as e:  
23. print(f'Critical Error: {e}')  
34.  
35. if len(next\_link) > 0:  
36. fetch\_from\_isic\_archive(next\_link)

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.

Maier Jonas

#### download\_images(…)

1. result = subprocess.run([sys.executable, 'DownloadImage.py', str(obj.to\_dict())],  
2. cwd=os.getcwd(), capture\_output=True)

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.

Maier Jonas

#### 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.**

Maier Jonas

### DownloadImage.py

Um die Bilder persistent auf der Datenbank speichern zu können, ist es notwendig, dass die Bilder heruntergeladen & anschließend verarbeitet werden. Gerade weil dieses Skript von dem Main-Teil aufgerufen wird, ist es notwendig, dass eine Überprüfung der gegebenen Parameter erfolgt.

1. # Check if link has been provided

2. assert len(sys.argv) > 1, "Not enough arguments given!"

Der erste Schritt ist, den vom Haupt-Skript übergebenen Datensatz zu verarbeiten. Das Normalisieren und Laden der Objekte ist ein essenzieller und notwendiger Schritt, um den Speicherort des Bildes zu extrahieren. Mithilfe der extrahierten URI können wir das Bild nun herunterladen und zum Weiterverarbeiten benutzten.

1. data = str(sys.argv[1])

2.     data = data.replace("'", '"')

3.     item = json.loads(data)

4.     response = requests.get(item['picture'])

Der nächste Schritt beinhaltet das „casten“ der, im vorherigen Schritt, erhaltenen Daten in eine Liste vom Datentyp Int. Aufgrund der etwas aufwendigeren Verarbeitung der Daten im Byteformat, ist es eine durchaus einfache Möglichkeit sich das Leben einfacher zu gestalten. Mithilfe des im nächsten Kapitel erklärten Microservices, können wir einfach die gesammelten Bilddaten an diesen Endpunkt mittels einer POST-Anfrage machen. Die Antwort dieses Endpunktes ist ein Bild, welches auf das Hautmelanom, mit etwas Abstand zum Rand, ausgeschnitten wurde.

1. if response.status\_code == 200:

2.         image\_data = list(response.content)  # Convert image bytes to a list of integers

3.         new\_response = requests.post("http://localhost:6969/", json={"image": image\_data})

Die folgenden Zeilen Code dienen der Verifizierung des Datentyps und des Inhalts. Mithilfe der „data“ Variable erstellen wir unseren JSON-string, welcher anschließend, mittels ein bisschen Trickserei, in eine Datei umgewandelt wird. Diese Datei wird letztendlich an den bereits definierten Endpunkt der Datenbank-API gesendet und dort persistent gespeichert.

1. img = new\_response.json()

2.

3.         if img == 'error!' or np.array(img, dtype=np.uint8).size == 0:

4.             print("Error")

5.             exit(-1)

6.

7.         data = {

8.             "picture": np.array(img, dtype=np.uint8).tolist(),

9.             "diagnosis": item['diagnosis']

10.         }

11.

12.         file = {"file": ("data.json", BytesIO(json.dumps(data).encode("utf-8")),

"application/json")}

13.         response = requests.post(url, files=file)

### find\_spot.py

Dieses Skript ist mit der künstlichen Intelligenz der wichtigste Bestandteil dieser Arbeit, da ohne gescheite Trainingsdaten die KI keine gewünschten Erfolge erzielen kann. Wie bereits erwähnt ist dieses Skript durch eine **Fast-API** zu einem Endpunkt gemacht worden, was eine einfachen aber zuverlässigen Zugang bietet. Bewerkstelligt wird das genannte Ergebnis mit den folgenden Zeilen Code:

1. class ImageRequest(BaseModel):

2.     image: List[int]  # Accept image as a list of bytes (integers)

3.

4. @app.post("/")

5. async def find\_spot(request: ImageRequest):

6.     image\_bytes = bytes(request.image)  # Convert list of integers back to bytes

7.     img = real\_crop\_image\_microservice(image\_bytes)  # Process image

8.

9.     if img is None:

10.         return {"error": "Processing failed"}

11.

12.     return {"file": img.tolist()}

Nun kommen wir zum eigentlichen Skript welches die Datenverarbeiten ermöglicht. Die verwendeten Pakete in diesem Skript waren einerseits OpenCV2, welches eine einfache Bildverarbeiten ermöglicht und dieses Projekt ermöglicht. Die zweite Abhängigkeit lautet numpy, welche uns das Arbeiten mit Zahlen und Arrays um vieles erleichtert.

1. import cv2

2. import numpy as np

In den ersten Zeilen dieser Funktion wird das Bild auf den benötigten Datentyp umgewandelt, folglich werden wichtige Werte für den Algorithmus berechnet.

1.  # Decode image from byte array

2.     bimage = np.frombuffer(image, dtype=np.uint8)

3.     image = cv2.imdecode(bimage, cv2.IMREAD\_COLOR)

4.

5.     # Get image dimensions

6.     height, width = image.shape[:2]

7.     center\_x, center\_y = width // 2, height // 2

Wegen des iterativen Ansatzes dieses Algorithmus ist es wichtig, dass eine obere Grenze für Iterationen festgesetzt wird, sodass das Programm nicht die Gefahr läuft in eine Endlosschleife zu kommen. Zusätzlich gibt es einen festen Wert, welcher die minimale Größe des gesuchten Bereich definiert.

 1.   # Initialize previous detected spot and pixel counts

2.     prev\_detected\_spot = np.zeros((0, 0, 0), dtype=np.uint8)

3.     latest\_big = np.zeros((0, 0, 0), dtype=np.uint8)

4.     iterations = 0

5.     max\_iterations = 300

6.     min\_area = 10  # Minimum contour area to consider

Nach dem Initialisieren der Variablen, ist es möglich den Iterativen Ansatz zu verfolgen. Ziel dieser Zeilen Code ist es eine zentrale Region zu definieren, welcher mit stetig wachsenden Iterationen größer wird und folglich in eine weitere Funktion gegeben wird. Dazwischen gibt es wieder eine Verifikation um die Dimensionen des angegeben Bildes zu garantieren.

1. # Loop for iterative processing of the image

2.     while iterations < max\_iterations:

3.         try:

4.             roi\_size = 25 + (iterations \* 100)  # Increment ROI size each iteration

5.             roi\_x1, roi\_y1 = center\_x - roi\_size, center\_y - roi\_size

6.             roi\_x2, roi\_y2 = center\_x + roi\_size, center\_y + roi\_size

7.             roi = image[roi\_y1:roi\_y2, roi\_x1:roi\_x2]

8.

9.             if roi.shape[0] == 0 or roi.shape[1] == 0:

10.                 print("ROI size exceeds image dimensions, breaking loop.")

11.                 break

12. # Process region of interest (ROI)

13.             center\_region = process\_roi(roi)

Das Ziel der „process\_roi“ Funktion liegt darin das übergebene Bild mittels Graustufenfilter, Trübheitsfilter und Cannyfilter das Bild auf den nächsten Schritt vorzubereiten. Man könnte sagen, dass die gesamte Funktion darauf hin arbeitet das Bild für die Suche von Pixeln mit bestimmten RGB-Wert zu finden – Die gefundenen Pixel sind somit auch der Rückgabeparameter dieser

Funktion.

1. def process\_roi(roi):

2.     """

3.     Process the ROI (Region of Interest) by converting to grayscale,

4.     applying Gaussian Blur, and thresholding.

5.     """

6.     try:

7.         # TODO: INCREASE / DECREASE CONTRAST IF BAD

8.         gray = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY)

9.         # qualized = cv2.equalizeHist(gray)

10.         blurred = cv2.GaussianBlur(gray, (5, 5), 0)

11.         edges = cv2.Canny(blurred, 50, 150)

12.         \_, thresholded = cv2.threshold(edges, 127, 255, cv2.THRESH\_BINARY\_INV)

13.         return thresholded

14.     except:

15.         print("error")

Nachdem wir eine Liste mit unregelmäßigen Pixeln erhalten haben können wir zum nächsten Schritt gehen. In dieser gefundenen Liste suchen wir nun alle Konturen, also in unserem Fall die Hautläsionen. Mithilfe der gefundenen Konturen suchen wir die größte Kontur und speichern diese mithilfe des Originalbildes in Graustufen in eine Zwischenvariable.

1. # Find contours in the thresholded image

2. contours, \_ = cv2.findContours(center\_region, cv2.RETR\_EXTERNAL,

cv2.CHAIN\_APPROX\_SIMPLE)

3.

4. print(f"CONTOURS AMMOUNT={len(contours)}")

5.

6. for idx, contour in enumerate(contours):

7. if cv2.contourArea(contour) > min\_area: # Filter small contours

8.

9. x, y, w, h = cv2.boundingRect(contour)

10. x += roi\_x1

11. y += roi\_y1

12. detected\_spot = cv2.cvtColor(image[y:y + h, x:x + w],

cv2.COLOR\_BGR2GRAY)

14. # Find the biggest spot

15. if prev\_detected\_spot.size < detected\_spot.size:

16. prev\_detected\_spot = detected\_spot

Nach jeder Iteration vergleichen wir das Aktuelle und das bereits gespeicherte Melanom auf ihre Größe. Wenn die Größe des gespeicherten Melanoms unverändert oder null ist, geben wir es als Rückgabeparameter der Funktion zurück. Wenn das gespeicherte Melanom kleiner ist, als das neue erhöhen wir den Iterationscounter um eins und starten diese Schleife erneut.

1.             if latest\_big.size == prev\_detected\_spot.size and prev\_detected\_spot.size != 0:

2.                 print(f"Returning last detected spot after {iterations} iterations.")

3.                 return latest\_big

4.             elif latest\_big.size < prev\_detected\_spot.size:

5.                 latest\_big = prev\_detected\_spot

6.             iterations += 1

## Datenbank-API

### AdonisJS – Ein allrounder Framework?

AdonisJS ist ein modernes und umfassendes Webframework für Node.js, das sich an die Prinzipien von Laravel orientiert und eine strukturierte sowie produktive Entwicklungsumgebung bietet. Es wurde speziell entwickelt, um Entwicklern eine robuste, sichere und performante Grundlage für Web- und API-Anwendungen zu liefern. AdonisJS setzt auf eine klare Architektur, eine integrierte ORM-Lösung und zahlreiche Features, die die Entwicklung vereinfachen.

#### MVC-Struktur

Die MVC-Architektur von AdonisJS erleichtert die Trennung von Geschäftslogik, Benutzeroberfläche und Datenbankinteraktionen. Mit der Implementierung von verschiedensten Technologien bietet dieses Framework eine zuverlässige Basis um folgende Komponenten klar und deutlich zu trennen:

* **Model:** Definiert die Datenstruktur und -logik mit dem Lucid ORM.
* **View:** Bietet eine optionale Template-Engine für server- oder auch clientseitiges Laden der Seiten.
* **Controller:** Verarbeitet Anfragen und orchestriert die Kommunikation zwischen Model und View.

#### Lucid ORM

Lucid ist das integrierte Object-Relational Mapping (ORM) von AdonisJS. Es ermöglicht die einfache Verwaltung von Datenbanken mit einer intuitiven API und unterstützt gängige relationale Datenbanken wie MySQL, PostgreSQL und SQLite. Lucid bietet Features wie:

* Migrationssystem zur Verwaltung von Datenbankschemata
* Beziehungen zwischen Modellen (One-to-One, One-to-Many, Many-to-Many)
* Abfrage-Builder mit einer klaren Syntax

Aufgrund des ausgewählten Tech Stacks müssen wir leider auf die in diesem Kapitel aufgezählten Funktionalitäten verzichten und selbst mit einem Vergleichbarem System für No-SQL Datenbanken aufkommen.

#### ****Routing-System****

AdonisJS besitzt ein leistungsfähiges Routing-System, das HTTP-Anfragen effizient verarbeitet. Es ermöglicht die Definition von benutzerfreundlichen Routen, Middleware-Handling und Parameter-Validierung. Mithilfe der bereitgestellten Funktionalitäten ermöglicht AdonisJS eine einfache und zuverlässige Art und Weise die Routen und das Backend nach Lust und Laune zu konfigurieren. Beispielsweise stellt sich das hinzufügen von Präfixen an eine Gruppe von Routen als ganz einfach heraus.

1. // Add prefix to PredictionController routes

2. router.group(() => {

3.   router.post('/savePrediction', [PredictionController, 'savePrediction']);

4.   router.post('/loadPrediction', [PredictionController, 'loadPrediction'])

5. }).prefix('/prediction')

#### Middleware und Sicherheitsfeatures

Middleware in AdonisJS ermöglichen die Manipulation von Anfragen und Antworten. Sie können zur Authentifizierung, Protokollierung oder Ratenbegrenzung eingesetzt werden. Zudem bietet AdonisJS eine integrierte CSRF-Schutz-Funktion und eine sichere Authentifizierungslogik mit JWT- oder Session-basierten Tokens. Der Fokus liegt bei der Middleware, da somit ein einfacher Loggingalgorithmus erstellt werden kann und mithilfe von **Pino** Log-Dateien für bestimmte Zeiträume gespeichert werden können. AdonisJS bieter aber auch verschiedenste Middlewares um die Richtigkeit der Daten verifizieren zu können.

1. export default class ForceJsonResponseMiddleware {

2.   async handle({ request }: HttpContext, next: NextFn) {

3.     const headers = request.headers()

4.     headers.accept = 'application/json'

5.   return next()

6.   }

7. }

#### Ace – Das AdonisJS Befehlszeilentool

Ace ist das integrierte Kommandozeilen-Tool von AdonisJS, das Entwicklern hilft, wiederkehrende Aufgaben zu automatisieren. Mit Ace können Controller, Modelle, Middleware und Migrationsdateien erstellt sowie Datenbankoperationen verwaltet werden. Es spielt eine zentrale Rolle bei der Entwicklung, da es die Produktivität steigert und manuelle Prozesse reduziert.

#### ****Wichtige Funktionen von Ace****

##### ****Entwicklungsserver & Projektverwaltung****

Ace ermöglicht es, den Entwicklungsserver mit automatischem Reloading zu starten, neue Projekte zu initialisieren und Umgebungsvariablen zu verwalten.

##### ****Generierung von Controllern, Modellen & Middleware****

Anstatt Dateien manuell zu erstellen, kann Ace diese automatisch generieren und mit Grundstrukturen versehen. Das spart Zeit und sorgt für Konsistenz im Code.

##### ****Datenbankverwaltung & Migrationen****

Ace hilft bei der Erstellung und Verwaltung von Datenbankmigrationen, Seedern und Verbindungen. Dadurch können Schemaänderungen organisiert und Testdaten einfach eingefügt werden.

##### ****Authentifizierung & Benutzerverwaltung****

Das Tool kann Authentifizierungsmechanismen mit wenigen Befehlen einrichten und so eine schnelle Implementierung von Login- und Benutzerverwaltungssystemen ermöglichen.

##### ****Eigene Befehle definieren****

Ace erlaubt es, eigene CLI-Befehle zu erstellen, um projektbezogene Aufgaben zu automatisieren, beispielsweise Datenimporte oder spezielle Wartungsroutinen.

#### Fazit

AdonisJS ist ein leistungsfähiges Framework für Entwickler, die eine strukturierte und produktive Umgebung für Node.js-Anwendungen suchen. Mit seiner MVC-Architektur, dem Lucid ORM und den integrierten Sicherheitsfeatures ist es eine ausgezeichnete Wahl für die Entwicklung skalierbarer Web- und API-Projekte. Darüber hinaus dominiert AdonisJS in vielen Punkten über andere bereits bestehende Frameworks, wie NestJS, ExpressJS und so weiter. Dazu kommt auch noch, dass der Tech Stack wie auf unser Projekt abgestimmt ist.

### Datenmodelle

Mongoose ist eine leistungsstarke ODM (Object Document Mapper)-Bibliothek für MongoDB und Node.js. Sie ermöglicht die Definition von Schemata für MongoDB-Dokumente und bietet eine strukturierte Art der Interaktion mit der Datenbank. Durch Mongoose können komplexe Datenstrukturen mit einfachen, objektorientierten Konzepten verwaltet werden. Diese Dokumentation behandelt die wesentlichen Konzepte hinter Mongoose-Modellen, Schemata und dem verwendeten mongoose-paginate-v2 Plugin, um eine tiefere Einsicht in die Architektur zu bieten.

#### Modelle und Schemata in Mongoose

#### **Schemata in Mongoose**

Ein **Schema** in Mongoose definiert die Struktur von Dokumenten in einer MongoDB-Kollektion. Es beschreibt die Eigenschaften und deren Typen, ob sie erforderlich sind, wie sie validiert werden und welche zusätzlichen Optionen zur Verfügung stehen. Ein Schema dient als eine Art Vorlage für die zu speichernden Daten und sorgt für eine konsistente Datenstruktur.

#### **Definition eines Schemas**

In der bereitgestellten Codebasis ist userSchema das definierte Schema für die userData-Kollektion:

1. const userSchema = new Schema<IUserData>({

2. email: {

3. type: Schema.Types.String,

4. required: true

5. },

6. password: {

7. type: Schema.Types.String,

8. required: true

9. },

10. mfa: {

11. type: Schema.Types.Boolean,

12. required: false

13. },

14. isAdmin: {

15. type: Schema.Types.Boolean,

16. required: false

17. }

18. }, { collection: "userData" });

19.

#### Warum Schemata verwenden?

* **Datenvalidierung:** Stellt sicher, dass alle gespeicherten Daten einem bestimmten Format entsprechen, wodurch Fehler und inkonsistente Daten vermieden werden.
* **Datenintegrität:** Verhindert inkonsistente Datenstrukturen innerhalb einer Kollektion und stellt sicher, dass alle Dokumente die gleichen Attribute aufweisen.
* **Erweiterbarkeit:** Ein Schema kann leicht erweitert oder geändert werden, ohne die gesamte Anwendung zu beeinträchtigen. Neue Felder können einfach hinzugefügt werden, ohne dass bestehende Daten verloren gehen.
* **Middleware-Unterstützung:** Erlaubt das Einfügen von Logik vor oder nach Datenbankoperationen, wodurch z. B. Hashing von Passwörtern oder das Setzen von Standardwerten automatisiert werden kann.

#### ****Modelle in Mongoose****

Ein **Modell** ist eine durch Mongoose erstellte Schnittstelle zur Datenbank und stellt Funktionen für CRUD-Operationen bereit (Create, Read, Update, Delete). Es wird aus einem Schema abgeleitet und bietet Methoden zur Interaktion mit der Datenbank.

#### **Definition eines Modells**

In der Codebasis wird das userDataModel mit mongoose.model erstellt:

1. export const userDataModel = mongoose.model<IUserData, mongoose.PaginateModel<IUserData>>

('userData', userSchema)

**Warum Modelle verwenden?**

* **Ermöglichen CRUD-Operationen:** Mit Modellen können wir Datensätze einfach erstellen, lesen, aktualisieren und löschen.
* **Bietet Methoden für erweiterte Abfragen:** Mongoose-Modelle enthalten viele nützliche Methoden wie find(), findById(), updateOne(), die den Zugriff auf Daten erleichtern.
* **Interagiert direkt mit der Datenbank:** Das Modell sorgt für eine klare Trennung zwischen Logik und Datenverwaltung und bietet eine optimierte Schnittstelle zur Datenbank.
* **Erweiterbarkeit:** Modelle können um benutzerdefinierte Methoden ergänzt werden, die spezielle Geschäftslogik implementieren.

#### Verwendung von mongoose-paginate-v2

In Anwendungen mit großen Datenmengen ist es wichtig, effiziente Paginierungsmechanismen zu implementieren, um die Performance zu verbessern und unnötige Last auf der Datenbank zu vermeiden. Hier kommt das Plugin mongoose-paginate-v2 ins Spiel.

**Einbindung des Plugins**

**Das Plugin wird im Schema wie folgt registriert:**

1. userSchema.plugin(paginate);

**Warum mongoose-paginate-v2 verwenden?**

* **Einfache Implementierung:** Paginierung kann mit wenigen Zeilen Code integriert werden.
* **Optimierte Abfragen:** Durch die Nutzung von limit() und skip() werden unnötige Last und Speichernutzung in der Datenbank reduziert.
* **Verbesserte Nutzererfahrung:** Ideal für APIs, die große Datenmengen verwalten müssen, indem nur die benötigten Datenabschnitte abgerufen werden.

**Beispielabfrage für eine paginierte Abfrage:**

1. public findPictures = async (ctx: HttpContext) => {

2. await checkState();

3. let queryParams = ctx.request.qs();

4.

5. const options = {

6. page: Number(queryParams.page) || 1,

7. limit: Number(queryParams.limit) || 10,

8. collation: {

9. locale: 'en',

10. },

11. };

12.

13. try {

14. return ctx.response.status(200).json(await pictureModel.paginate({}, options));

15. } catch (error) {

16. console.error(error);

17. return ctx.response.status(500).json({ message: 'An error occurred' });

18. }

19. }

### API-Controller

#### Routing & Routen in AdonisJS

AdonisJS ist ein moderne und leistungsstarkes Node.js-Framework, das ein intuitives Routing-System bietet. Routen sind das Herzstück jeder Webanwendung, da sie die Anfragen von Nutzern an die entsprechenden Controller und Handler weiterleiten.

##### Definition von Routen

Routen in AdonisJS werden in der Datei ***start/routes.ts*** definiert. Die folgenden Zeilen Programmcode unterteilen die Verfügbaren Endpunkte in drei einzelne Gruppen und teilt die Logik somit in einfach verständliche Teile.

Durch das hinzufügen eines Präfixes können wir eine weitere Unterteilung machen, welche in zukünftigen Prozessen das Interagieren einfacher machen.

1. router.group(() => {

2.   router.post('/savePrediction', [PredictionController, 'savePrediction']);

3.   router.post('/loadPrediction', [PredictionController, 'loadPrediction'])

4. }).prefix('/prediction')

##### Arten von Routen

AdonisJS unterstütz, wie andere moderne Frameworks auch, die folgenden http-Methoden:

* Route.get('/path', 'Controller.method') – Holt Daten ab
* Route.post('/path', 'Controller.method') – Erstellt neue Daten
* Route.put('/path', 'Controller.method') – Aktualisiert bestehende Daten
* Route.delete('/path', 'Controller.method') – Löscht Daten

##### Middleware

Eine Middleware kann in der Verbindung mit Authentifizierung auch Sicherheit in ein System bringen. In unserem Fall jedoch, benutzen wir diese Middleware, um einen simplen Logging-Algorithmus zu erstellen, der die folgenden Aufgaben hat:

* Logging der IP-Adressen und aufgerufenen Routen
* Anzahl der Aufrufe speichern

1. async handle(ctx: HttpContext, next: NextFn) {

2.     const ip = ctx.request.ip();

3.     const route = ctx.request.url();

4.     const key = `${ip} -> ${route}`

5.

6.     if(logStorage[key]){

7.       logStorage[key].count++;

8.       logger.info(`${key} [${logStorage[key].count}]`)

9.     }else{

10.       logStorage[key] = {count: 1};

11.       logger.info(`${key} [1]`)

12.     }

13.

14.     const output = await next()

15.     return output

16.   }

#### Loggen der Daten

Das Mitaufzeichnen des einkommenden Netzwerkverkehrs ist zwar kein notwendiger Schritt, jedoch ist er für ein System wie dieses mehr als nur empfohlen. Durch die große Anzahl an **npm-Paket**, welchen den Fokus auf das Loggen und Erstellen von Log-Dateien gibt, ist die Auswahl nicht einfach gefallen. AdonisJS gibt die Möglichkeit, einfach mithilfe von **Pino** Log-Dateien zu erstellen, welche sich nach bestimmten Perioden löschen oder wechseln.

# Projektabschluss

## Installation / Software deployment

# Literaturverzeichnis

# Abbildungsverzeichnis