FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Fachhochschule Aachen, Campus Jülich

Fachbereich 09 - Medizintechnik und Technomathematik
Studiengang Angewandte Mathematik und Informatik

Seminararbeit

Implementierung einer Schnittstelle zwischen Python und Node.Js zur bidirektionalen Übergabe von großen Datenmengen

Autor:
Sasha Knarren, 3570908

Betreuer:
Prof. Dr. Alexander Voß
Hendrik Eisbein

Aachen, 22. Dezember 2024

Zusammenfassung

Mit der zunehmenden Bedeutung des Machine Learnings, steigt auch die Anforderung daran, solche Funktionalitäten ohne technische Kenntnisse anzusprechen und wichtige Ergebnisse auslesen zu können. So sollen auch für das Forschungsprojekt PrEvelOp zwei Anwendungen entwickelt werden, um im Zusammenspiel die Wirtschaftlichkeit von Unternehmen zu steigern. Zum einen soll eine Python-Bibliothek geschrieben werden, die für das Projekt wichtige Funktionen des Machine Learnings enthält. Zusätzlich soll eine Web-Anwendung implementiert werden, welche diese Funktionen nutzt und aus den Daten visuelle Ergebnisse generiert. Die Aufgabe dieser Arbeit ist es, die Kommunikation zwischen diesen beiden Anwendungen zu ermöglichen. Dazu müssen große Datenmengen über das Internet übertragen und in beiden Programmiersprachen übersetzt werden können.

In dieser Arbeit werden die Vor- und Nachteile verschiedener Technologien zum Entwickeln einer Schnittstelle diskutiert, ein effizienter struktureller Aufbau eines Backends konzipiert und darauf aufbauend eine nutzerfreundliche Integration in die Web-Anwendung dargestellt. Als Technologien werden HTTP zur Kommunikation über das Internet, JSON zur Serialisierung von Daten und FastAPI als Framework für das Backend vorgestellt. Das Backend wird durch strenge Typenprüfung, Trennung der Verantwortlichkeiten und ein Strategy Pattern implementiert. Die Integration in die Web-Anwendung erfolgt über parallele asynchrone Abfragen, die an geschickten Stellen platziert sind und bei Bedarf hinter Ladeanimationen versteckt werden können.

Diese Schnittstelle ermöglicht es dieser und anderen Web-Anwendungen, die Ergebnisse der entwickelten Machine Learning Algorithmen visuell darzustellen. Dadurch kann eine breitere Masse an potentiellen Nutzern abgedeckt werden und durch die Kombination von Expertenwissen, die Wirtschaftlichkeit in der Produktion effizient gesteigert werden.

Inhaltsverzeichnis

1 Motivation

Datascience und Machine Learning sind rasant wachsende Forschungsthemen, die in immer mehr Bereichen Anwendung finden. So können auch viele Algorithmen und Modelle aus diesen Bereichen verwendet werden, um beratend in der Industrie zu dienen. Dafür werden in der Produktion gesammelte Daten analysiert und daraus, anhand unterschiedlicher Modelle, Entscheidungen empfohlen. Da dabei allerdings Programmierkenntnisse erforderlich sind, ist es schwierig für Fachfremde solche Analysen durchzuführen. Durch eine grafische Nutzeroberfläche kann dieses Problem gelöst werden. Nutzer können dadurch leichter mit zuvor entwickelten Algorithmen interagieren und die Ergebnisse besser interpretieren. Ein Problem dabei liegt darin, dass zur Entwicklung der verschiedenen Machine Learning Algorithmen meistens andere Programmiersprachen verwendet werden, als zur Entwicklung von Nutzeroberflächen. Aus diesem Grund muss eine Schnittstelle konzeptioniert und umgesetzt werden, die eine schnelle und verlustlose Kommunikation zwischen den Algorithmen und der Oberfläche ermöglicht.

1.1 Das Forschungsprojekt PrEvelOp

Den Rahmen dieser Arbeit bildet das Forschungsprojekt PrEvelOp. Es wird in Kooperation mit dem FIR - Forschungsinstitut für Rationalisierung und des WZL - Werkzeugmaschinenlabor der RWTH Aachen umgesetzt. Ziel des Projektes ist es, mithilfe von Machine Learning Methoden die Wettbewerbsfähigkeit von kleinen und mittleren Unternehmen zu steigern. Dafür sollen Ähnlichkeiten der Werkstücke erkannt werden. Die Werkstückbeschreibungsdaten bestehen dabei aus CAD-Beschreibungen eines Bauteils und Arbeitsplandaten für die Herstellung dieses Bauteils. Anhand der erkannten Ähnlichkeiten können Maßnahmen empfohlen werden, die Varianz in der Produktion verringern können, ohne dass dafür Produktvarianten abgeschafft werden müssen.

1.1.1 Aufgaben des FIR e. V. an der RWTH Aachen

Der Fokusbereich des FIR e. V. in diesem Projekt umfasst die Aufgabe der Entwicklung der Machine Learning Logik. Diese wird in Python unter Verwendung verschiedener vorhandener Pakete geschrieben und soll als Open-Source-Bibliothek zur Verfügung stehen. Zudem werden die Daten der Industriepartner verwaltet, um diese zu untersuchen und Tests unter echten Bedingungen durchzuführen.

1.1.2 Aufgaben der Abteilung Produktionsmanagement am Lehrstuhl Produktionssystematik

Die Aufgabe der Abteilung Produktionsmanagement in diesem Projekt ist das Entwickeln einer Web-Anwendung. Die Applikation soll als Demo für die entwickelte Bibliothek dienen und für Test- und Übungszwecke bereitgestellt werden. Da die eingegeben Daten sensible Unternehmensinformationen wie beispielsweise Produktkonfigurationen enthalten können und einer Geheimhaltungsvereinbarung unterliegen, ist ein Speichern der Daten auf einer Datenbank nicht möglich. Als Aushilfe dient die Verwendung von Electron. Mithilfe von Electron können Node.Js Apps auf dem Desktop ausgeführt werden, was das Ablegen der Daten auf dem Rechner ermöglicht. Es wird entsprechend der Anforderung eine Node.Js Electron Applikation implementiert, welche die Daten lokal speichert und Zugriff auf die entwickelten Machine Learning Funktionen besitzt.

1.2 Aufgabe und Aufbau der Arbeit

1.2.1 Aufgabe der Arbeit

Diese Arbeit beschäftigt sich mit dem erwähnten Zugriff des Frontends auf die Machine Learning Funktionen in dem Python Code. Damit diese Prozesse miteinander kommunizieren können, muss eine Schnittstelle entwickelt werden. Da das Frontend lokal auf den Rechnern der Partner und Kunden laufen, der Python Code allerdings auch global zur Verfügung stehen soll, ist eine Kommunikation über das Internet vorgesehen. So kann dann, zusammen mit den Python-Funktionen, die Schnittstelle als Backend für die Node.Js App dienen. Die extern, also durch den FIR e. V., zu schreibende Bibliothek muss dafür nahtlos in das Backend eingefügt und alle vorhandenen Funktionen abgedeckt werden. Zusätzlich muss im Frontend die Kommunikation mit der Schnittstelle nutzerfreundlich integriert werden. Da zum Teil sehr große Datenmengen verschickt werden müssen, ist es zudem wichtig, dass die Kommunikation effizient und schnell verläuft.

Zusammenfassend ist die Aufgabe der Arbeit, die Konzeption und Implementierung einer zuverlässigen und schnellen Verbindung zwischen einem Electron Node.Js Frontend und einem Python Backend für die Berechnung und das Clustering von Prozess- und Produktdaten.

In 1.1 wird ein Zusammenspiel der verwendeten Technologien und zu schreibenden Anwendungen veranschaulicht.

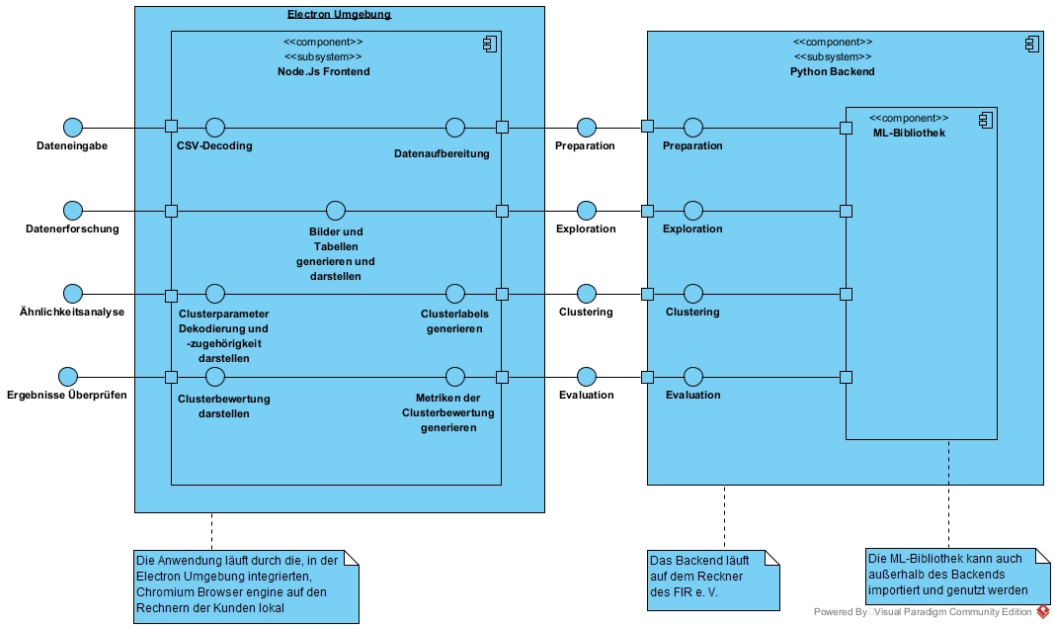


Abbildung 1.1: Komponentendiagramm der zu entwickelnden Anwendungen

1.2.2 Aufbau der Arbeit

Die Durchführung dieser Aufgabe erfolgt anhand von drei Abschnitten, die zu einem theoretischen und einem praktischen Teil zusammengefasst werden können. Zuerst müssen im theoretischen Teil Entscheidungen darüber getroffen werden, welche Technologien für die Implementierung der Schnittstelle verwendet werden. Anschließend wird in dem ersten praktischen Teil die Schnittstelle auf der Seite des Backends konzeptioniert und umgesetzt. Schließlich kann in dem zweiten praktischen Teil auf Basis des implementierten Backends die benutzerfreundliche Kommunikation des Frontends mit dem Backend entworfen und implementiert werden.

Verwendung der Technologien

Die verwendeten Technologien bilden die Grundlagen jeder Software. Somit ist es wichtig, im theoretischen Teil, die verschiedenen Optionen in Betracht zu ziehen und die jeweiligen Vor- und Nachteile abzuwägen. In diesem Abschnitt werden Entscheidungen für die Verwendung des Kommunikationsprotokolls, des Datenaustauschformats und des Frameworks, mithilfe dessen das Backend geschrieben wird, besprochen.

Konzeption und Implementierung des Backends

In der ersten Hälfte des praktischen Teils wird beschrieben, wie die einzelnen Komponenten aufgebaut sind und wie diese miteinander in Beziehung stehen. So wird der Aufbau, der dort verwendeten und geschriebenen Klassen erklärt und die Entscheidung dafür dargelegt. Es

wird festgelegt, welche Form die Nachrichten zwischen dem Frontend und dem Backend haben müssen und unter welchen Umständen diese ausgetauscht werden. Außerdem wird in diesem Teil die Wahl der Struktur für die verwendeten Dateien beschrieben und begründet.

Integration der Schnittstelle in das Frontend

Die zweite Hälfte des praktischen Teils ist die Integration und der Ablauf der Kommunikation mit der Schnittstelle in dem Frontend. Inhaltlich geht es hierbei darum, wann und unter welchen Bedingungen die Kommunikation stattfindet. Dabei wird besonders auf die User-Experience geachtet, indem durch geschickte Zeitpunkte der Anfragen, Wartezeiten vermieden werden und, wenn diese nicht vermeidbar sind, entsprechende Maßnahmen zur Information der Nutzer getroffen werden.

2 Grundlagen

Im Rahmen der Konzeption und Implementierung einer API werden hier viele verschiedene Technologien in Betracht gezogen. Grund dafür sind die verschiedenen Vor- und Nachteile, aber auch die verschiedenen Zusammenhänge zwischen den einzelnen Technologien. Beispielsweise schränkt die Wahl der Programmiersprache und des Frameworks im Frontend den Einsatz bestimmter Technologien ein. So können bestimmte Elemente besonders effizient, andere hingegen besonders schwierig zu implementieren sein. Weiterhin sind Entscheidung, wie die Wahl des Frameworks im Backend, von anderen, wie der Wahl des Kommunikationsprotokolls, abhängig. Somit ist es wichtig, zuerst Klarheit und Verständnis für die in dieser Arbeit betrachteten Technologien und deren mögliche Verbindungen zu schaffen.

2.1 Frontend

Da das Frontend durch mehrere Entwickelnde in der Abteilung Produktionsmanagement umgesetzt wird, wurden bereits die Programmiersprache sowie das Framework des Frontends festgelegt. Allerdings ist die Berücksichtigung dieser Technologien wichtig für mehrere darauf aufbauende Entscheidungen.

2.1.1 Node.Js

Node.Js stellt eine effiziente und vielseitig einsetzbare Laufzeitumgebung für JavaScript-Applikationen dar. Als Open Source Projekt können viele Funktionen als Pakete ausgelagert werden. Diese Pakete können dann über den sogenannten Node Package Manager, kurz npm, installiert werden. Dadurch kann Node.Js von Single-Page-Applikationen bis hin zu REST-APIs eingesetzt werden. Zudem bietet es eine hohe Unterstützung für viele Protokolle wie HTTP und Websockets sowie eine schnelle Serialisierung der Daten. Mithilfe von Events, Callbacks und sogenannten Promises können mehrere Funktionen asynchron ausgeführt und erst nach Abschluss der Operation die Ergebnisse abgefragt werden. So können Anfragen über das Internet gestellt werden, ohne dabei die Funktionsfähigkeit von davon unabhängigem Code zu beeinträchtigen. Für die Integration dieser API sind solche Promises wichtig, da sie die User-Experience bei langen Verarbeitungszeiten durch anspruchsvolle Methoden und große Datenmengen erhöhen.

2.1.2 Electron

Eine Anforderung an das Frontend besteht darin, lokal auf den Rechnern der Kunden verwendet zu werden, um die Produktdaten dort lokal zu speichern, anstatt in einer externen Datenbank.

Da Node.Js in der Regel serverseitig ausgeführt wird, erschwert dies den Einsatz. Dagegen kann die Verwendung von Electron Abhilfe schaffen. Electron integriert die Chromium-Browser-Engine sowie die Node.Js-Laufzeitumgebung zusammen mit dem geschriebenen und kompilierten HTML-, CSS- und JavaScript-Code in eine ausführbare .exe Datei. Dies ermöglicht die Ausführung der Anwendung, ohne dass diese zuvor von einem externen Server bereitgestellt werden muss. [Ele24]

So können Daten unkritisch im Frontend verwaltet und zwischengespeichert werden. Die Kommunikation mit dem Backend muss jedoch über das Internet erfolgen, da Frontend und Backend nicht auf dem selben Gerät laufen.

2.2 Schnittstelle

Die Schnittstelle steuert die Kommunikation zwischen Frontend und Backend. Es müssen Daten zwischen zwei unterschiedlichen Programmiersprachen ausgetauscht werden, ohne dass dabei Informationen verloren gehen. Außerdem muss die Kommunikation einheitlich erfolgen. Beide Elemente müssen diese verstehen können und alle Anforderungen müssen durch sie abgedeckt sein. Somit ist es wichtig, die verschiedenen möglichen Technologien zu verstehen und daraus die geeignete zu finden.

2.2.1 Kommunikationsprotokolle

Mithilfe von Kommunikationsprotokollen kann geregelt über Netzwerke kommuniziert werden. Dabei haben sich textbasierte Protokolle als vielseitig einsetzbar und sicher bewiesen. Der interne Aufbau und Ablauf einer Applikation kann durch diese Wahl, sowohl im Frontend als auch im Backend, stark beeinflusst werden. In der Kommunikation über das Internet sind vor allem die zwei Protokolle HTTP und Websockets weit verbreitet. Beide bieten eine Reihe von unterschiedlichen Funktionen mit denen zwei Systeme kommunizieren können.

HTTP

Das Hypertext Transfer Protocol ist die meistgenutzte Art und Weise, im Internet zu kommunizieren. Es ist nativ in Browsern interpretierbar und baut auf zuverlässigen Technologien wie TCP auf, um Datenverluste zu vermeiden. Es besitzt eine feste Struktur und verschiedene Methoden, um eine gute Kommunikation sicherzustellen.

Für jede HTTP-Anfrage gibt es eine entsprechende Antwort. Die Anfrage besteht dabei aus einer URL, einer Methode, einer Reihe von Headern und einem von der Methode abhängigen Body. Die URL kann dabei in drei Teile zerlegt werden. Der erste Teil stellt die Adresse der jeweiligen Webseite oder API dar und endet mit einer sogenannten Top-Level-Domain, wie .de, .com oder .org. Danach kann, anfangend mit einem Slash (`/`), ein Pfad zu den gewünschten Funktionen beginnen. Dieser Pfad kann dabei aus mehreren Ebenen bestehen, die wiederum durch ein Slash getrennt sind. Zuletzt kann eine sogenannte Query an die URL angehängt werden. Diese beginnt mit einem Fragezeichen (`?`) und kann mehrere Schlüssel-Wert-Paare als *Schlüssel*`=`*Wert* darstellen, die jeweils durch ein Und-Zeichen (`&`) getrennt

werden. HTTP-Anfragen können mit neun Methoden gestellt werden. Die Methoden OPTIONS, TRACE, HEAD und CONNECT sind meist intern und für die Wahl der Schnittstelle nicht relevant. Die Methoden GET, PUT, PATCH, POST und DELETE bilden, vor allem in der REST-Architektur, wichtige Funktionen ab, da mit ihnen über dieselbe URL unterschiedliche Verhaltensweisen abgebildet werden können. Die beiden wichtigsten Methoden für reines HTTP sind GET und POST. GET ist die Operation, die automatisch durch den Browser gestellt wird, wenn eine URL aufgerufen wird. Da Anfragen mit GET keinen Body besitzen, müssen alle benötigten Informationen in der Query enthalten sein. Daher eignet sich GET weniger für die Übertragung von großen Datenmengen und mehr für die Abfrage spezifischer Funktionen. POST-Anfragen hingegen verfügen sowohl über den Body als auch über die Query und können daher zur Übertragung großer Datenmengen verwendet werden. Schließlich enthält jede HTTP-Anfrage eine Reihe von Headern als Schlüssel-Wert-Paare, die Informationen über den Zustand der Verbindung und über die erwartete Antwortnachricht enthalten. Eine HTTP-Antwort beinhaltet einen Status, eine Liste von Headern und einen Body. Der Status enthält Informationen über den Erfolg oder Misserfolg der Anfrage und wird als dreistelliger Code angegeben. Die Header der Antwort ähneln denen der Anfrage und sind größtenteils Antworten auf diese. Der Body der Antwort beinhaltet die angeforderten Informationen, wie zum Beispiel ein serialisiertes Objekt oder auch HTML, um Webseiten visuell abzubilden.

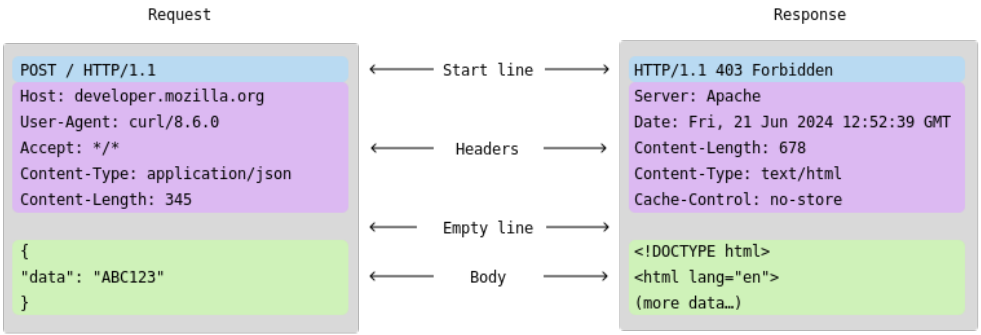


Abbildung 2.1: Aufbau HTTP Anfrage und Antwort
Quelle: https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages stand 07.12.2024

Durch die verschiedenen Elemente, wie Methoden, URL, Body und Status können fast alle benötigten Funktionen und die Fehlerbehandlung einer API durch HTTP realisiert werden.

REST

Representational State Transfer ist kein direktes Kommunikationsprotokoll, sondern eher eine Architektur, die auf HTTP aufbaut, um die Qualität und Wartbarkeit einer Web-Applikation zu erhöhen. Dafür stellt REST eine Reihe von Anforderungen an das Programm und legt auch die Verwendung von bestimmten HTTP-Methoden fest. Die erste Anforderung besagt, dass Ressourcen innerhalb einer REST-Applikation auf eine festgelegte Art und Weise von überall gleich aufgerufen werden können. Eine REST-Applikation soll zudem keinen Zustand besitzen. Anfragen sollen also unabhängig von den vorausgegangenen Anfragen bearbeitet

werden können. Eine Trennung von Client und Server sorgt für einen problemlosen Austausch von Technologien auf beiden Seiten, sofern dieser benötigt wird. Weiterhin ist eine darauf aufbauende Anforderung, dass innerhalb des Servers einzelne Elemente unabhängig voneinander sind, so dass diese auch bei Gebrauch ausgetauscht werden können. Antworten einer API sollen zudem zwischengespeichert werden können, um Wartezeiten zu verringern. Die letzte Anforderung beinhaltet die Möglichkeit auch ausführbaren Code an das Frontend senden zu können. Diese ist aber optional und wird aus Sicherheitsgründen selten implementiert. Durch REST werden die HTTP-Methoden GET, PUT, POST und DELETE an bestimmte Funktionen gebunden. So soll mit GET ein Lesebefehl, mit PUT oder PATCH ein Befehl für das Ändern einer Ressource, mit POST einer für das Erstellen einer Ressource und mit DELETE ein Löschbefehl ausgeführt werden. [Lat23]

So kann durch die REST-Architektur eine stabile und gut zu implementierende API für verschiedene Funktionen geschrieben werden.

Websockets

Während bei HTTP der Datenaustausch ausschließlich durch die Anfragen der Clients gesteuert wird, bietet das Websocket-Protokoll dem Server die Möglichkeit, bei Bedarf Daten direkt an die Clients zu senden. So können Daten in Echtzeit zwischen mehreren Clients ausgetauscht werden, indem der Server diese einfach über die Websockets weiterleitet. Außerdem können Berechnungen im Hintergrund durchgeführt und die Ergebnisse zu einem späteren Zeitpunkt an die Clients zurückgesendet werden. Dazu wird bei der Verbindung von Client und Server eine HTTP-Anfrage gestellt, um die Verbindung auf Websocket zu erweitern. Sind beide Websocket-fähig, kann dann das Protokoll gewechselt werden. Ein Nachteil von Websockets ist jedoch, dass es nur eine Schnittstelle zwischen Server und Client gibt und somit keine klare Trennung von Funktionen und Daten besteht.

2.2.2 Datenaustauschformate

Ein nicht unwesentlicher Teil einer Schnittstelle ist die Wahl des Übertragungsformats. Es gibt viele verschiedene Formate, mit jeweils auch unterschiedlichen Vor- und Nachteilen. Beispiele dafür sind YAML, Protocol Buffers, CBOR, JSON oder XML. Die Wahl der Programmiersprache hat einen großen Einfluss auf die Wahl des Übertragungsformats. So muss die Sprache vor allem kompatibel sein und hat einen positiven oder negativen Einfluss auf die Serialisierung. Durch die benannte Anforderung des Frontends, in JavaScript und Node.Js geschrieben zu sein, fällt die Vorauswahl vor allem auf JSON und XML, da diese bereits häufig in der Kommunikation von Node.Js verwendet werden.

XML

Die Extensible Markup Language, abgekürzt XML, dient weit verbreitet als Möglichkeit, Daten zu beschreiben und damit in serialisierter Form auszutauschen. Sie wurde 1998 durch das World Wide Web Consortium veröffentlicht.

XML ist textbasiert und besteht aus Tags, Attributen und einem Text zwischen den Tags. Tags sind dadurch gekennzeichnet, dass sie mit einem `<` Zeichen beginnen und mit einem `>` Zeichen enden. Ein Beispiel dafür wäre `<Wert>`. Für jedes Start-Tag gibt es ein entsprechendes End-Tag. Dieses hat die selbe Bezeichnung und unterscheidet sich dadurch, dass es mit `</` beginnt. Ein Beispiel wäre `<Wert>1</Wert>`. Innerhalb eines Tags können Attribute definiert werden. Diese bestehen aus dem Titel des Attributs, gefolgt von einem Gleichzeichen und dem Wert des Attributs in doppelten Anführungszeichen. Ein Beispiel wäre `<Wert Attribut="1">`. Mehrere Tags können wie in einer Baumstruktur ineinander verschachtelt werden, um komplexe Daten darzustellen. Dabei dürfen Tags nur auf der gleichen Ebene wieder geschlossen werden. `<Tag1><Tag2></Tag1></Tag2>` wäre nicht erlaubt. Wenn ein XML-Dokument diesen Regeln entspricht, wird es als wohlgeformt bezeichnet. Zusätzlich kann ein XML-Dokument durch ein sogenanntes XML-Schema (XSD) eingeschränkt werden. Dort kann zum Beispiel festgelegt werden, welchen Inhalt bestimmte Tags haben sollten.

Durch eine hohe Standardisierung und diese Regeln können Datenstrukturen sehr genau in XML-Dokumente übersetzt und auch mit geringer Fehlerwahrscheinlichkeit versendet werden. XML war daher auch lange Zeit das meistgenutzte Datenaustauschformat im Web und gab AJAX (Asynchronous JavaScript And XML), einer der ersten Möglichkeiten in Single Page Applications Daten von Servern abzufragen, den Namen. Nachteile von XML währenddessen sind der hohe Overhead und damit längere Wartezeiten bei Anfragen mit XML-generierten Dokumenten. [Gru18] [LCCY12]

JSON

JavaScript Object Notation, kurz JSON, ist ein weit verbreitetes textbasiertes Format zur Serialisierung von Daten. Es wurde von Douglas Crockford aus den Literalen und der Objektsyntax von JavaScript abgeleitet. Es stellt auf natürliche Art und Weise skalare Variablen, lineare Listen und Schlüssel-Wert-Paare dar und deckt damit die meistgenutzten Programmier-Strukturen ab. [Sev12]

In JSON können vier primitive und zwei strukturelle Datentypen abgebildet werden. Die vier primitiven Datentypen können als String, Nummer, Boolean und null unterschieden werden. Strings werden durch die doppelten Anführungszeichen (`"`) vor und nach dem Wert gekennzeichnet. Nummern bestehen aus einzelnen Ziffern, die bei Bedarf durch einen Punkt (`.`) als Dezimalzahlen dargestellt werden können. Booleans besitzen die beiden Literale `true` und `false`. Das Literal `null` kennzeichnet einen nicht gesetzten Wert. Im Vergleich zu den primitiven, sind das Array und das Objekt strukturelle Datentypen. Das Objekt wird, wie in 2.2 visualisiert, durch geschweifte Klammern am Anfang und am Ende (`{}`) und dazwischen eine durch Komma (`,`) getrennte Menge von Schlüssel-Wert-Paaren gebildet. Der Schlüssel muss immer ein String sein und der Wert kann einem beliebigen Datentyp entsprechen. Arrays werden durch eckige Klammern (`[]`) am Anfang und Ende gekennzeichnet und enthalten eine durch Komma getrennte Liste von Werten, wie in 2.3 verdeutlicht wird. [Bra14]

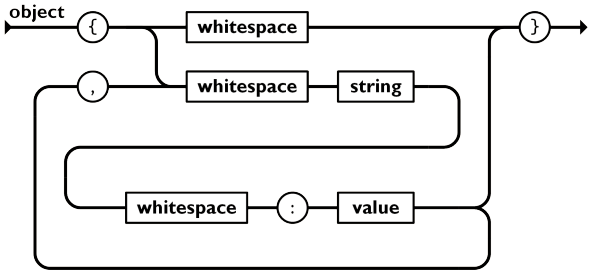


Abbildung 2.2: Objektsyntax von JSON
Quelle: https://www.json.org/ stand 16.11.2024

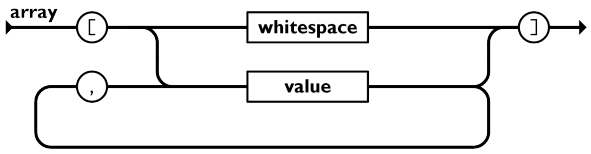


Abbildung 2.3: Arraysyntax von JSON
Quelle: https://www.json.org/ stand 16.11.2024

Vorteile von JSON gegenüber anderen Formaten sind die geringe Dateigröße durch einen kleinen Overhead, was die Datenübertragungseffizienz erhöht, eine schnelle De- und Serialisierung der Daten, sowie eine einfache Lesbarkeit. [LCCY12], [Eri24]

2.3 Backend

Ebenso, wie im Frontend, wurde auch im Backend zuvor entschieden, die Programmiersprache Python zu verwenden. Die Wahl des Frameworks ist allerdings Teil dieser Arbeit. Diese Wahl baut auf vielen davor getroffenen Entscheidungen auf und bestimmt den Programmierstil der Schnittstelle. Es gibt verschiedene Python Frameworks, die für die Web-Entwicklung genutzt werden. Die bekanntesten dabei sind Flask, Django und FastAPI. Jedes dieser Frameworks hat einzelne Vor- wie auch Nachteile und spezifische Anwendungsfälle.

Flask

Flask ist ein sogenanntes Microframework. Die Anforderungen an die Entwickelnden sind minimal und es sind nur wenige Funktionen im Detail implementiert. Zwei der wenigen integrierten Elemente sind die Webserver-Bibliothek Werkzeug, um Routing, Debugging und weitere Webserver-Funktionalitäten bereitzustellen und Jinja2, eine Template-Engine, um visuelle Ergebnisse, wie HTML zu generieren. Sofern benötigt, sollten weitere Funktionen als Bibliothek eingebunden werden. Dies sorgt für einen schlanken Aufbau, in dem theoretisch das gesamte Programm in eine Datei geschrieben werden kann sowie für eine schnelle Erfolgskurve bei kleinen Web-Anwendungen mit wenig Funktionalität. Zusätzlich ist der Einsatzbereich sehr groß, da mithilfe von verschiedenen Erweiterungen sowohl REST-APIs als auch, beispielsweise

mit Jinja2, Webseiten erzeugt werden können. Allerdings bedeutet die minimalistische Architektur auch, dass wenn aufwendigere und vor allem für den produktiven Einsatz vorgesehene Funktionalitäten, implementiert werden sollen, der Aufwand durch die Einbindung mehrerer Bibliotheken stark ansteigt. [Rel19]

Django

Django ist eines der größten und bekanntesten Python Web-Frameworks. Ähnlich, wie Flask, kann Django für viele verschiedene Anwendungsfälle, wie REST-APIs und Webseiten, verwendet werden. Allerdings ist die freie Wahl des Programmierstils dabei eingeschränkt. Mit Django geschriebene Anwendungen müssen festen Strukturen folgen, die Prinzipien, wie *Don't repeat yourself*, *Explicit is better than implicit* und *Loosely coupled architecture* einhalten. Bei großen Projekten sorgt dies für eine Erhöhung der Wartbarkeit und Entwicklungsgeschwindigkeit, sowie eine Verringerung der Fehleranfälligkeit. Zudem sind die meisten Funktionalitäten direkt mit der Installation von Django abgedeckt, und können somit gut integriert werden. Nachteile von Django sind der anfänglich hohe Aufwand, eine kleine Web-Anwendungen zu schreiben und die eher langsame Geschwindigkeit. [Rub17]

FastAPI

Als eines der schnellsten Python Web-Frameworks, bietet FastAPI viele Vorteile, um eine effiziente API zu schreiben. FastAPI ist, ähnlich wie Flask, leichtgewichtig und lässt den Entwickelnden viele Freiheiten in der Gestaltung des Aufbaus. Das Framework basiert auf vier Hauptelementen. Die Basis bilden das Framework Starlette und der Uvicorn Server. Starlette beinhaltet dabei die Grundlage der Web-Funktionalitäten. Dazu gehören die Anbindung an Funktionen, die Asynchronität ermöglichen, die Unterstützung von Websockets, eine Reihe von Event-Handlern, Routing, Templating, Cookies und Sessions. Um Starlette und FastAPI Anwendungen zu starten, bietet Uvicorn einen der schnellsten asynchronen Server mit vielen unterstützenden Funktionen, wie die uvloop, um Funktionen im Hintergrund auszuführen. Eine wichtige Funktion, um über das Web kommunizieren zu können, ist die Übersetzung und Validierung der jeweiligen Daten. Diese Aufgaben werden durch Pydantic übernommen. Dabei können Python Klassen mithilfe von Pydantic angelegt und diese dann bei Gebrauch beispielsweise in eine Datenbank übertragen oder als JSON übersetzt werden. Mithilfe von Python type hints können die Ein- und Ausgabewerte im Routing definiert werden, wodurch Pydantic diese direkt von beziehungsweise nach JSON übersetzt und dabei validiert, ob die Ein- oder Ausgaben die korrekte Form besitzen. Diese automatische Übersetzung erspart den Entwickelnden viel zum Teil redundante Schreibarbeit. Zuletzt wird durch die type hints automatisch eine Dokumentation im Sinne der OpenAPI generiert. Diese gibt als HTML- oder JSON-Dokument einen Überblick über die Schnittstellen und einen Einblick in die Struktur der Ein- und Ausgabedaten. Nachteile der FastAPI sind die mögliche Komplexität von asynchronen Funktionen und die auf die Entwickelnden ausgelagerte Aufgabe, eine gute Struktur des Codes der API zu entwerfen. Dies kann bei unvorsichtiger Wahl die Wartbarkeit erheblich verschlechtern. [Lat23]

3 Verwendung der verschiedenen Technologien in diesem Projekt

Mit dem Wissen über die vielen Technologien, die für diese API infrage kommen, kann nun entschieden werden, welche dieser Technologien sich am besten für den Anwendungsfall eignet.

3.1 Verwendung von HTTP als API-Technologie

Als Basis der Übertragung, bestimmt die Wahl des Kommunikationsprotokolls vieles in der API. Davon hängen beispielsweise die Struktur der Nachrichten, die Anzahl der Schnittstellen und die Wahl des Frameworks ab. Da die API über das Internet angesprochen werden soll, ist die Verwendung von häufig im Web eingesetzten Technologien, wie HTTP oder Websockets definitiv im Vorteil gegenüber den einfachen TCP-Sockets. Die Verwendung einer der Technologien schließt zwar die andere nicht unbedingt aus, allerdings ist es, da der notwendige Aufbau für beide sich stärker unterscheidet, wichtig, eine Technologie vordergründig zu nutzen, um Einheitlichkeit im Projekt zu schaffen. Als mögliche Kandidaten für das Kommunikationsprotokoll wurden in den Grundlagen HTTP und Websockets vorgestellt. Darüber hinaus besteht die Möglichkeit, HTTP durch eine REST-Architektur zu erweitern. Für den Anwendungsfall müssen die Kandidaten vor allem anhand von vier Anforderungen verglichen werden: Die Verbindung muss viele Daten möglichst schnell übertragen können, sie muss Anbindungen an verschiedene Funktionen bieten, sie muss einfach erweiterbar und wartbar sein und das Frontend muss auf Verarbeitungszeiten und Fehler im Backend reagieren können.

3.1.1 Gegenüberstellung der verschiedenen Technologien

Trotz ihres gemeinsamen Einsatzgebietes unterscheiden sich HTTP und Websockets stark in ihrer Verwendung. HTTP wird aufgrund des breiten Spektrums an möglichen Operationen in den meisten Anwendungen verwendet. Die Kommunikation erfolgt geregelt nach einem festen Request-Response-Prinzip, wodurch eine zuverlässige Kommunikation sichergestellt werden kann. Mithilfe von Statuscodes können Fehler schnell erkannt und behandelt werden. Die verschiedenen Methoden, sowie die Aufteilung in Pfade, ermöglichen eine schnelle und einfache Trennung der Schnittstellen und damit eine hohe Abdeckung unterschiedlicher Funktionen. Websockets hingegen ermöglichen eine bidirektionale Verbindung, in der Daten vom Server an die Clients gesendet werden können, ohne dass diese vorher angefordert werden müssen. Websockets werden daher vor allem für den Austausch von Live-Daten benötigt. Indem nach einer Berechnung eine Nachricht an die Clients zurückgeschickt wird, ermöglichen Websockets

auch ein asynchrones Arbeiten an den Daten. Zudem weisen Nachrichten über Websockets im Vergleich zu HTTP-Anfragen einen geringeren Overhead auf und sind somit schneller versendet. Allerdings führt die direkte Bindung an ein Socket anstelle mehrerer Pfade zu einem erhöhten Aufwand, um verschiedene Funktionen anzusprechen. [Wöl23]

3.1.2 Vorteile von HTTP im Anwendungsfall

Die im Anwendungsfall formulierten Kriterien können, mit Ausnahme der Geschwindigkeit, mit HTTP besser erfüllt werden. Das Backend verfügt über eine Reihe verschiedener Funktionen, um die Daten zu berechnen. Jede dieser Funktionen benötigt unterschiedliche Eingabewerte und liefert unterschiedliche Ausgabewerte. Diese Vielfalt kann am besten mit HTTP realisiert werden, da die Pfade eine eindeutige Zuordnung ermöglichen und somit diese Entscheidung nicht in die übertragenen Daten ausgelagert werden muss. Aus den HTTP-Antworten ist direkt ersichtlich, welches Ergebnis geliefert wird, ob die Bearbeitung noch länger dauert oder ob und warum die Anfrage fehlgeschlagen ist. Websockets haben zwar eine höhere Performance in der Übertragung, der mögliche Zeitaufwand wird aber hauptsächlich durch die Verarbeitungszeit der Funktionen erzeugt, weshalb der Geschwindigkeitsunterschied in der Übertragungszeit minimal ist.

Da dieser Anwendungsfall eine große Überschneidung mit den Anwendungsfällen von HTTP aufweist, ist die Verwendung von HTTP optimal. Als erweiterte Architektur für HTTP-APIs bietet sich zusätzlich REST an. Durch die Anforderung, die an eine REST-API gestellt werden, wird eine Implementierung nach Clean Code Prinzipien erzwungen. Da im Backend nur Ressourcen in Form von verarbeitenden Funktionen vorliegen, können einige Anforderungen, wie die Abbildung der HTTP-Methoden als Lese- und Schreibbefehle, nur teilweise umgesetzt werden. Für jede Schnittstelle müsste die POST-Methode verwendet werden, da diese über einen Body verfügt und damit größere Datenmengen besser übertragen kann als die GET-Methode. Andere Anforderungen, wie die Zustandslosigkeit und die Client-Server-Architektur können jedoch für eine bessere Qualität übernommen werden.

Somit kann festgelegt werden, dass eine HTTP-API die beste Möglichkeit darstellt, Frontend und Backend über das Internet zu verbinden. Dabei kann sie durch Anforderungen der REST-Architektur und einer starken Trennung der jeweiligen Schnittstellen, als direkte Abbildung von einer Funktion auf einen Pfad profitieren.

3.2 Verwendung von JSON als Datenübertragungsformat

Darüber hinaus ist zu klären, wie die vorhandenen Daten zwischen den beiden Prozessen möglichst effizient ausgetauscht werden können. Die De- und Serialisierung sollte auf beiden Seiten schnell und ohne Informationsverlust erfolgen. Die serialisierten Daten sollten einen geringen Speicherbedarf haben, um schnell über die Schnittstelle übertragen werden zu können.

XML und JSON sind als Standards zum Austausch von Daten etabliert. Dabei überzeugt XML durch eine hohe Zuverlässigkeit und Standardisierung sowie der Fähigkeit, auch komplexe Datenstrukturen ohne Informationsverlust zu beschreiben. JSON hingegen hat die Vorteile,

dass es einfach serialisiert werden kann, weniger Speicherplatz benötigt und verständlicher für Programmierer ist.

3.2.1 Gegenüberstellung der verschiedenen Technologien

Beim Vergleich von XML und JSON lassen sich unterschiedliche Ansätze der Serialisierung feststellen. Während bei XML die sichere und verlustfreie Übersetzung im Vordergrund steht, konzentriert sich die Verwendung von JSON vor allem auf die Kriterien Einfachheit und Schnelligkeit.

XML verfügt somit über eine Version der Sprache selbst, mit der neue Elemente hinzugefügt werden können, ohne alte Dokumente dabei zu gefährden. Des Weiteren gibt es zahlreiche Möglichkeiten, wie zum Beispiel XSD, um die Struktur und den Inhalt der einzelnen Tags in einem XML-Dokument zu definieren und einzuschränken. Weiterhin gibt es zu jedem öffnenden Tag auch ein schließendes Tag, wodurch die Struktur verstärkt wird. Zusätzlich gibt es zahlreiche Beschreibungsmöglichkeiten wie sprechende Bezeichnungen von Tags oder Attribute innerhalb von Tags. Allerdings leidet darunter die Lesbarkeit und der Overhead steigt an.

JSON dagegen weist keine Version auf, wodurch eine Abwärtskompatibilität erzwungen wird. Die Erweiterbarkeit von JSON wird somit zwar eingeschränkt, allerdings können alte Dokumente auch nach längerer Zeit noch interpretiert werden. Es gibt eine direkte Unterscheidung der Typen durch Literale, weshalb kein zusätzliches Dokument benötigt wird. Variablen können einfach als Schlüssel-Wert-Paare dargestellt werden. Durch die einfache Struktur mit nur wenigen wichtigen Zeichen (`"`, `{`, `[`, `:`) ist ein JSON-Dokument leicht verstanden und zu debuggen. [Sev12] [LCCY12] [Gru18]

Dies alles führt dazu, dass in XML beschriebene Objekte einen erhöhten Overhead aufweisen. Als Beispiel kann die Serialisierung einer einfachen Nachricht gezeigt werden. Diese soll die IP-Adresse des sendenden Clients und den Inhalt der Nachricht speichern. Wollte man eine solche Nachricht mit XML sicher serialisieren, bräuchte man ein entsprechendes Schema und das Dokument selbst:

XSD Schema

```
<?xml version="1.0"?>
<xs:schema id="EchoMessage"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    xmlns:tns="http://www.fh-aachen.de/">
    <xs:simpleType name="EchoMessageType"
        final="restriction">
        <xs:restriction base="xs:string">
            <xs:enumeration value="DEFAULT" />
            <xs:enumeration value="BROADCAST" />
            <xs:enumeration value="EXIT" />
            <xs:enumeration value="SHOWSTAT" />
            <xs:enumeration value="SHOWALLSTAT" />
```

```
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="echoMessage">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="sender" type="xs:string"
                    minOccurs="1" maxOccurs="1" />
                <xs:element name="content" type="xs:string"
                    minOccurs="1" maxOccurs="1" />
            </xs:sequence>
            <xs:attribute name="type"
                type="EchoMessageType" />
        </xs:complexType>
    </xs:element>
</xs:schema>
```

XML-Dokument

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<echoMessage type="DEFAULT">
    <sender>127.0.0.1</sender>
    <content>test</content>
</echoMessage>
```

Wenn dagegen das selbe Objekt mit JSON serialisiert werden sollte, bräuchte man nur ein relativ kurzes Dokument:

JSON Dokument

```
{
  "sender": "127.0.0.1",
  "content": "test"
}
```

Das JSON-Dokument ist also wesentlich kürzer und benötigt weniger Speicherplatz. Dies hat viele zeitliche Vorteile. Einerseits kann das Dokument schneller serialisiert und deserialisiert werden, andererseits kann es schneller versendet werden [LCCY12]. Dafür können komplexere Datenstrukturen mit XML sehr genau beschrieben werden.

Zusammenfassend kann gesagt werden, dass sowohl JSON als auch XML über eine sehr hohe Verbreitung und eine breite Unterstützung in den verschiedenen Programmiersprachen verfügen. Allerdings ist JSON die weitaus schnellere Variante, während XML für komplexere Anwendungsfälle besser geeignet ist.

3.2.2 Vorteile von JSON im Anwendungsfall

In der zu schreibenden API müssen Tabellen und Optionen zwischen den beiden Anwendungen versendet werden. Die Tabellen bestehen aus Spaltennamen, spaltenspezifischen generischen

Typen und den einzelnen Werten in den Zeilen. Die Optionen sind meist textbasiert und bestehen aus Listen von Schlüsselwörtern, speziellen Spaltennamen oder Nummern, die etwas über die Anzahl der Cluster aussagen. Diese Informationen können einfach als JSON serialisiert werden, ohne dass ein Schema erforderlich ist. Die Spaltennamen können als Strings beschrieben werden, die Typen werden automatisch als Literale abgeleitet, die Zeilen können als Arrays übersetzt werden. Dasselbe kann bei den Optionen angewandt werden, indem diese durch Schlüssel-Wert-Paare, Literale und Arrays übersetzt werden. Damit wäre der zusätzliche Overhead von XML-Dokumenten inklusive ihrer Schemata obsolet.

Außerdem ist JSON durch die gewählten Programmiersprachen im Vorteil. Im Frontend wird ein Node.Js Framework verwendet. Da JSON auf der Objektdefinition und den Literalen von JavaScript basiert, kann die Serialisierung sehr schnell erfolgen. Aber nicht nur das Frontend verfügt über eine schnelle Serialisierung. Auch in Python gibt es durch die Popularität von JSON, schnelle und effiziente Bibliotheken, um Strukturen als JSON zu serialisieren.

Teilweise müssen Tabellen mit mehreren tausend Zeilen über die Schnittstelle geschickt werden. Daher ist ein geringer Overhead und eine schnelle Serialisierung bedeutend für eine gute User-Experience. Aus diesen Gründen ist JSON die beste Wahl, um Werte zwischen den beiden Prozessen auszutauschen.

3.3 Verwendung von FastAPI als Framework für das Python Interface

Zuletzt steht noch die Entscheidung aus, welches Framework für die Implementierung der API verwendet werden soll. Zur Auswahl stehen dabei die drei bekannten Frameworks Flask, Django und FastAPI. Die Wahl des Frameworks basiert unter anderem auf allen vorherigen Entscheidungen. So sollte eine Synergie zwischen JSON, HTTP und dem Framework bestehen. Außerdem sollte das Framework in der Lage sein, einige anwendungsspezifische Anforderungen zu erfüllen. Die mit dem Framework geschriebene API sollte Anfragen parallel und vor allem schnell bearbeiten können. Zudem sollte sie die Aufgabe des Backends optimal abdecken. Diese besteht darin, bestimmte Funktionen an HTTP-Pfade anzubinden, während das Backend auf dem Rechner des FIR e. V. läuft. Dafür sollte genügend Funktionalität zur Verfügung stehen, aber zu viel unnötige Funktionalität vermieden werden, um den internen Overhead zu verringern.

3.3.1 Gegenüberstellung der verschiedenen Technologien

Die drei vorgestellten Frameworks unterscheiden sich anhand der Freiheit, die sie den Entwickelnden lassen, sowie anhand der Anzahl der vorinstallierten Funktionen. Flask lässt den Entwickelnden die meisten Freiheiten und hat die wenigsten vorinstallierten Funktionen. Django hingegen schränkt die Entwickelnden am meisten ein und bringt die meisten vorinstallierten Funktionen mit. FastAPI bietet ebenfalls einen freien Programmierstil, bringt aber die wichtigsten Funktionen vorinstalliert mit. Um eine Flask Applikation für den Einsatz in einer größeren API vorzubereiten, müssten weitere Pakete installiert werden, weshalb Flask meistens eher für Prototyping verwendet wird. FastAPI und Django hingegen können

schneller veröffentlicht werden. Mit sowohl Flask als auch FastAPI können vor allem kleinere APIs sehr schnell geschrieben werden. Django ist dagegen bei großen Webanwendungen mit vielen Entwickelnden und vielen Funktionen im Vorteil. Zudem wird mit Django oft auch das Frontend direkt mitgeliefert.

3.3.2 Vorteile von FastAPI im Anwendungsfall

Die zu schreibende API besteht aus einer Reihe von verarbeitenden Funktionen, die ausschließlich aus ihren Eingabewerten, Ergebnisse berechnen und diese zurückgeben. Es wird keine Datenbank, Session und keine HTML-Ausgabe benötigt. Die Funktionen sollten an Pfade gebunden sein, die mit HTTP POST-Anfragen angesprochen werden können. Außerdem sollte die API performant sein und eine einfache Möglichkeit bieten, JSON Ein- und Ausgabedaten zu verarbeiten. Alle drei Frameworks können diese Anforderungen erfüllen, wobei FastAPI allerdings dafür am besten geeignet ist. Einer der wichtigsten Faktoren ist die Geschwindigkeit. APIs, die mit FastAPI geschrieben wurden, sind durch Uvicorn und Starlette weitaus schneller als APIs, die mit Flask oder Django geschrieben wurden. Ein weiterer Vorteil von FastAPI ist die einfache Schreibweise, durch welche kleine bis mittelgroße APIs auch von Fachfremden schnell geschrieben werden können. Ein dritter wichtiger Faktor ist die direkte Übersetzung zwischen JSON und Python Objekten durch Pydantic. Dadurch ist es möglich, sowohl problemlos zwischen JavaScript und Python über JSON zu kommunizieren als auch Ein- und Ausgabedaten direkt zu überprüfen, so dass keine Informationen ausgelassen werden. Ein weiterer Vorteil ist die automatische Erstellung von Dokumentation. Diese ermöglicht anderen Entwickelnden des Frontends, die benötigte Datenstruktur schnell zu verstehen und die Definition der Schnittstellen einzusehen, ohne sich vorher mit FastAPI oder Python vertraut machen zu müssen.

Zusammenfassend lässt sich festhalten, dass das Framework FastAPI eine hohe Performance, einen einfachen Programmierstil, eine gute Serialisierung der Daten, eine automatische Typenprüfung und eine automatisch generierte Dokumentation bietet. Daher stellt es die beste Wahl für die Grundlage dieser API dar.

4 Implementierung der Schnittstelle

4.1 Aufbau der Datenstruktur und praktischer Ansatz der API

4.1.1 Wahl der Datenstruktur im JSON-Format

Die Struktur der zu versendenden Informationen lassen sich in zwei Hauptgruppen aufteilen:

Datenstruktur für Tabellen

Tabellen enthalten die eigentlichen Daten, geordnet nach Spaltennamen und mit einem festen Typen. Dabei kann es sich um Benutzereingaben handeln, aber auch um Werte, die im Code verändert oder geclustert wurden. Um dies in JSON darzustellen, bietet sich eine spaltenweise Objektsyntax an. Ein Objekt repräsentiert dabei die Tabelle, während die Schlüssel des Objekts die Spaltennamen darstellen. Als Werte für die Schlüssel können alle vertikal ausgelesenen Daten einer Spalte in einem Array eingegliedert werden. Beim Empfang in der API wird diese Struktur als ein Dictionary aus String-Schlüsseln und List-Werten übersetzt. Diese Struktur hat vor allem zwei verschiedene Vorteile. Der erste bezieht sich auf die Geschwindigkeit der Übertragung und Serialisierung, da die Anzahl der benötigten Symbole minimiert wird und es keine Wiederholungen von beispielsweise Spaltennamen gibt. Der zweite Vorteil bezieht sich auf die Verwendung von Pandas als Bibliothek für die Datenverarbeitung. Pandas verfügt mit der Funktion `pandas.DataFrame.from_dict` über eine direkte Möglichkeit, diese Datenstruktur effizient als Dataframe zu interpretieren, ohne zuvor langsame Python-Algorithmen zur Transformation der Struktur ausführen zu müssen.

Beispiel Tabelle als JSON Objekt

```
{
  "FirstColumnName": ["FirstValue", "SecondValue", ...],
  "OtherColumnName": [1, 4, 5, 2, ...],
  ...
}
```

Datenstruktur für Optionen

Die Optionen enthalten verschiedene Parameter, unter denen die jeweiligen Funktionen unterschiedliche Ergebnisse liefern. Dies können die Anzahl der Cluster, die Aufteilung der Spalten in Binär, Kategorisch, Numerisch oder Schlüssel und ähnliche Parameter sein. Die Möglichkeiten, diese Optionen zu übersetzen, sind eine Array-Schreibweise und eine Objekt-Schreibweise. Die

Array-Schreibweise weist den Vorteil auf, dass weniger Overhead produziert wird. Allerdings ist die Reihenfolge der Werte im Array wichtig. Werden die Optionen jedoch als Objekt definiert, können sie anhand der Schlüsselnamen erkannt werden, ohne auf die Reihenfolge achten zu müssen. Außerdem kann durch Pydantic festgestellt werden, ob die Typen der Optionen korrekt sind. Aus diesen Gründen ist es vorteilhafter, den geringen zusätzlichen Overhead in Kauf zu nehmen, um die Position und den Typ der Optionen zu sichern.

Datenstruktur für Operationen mit mehreren Parametern und Rückgabewerten

Eine Anfrage mit kombinierten Informationen aus verschiedenen Tabellen und verschiedenen Optionen kann am effizientesten durch ein doppelt geschachteltes Objekt gelöst werden. Das Objekt, also der Inhalt des Bodies der Anfrage, besitzt auf der obersten Ebene zwei Attribute. Die Schlüssel der Attribute sorgen für eine Trennung zwischen Tabellen und Optionen. Als Wert des Attributs für die Tabellen bietet sich ein weiteres Objekt an. In dieses können die verschiedenen Tabellen als Schlüssel-Wert-Paare eingefügt werden. Die Schlüssel sind dabei die Namen der Tabellen, während die Werte jeweils die Tabellen in der zuvor beschriebenen spaltenweisen Darstellung darstellen. Die Vorteile dieser Darstellung der Tabellen sind dieselben, wie die bei der Darstellung der Optionen. Als Wert des Attributs für die Optionen kann die beschriebene Objektstruktur der Optionen verwendet werden. Somit können alle benötigten Funktionen effizient und mit geringem Fehlerpotential zwischen den Prozessen ausgetauscht werden.

Beispiel gesamter Anfrage-Body als JSON Objekt

```
{
  "tables": {
    // Codierung der einzelnen Tabellen s.O.
    "firstTablename": {
      "FirstColumnName": ["firstValue", ...],
      ...
    },
    "secondTablename": ...
  },
  "options": {
    "firstOptionname": "Optionvalue",
    "secondOptionname": ["firstValue", "secondValue", ...],
    "thirdOptionname": 3,
    ...
  }
}
```

4.1.2 Aufbau der API, mit strenger Typenprüfung und clean code Ansätzen

Das Framework FastAPI ermöglicht den Entwickelnden eine große Freiheit bei der Gestaltung der API. Mithilfe von Pydantic und Python type hints können Anforderungen an die Ein- und die Ausgabedaten gestellt werden. Die Stärke dieser Anforderungen kann ebenfalls variabel

durch die Entwickelnden bestimmt werden. Somit stellt sich die Frage, wie die einzelnen Aspekte und Funktionen der API effizient und clean realisiert werden können und wie stark die Typisierung sowie Fehlervermeidung umgesetzt werden.

Strenge Typisierung durch Pydantic models und Python type hints

Python ist in der Regel eine Sprache mit loser Typisierung. Seit der Python Version 3.5 gibt es die sogenannten Python type hints. Diese vereinfachen den Umgang mit Variablen und Funktionen durch Warnungen bei abweichenden Ein- oder Rückgabewerten und verbesserten Codevorschlägen. Diese Funktion nutzt Pydantic und damit auch die FastAPI, zur Definition von Ein- und Ausgabeparametern. Die Typen und Namen der einzelnen Attribute des JSON-Bodies von Anfrage und Antwort können durch Klassen, die von Pydantics BaseModel erben, genau definiert werden. Dabei können einfache Klassen, tiefe Verschachtelungen und sogar generische Typen verwendet werden. Die einzelnen Funktionen, die durch HTTP-Pfade abgedeckt werden, benötigen in den meisten Fällen sehr spezifische Parameter. Dies können mehrere verschiedene Tabellen, oder auch eine Liste von Spaltennamen als Optionen sein. Um eine Verwechslung der verschiedenen Parameter zu vermeiden, wurde, wie im Abschnitt über die JSON-Struktur beschrieben, eine Aufteilung der Parameter mit den Parameternamen als Schlüssel der Attribute eines Objektes gewählt. Zusätzlich ist es nun auf Seiten der API möglich, den Typ der Werte der Attribute festzulegen. Dadurch kann einer Verwechslung der Parameter vorgebeugt werden. Im Framework sind automatische Typenprüfungen sowie eine Fehlerbehandlung bei Fehlschlag der Typenprüfung integriert. So wird beispielsweise ein vordefinierter Fehler mit Statuscode 422 und einer Beschreibung automatisch generiert, wenn das JSON-Objekt in der Anfrage nicht mit dem vorbestimmten Hint übereinstimmt. Ein Beispiel dafür wäre:

```
# Other imports
from pydantic import BaseModel
# Other functionality

class ExampleTables(BaseModel):
    exampleOne: Dict[str, List]
    exampleTwo: Dict[str, List]

class ExampleOptions(BaseModel):
    name: str
    amount: int

class ExampleParameter(BaseModel):
    tables: ExampleTables
    options: ExampleOptions

@app.post("/example")
def examplePath(parameters: ExampleParameter
    ) -> Dict[str, List]:
    # Converting
    response = # Method that generates a table
```

```
return response
```

Beispiel Python type hints und Pydantic in einer Schnittstelle

Dort werden zuerst drei models erzeugt, die von `pydantic.BaseModel` erben. Damit wird ein Request-Objekt, nach der zuvor beschriebenen Struktur erzeugt. Dieses beinhaltet zwei Tabellen, `exampleOne` und `exampleTwo`, sowie zwei Optionen, `name` als String und `amount` als Integer. In der Funktion `examplePath` wird für den Pfad `/example` festgelegt, dass als Eingabeobjekt des Bodies `ExampleParameter` erwartet wird. Als Rückgabewert wird eine Tabelle vorausgesetzt. Durch Pydantic würde das Eingabeobjekt auf korrekte Attributnamen und die Ein- und Ausgabewerte auf korrekte Typen überprüft. Die automatische Dokumentationsfunktion von FastAPI würde diese type hints übernehmen und beispielsweise diese Visualisierung generieren:

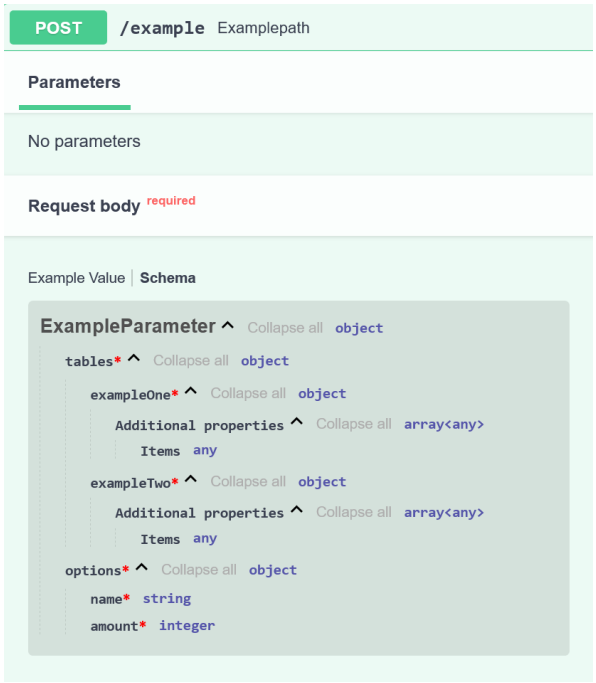


Abbildung 4.1: Dokumentation type hints und Pydantic Schnittstelle

Wie bereits erläutert, kann die JSON-Datenstruktur von Tabellen in Python als Dictionary mit Strings als Schlüsseln und Listen als Werte beschrieben werden. Für die Optionen könnte ebenfalls ein Dictionary verwendet werden. Allerdings ist bei diesem Vorgehen keine automatische Prüfung der Werte möglich. Dies müsste also manuell in unübersichtlichen und möglicherweise langsamen Abfragen erfolgen. Daher ist eine Klasse, die von `pydantic.BaseModel` erbt, wesentlich besser geeignet. In dieser können die Anzahl der Attribute, die Namen der Attribute und die Typen genau beschrieben werden. Dasselbe ist ebenfalls für die verschiedenen Tabellen vorteilhaft. Um die Klassen ohne viele Wiederholungen zu erstellen, ist eine Schreibweise mit generischer Typisierung optimal. Die oberste Klasse besitzt demnach zwei Attribute, eines für die Tabellen und eines für die Optionen. Diese sind vom Typ generisch, sodass bei Bedarf eine Klasse mit den exakten Angaben geschrieben und als generischer Typ eingesetzt werden

kann. Auf diese Weise kann die Datenstruktur sauber, sicher und streng in die API integriert werden.

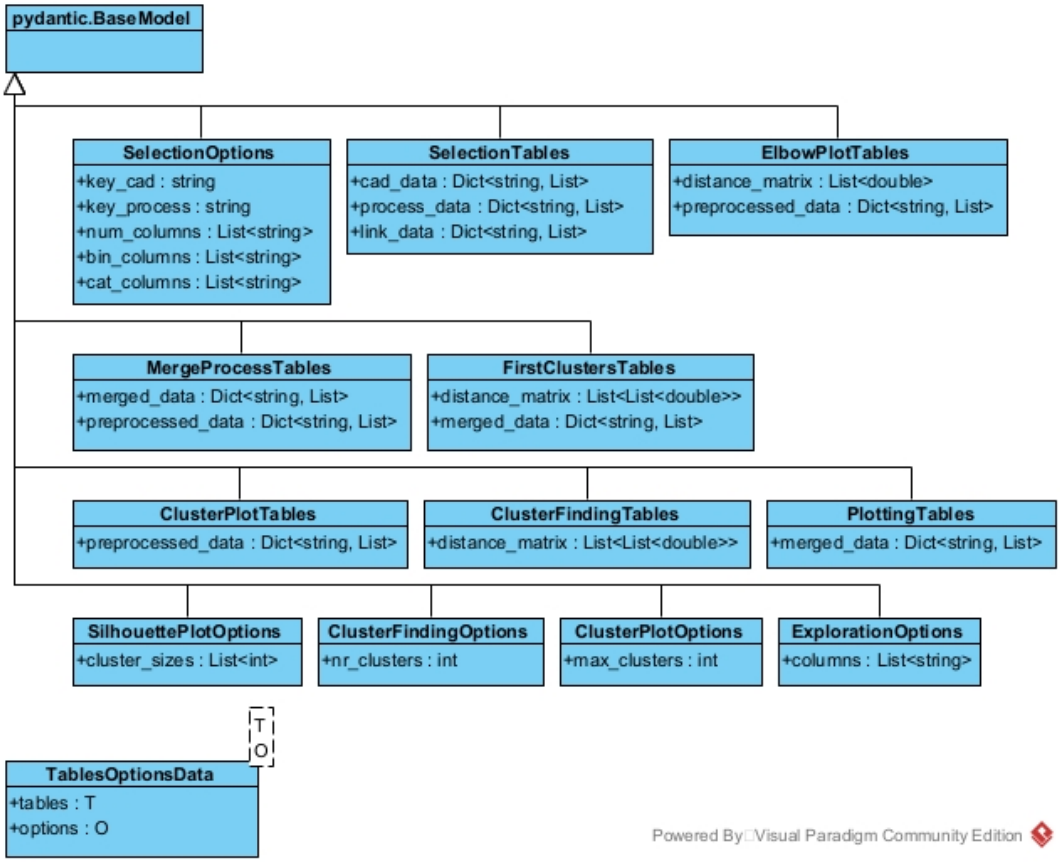


Abbildung 4.2: Models Struktur als UML-Diagramm

Strategy Pattern zur einfachen und schnellen Erstellung von asynchronen Routen

Ein großer Vorteil der FastAPI ist die Möglichkeit, einfach asynchrone Funktionen zu schreiben. Diese können durch Uvicorn und Starlette sehr effizient im Hintergrund ausgeführt werden, während die API für parallele Anfragen offen bleibt.

Um parallele Anfragen zu ermöglichen, muss für jede Anfrage zunächst die Eventloop geholt werden. Im Anschluss muss die eigentliche Methode darin ausgeführt werden. Schließlich muss per `await` auf das Ergebnis gewartet werden. Ohne diese Schritte müsste eine weitere Anfrage auf die Bearbeitung der ersten warten. Eine ständige Wiederholung dieser Schritte im Code würde jedoch zu einer hohen Redundanz und damit zu einer hohen Fehleranfälligkeit beim Editieren führen. Durch die Verwendung von Software-Patterns, kann dem entgegengewirkt werden. Das Strategy Pattern besteht aus einem Interface, welches für den Kontext benötigte, abstrakte Funktionen beinhaltet, sowie den jeweiligen Unterklassen dieses Interfaces, welche dem Anwendungsfall entsprechende Realisierungen der Funktionen aufweisen. Im Kontext kann dann je nach aktuell benötigter Funktionalität die Implementation des Interfaces gewechselt

werden. Im Anwendungsfall besteht der Kontext aus den zuvor genannten Schritten zur asynchronen Gestaltung komplexer Berechnungen, sowie dem jeweiligen Pfad. Der Pfad bestimmt dabei über die Funktionsbindung und eine Dependency Injection die jeweilige Implementation des Interfaces. Das Interface enthält lediglich eine Funktion, die durch Generics variable Parameter und Rückgabewerte besitzt. Die jeweiligen erbenden Strategien überschreiben diese Methode mit ihren zuvor bestimmten und dort festen Typen. In ihnen erfolgt eine Übersetzung der Eingabedaten der HTTP-Anfrage zu Parametern, welche von den bearbeiteten Funktionen benötigt werden. Anschließend werden die Funktionen aufgerufen, wobei die Rückgabewerte wieder in ein Objekt für die Antwort übersetzt und an den Kontext übergeben werden. Somit muss beim Anlegen eines neuen Pfades in der Schnittstelle nur eine weitere Strategie geschrieben und diese in per Dependency Injection an die asynchrone Funktion im Kontext übergeben werden.

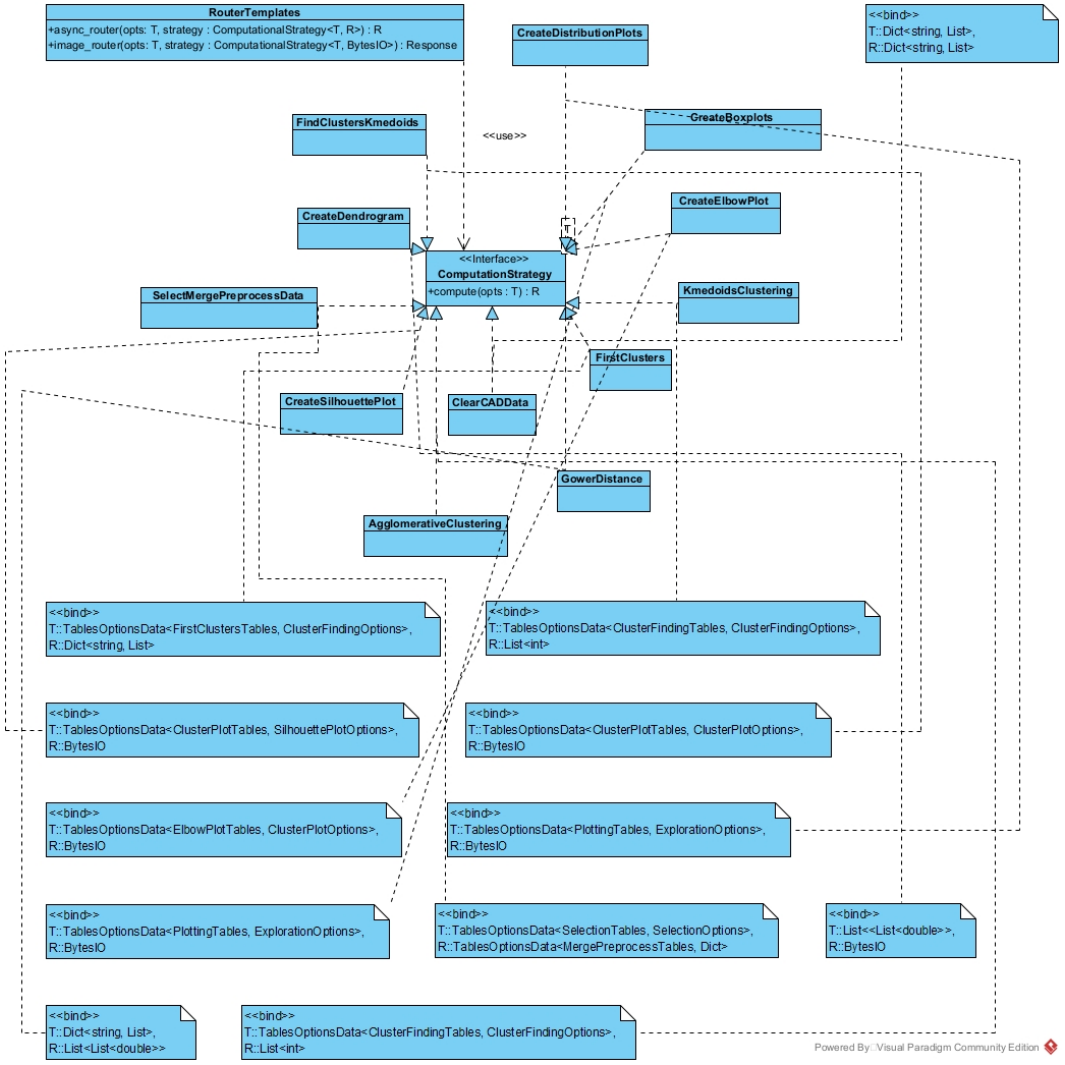


Abbildung 4.3: Verwendete Strategy Struktur als UML-Diagramm

Ordnerstruktur der verschiedenen Funktionen

Durch die lose Struktur, die FastAPI den Entwickelnden zur Verfügung stellt, wird die Struktur der Anwendung fast vollständig den Entwickelnden überlassen. Dies geht so weit, dass theoretisch ganze APIs in eine einzige Datei geschrieben werden können. Eine solche Struktur würde jedoch die Wartbarkeit verschlechtern und die Fehleranfälligkeit erhöhen, da sie die Prinzipien des Clean Code verletzt. Daher ist eine gut durchdachte Ordnerstruktur wichtig.

Zunächst ist die Verwendung von Routern ein wichtiger Schritt, um die Datei `main.py` zu entlasten und die Definition der verschiedenen Routen auszulagern. Weiterhin ist es sinnvoll, die Klassen, die für die Typisierung der API erstellt werden, ab hier Models genannt, ebenfalls in separate Dateien zu schreiben. Hinzu kommen die extern programmierten Funktionen, welche in vier Dateien übergeben werden, also bereits ausgelagert sind. Diese Ansammlung von Dateien muss aber noch in eine logische Ordnerstruktur gebracht werden, damit die einzelnen Elemente leichter gefunden werden können. Eine mögliche Gliederung wäre die Trennung nach Funktionalität der vorgegebenen Datascience Methoden. Diese gliedern sich in die Bereiche Preparation, Exploration, Clustering und Validation. So könnte man vier Ordner mit jeweils einem Router, einer Datei mit den extern programmierten Funktionen und einer Datei für die in den Schnittstellen verwendeten Modelle anlegen. Dies würde jedoch die Wiederverwendung einiger Modelle in einem anderen Funktionsbereich erschweren. Daher ist eine Aufteilung nach den internen Aufgaben in der API die bessere Lösung. Dabei ist die Bildung von drei Ordnern erforderlich: ein Ordner für Router, ein Ordner für Models und ein Ordner für externe Funktionen. Diese Aufteilung würde die Trennung der Verantwortlichkeiten und die REST-Architektur besser umsetzen. Dies ist darin begründet, dass die Zuständigkeiten eher API-internen Aufgaben, wie Routing und Verarbeitungslogik entsprechen und weniger den von außen definierten funktionalen Unterschieden. Eine solche funktionale Trennung in mehrere Dateien kann jedoch auch innerhalb der Ordner erfolgen, solange der aufgeteilte Code nicht an anderen Stellen wiederverwendet werden kann. So sollte der Code für das zuvor beschriebene Strategy-Pattern für jeden Router in einer Datei zur Verfügung stehen. Aufgrund der schnell wachsenden Anzahl an Strategien ist es auch denkbar, einen weiteren Ordner, in dem bereits existierenden Ordner der Router, für Strategy-Dateien anzulegen. Die Models können hier ähnlich wie die Router aufgeteilt werden. Für wiederverwendbare Models wie `TablesOptionsData` kann eine separate Datei deklariert werden, während spezifische funktionsgebundene Models in jeweilige Dateien ausgelagert werden. Die externen Funktionen würden in einem separaten Ordner zusammengefasst werden.

4.2 Verbinden des Frontends mit dem Backend

Die effiziente Anbindung des Frontends an die API ist ebenso wichtig, wie eine schnelle API im Backend. Natürlich ist es möglich, eine Reihe von Schaltflächen darzustellen, die jeweils einen Pfad adressieren und dann den Nutzern eine Reihenfolge vorzugeben. Dies würde jedoch die User-Experience, durch lange Wartezeiten und eine hohe Wahrscheinlichkeit, die Prozedur neu starten zu müssen, wenn die Reihenfolge nicht eingehalten wird, beeinträchtigen. Um die Handhabung des Tools zu erleichtern, ist es erforderlich, eine automatische Reihenfolge zu

erzwingen, Wartezeiten möglichst zu verbergen, und den Nutzern eine klare Orientierung zu ermöglichen.

4.2.1 Finden einer effizienten Reihenfolge der Anfragen

Vorhandene Schnittstellen

Innerhalb der vier Hauptfunktionen Preparation, Exploration, Clustering und Validation sind zahlreiche kleinere Funktionen integriert. Für jede Funktion existiert eine eigene Schnittstelle. Zu Beginn der Laufzeit sollten im Frontend drei Tabellen vorliegen, nämlich eine automatisch erstellte Tabelle mit CAD-Daten, eine mit Prozessdaten und eine, um beide zu verbinden.

Tabelle 4.1: Vorhandene Schnittstellen

Funktionsname	**Erklärung**
Clear CAD Data	Mit dieser Funktion werden alle CAD-Daten so formatiert, dass sie danach vom Backend besser verarbeitet werden können.
Select Merge Preprocess Data	Alle eingangs hochgeladenen Tabellen werden zusammengeführt und Duplikate entfernt. Zusätzlich wird eine Kopie der verschmolzenen Tabelle, im Folgenden merged Tabelle genannt, mit kombinatorischen Werten angereichert. Diese angereicherte Tabelle wird im Folgenden als preprocessed Tabelle bezeichnet. Für diese Schritte müssen die Spalten typisiert werden. Dabei ist zu unterscheiden, ob es sich um Schlüssel der Tabellen, numerische Daten, kategorische Daten oder binäre Daten handelt.
Create Boxplots	Mithilfe der merged Tabelle können für ausgewählte Spalten Boxplots erstellt werden.
Create Distribution Plots	Mithilfe der merged Tabelle können für ausgewählte Spalten Histogramme erstellt werden.
Gower Distance	Mithilfe der merged Tabelle wird eine Distanzmatrix zwischen den Einträgen bestimmt.
Plot Dendrogram	Mithilfe der Distanzmatrix wird ein Dendrogramm des hierarchischen Clusterings erzeugt.
Elbow Plot	Mithilfe der preprocessed Tabelle und der Distanzmatrix wird bis zu einer gewünschten maximalen Anzahl von Clustern ein Ellbogen Plot des agglomerativen Clusterings erstellt.
Find Clusters K-Medoids	Mithilfe der preprocessed Tabelle wird bis zu einer gewünschten maximalen Anzahl von Clustern ein Ellbogen und ein Silhouetten Plot des K-Medoids-Verfahrens generiert.
Silhouette Plot	Mithilfe der preprocessed Tabelle werden für mehrere gewünschte Clustergrößen Silhouetten Plots erstellt.
Agglomerative Clustering	Mithilfe der preprocessed Tabelle wird für eine gewünschten Anzahl von Clustern das agglomerative Clustering-Verfahren angewandt und die Clusterlabels der Einträge zurückgegeben.

Funktionsname	Erklärung
K-Medoids Clustering	Mithilfe der preprocessed Tabelle wird für eine gewünschten Anzahl von Clustern das K-Medoids-Verfahren angewandt und die Clusterlabels der Einträge zurückgegeben.
First Clusters	Mithilfe der merged Tabelle und der Distanzmatrix wird für eine gewünschten Anzahl von Clustern die ersten Cluster mit der kleinsten Distanz in einem agglomerativen Clustering-Prozess zurückgegeben.
Evaluate Clustering	Für die Verbindung aus der merged Tabelle und den jeweiligen Clusterlabels werden Metriken, wie der Davies-Bouldin-Index, der Dunn-Index, der Calinski-Harabasz-Score und der Silhouette-Koeffizient berechnet, um eine Bewertung vorzunehmen.
Feature Importance	Für die Verbindung aus der merged Tabelle und den jeweiligen Clusterlabels wird ein Balkendiagramm erstellt, um die Relevanz von Features für die Clusterbildung mithilfe eines Random-Forest-Classifiers zu veranschaulichen.
Plot Results 2D	Für die Verbindung aus der merged Tabelle und den jeweiligen Clusterlabels wird nach einer t-SNE Dimensionsreduktion eine zweidimensionale Visualisierung der Cluster und Datenpunkte erstellt.

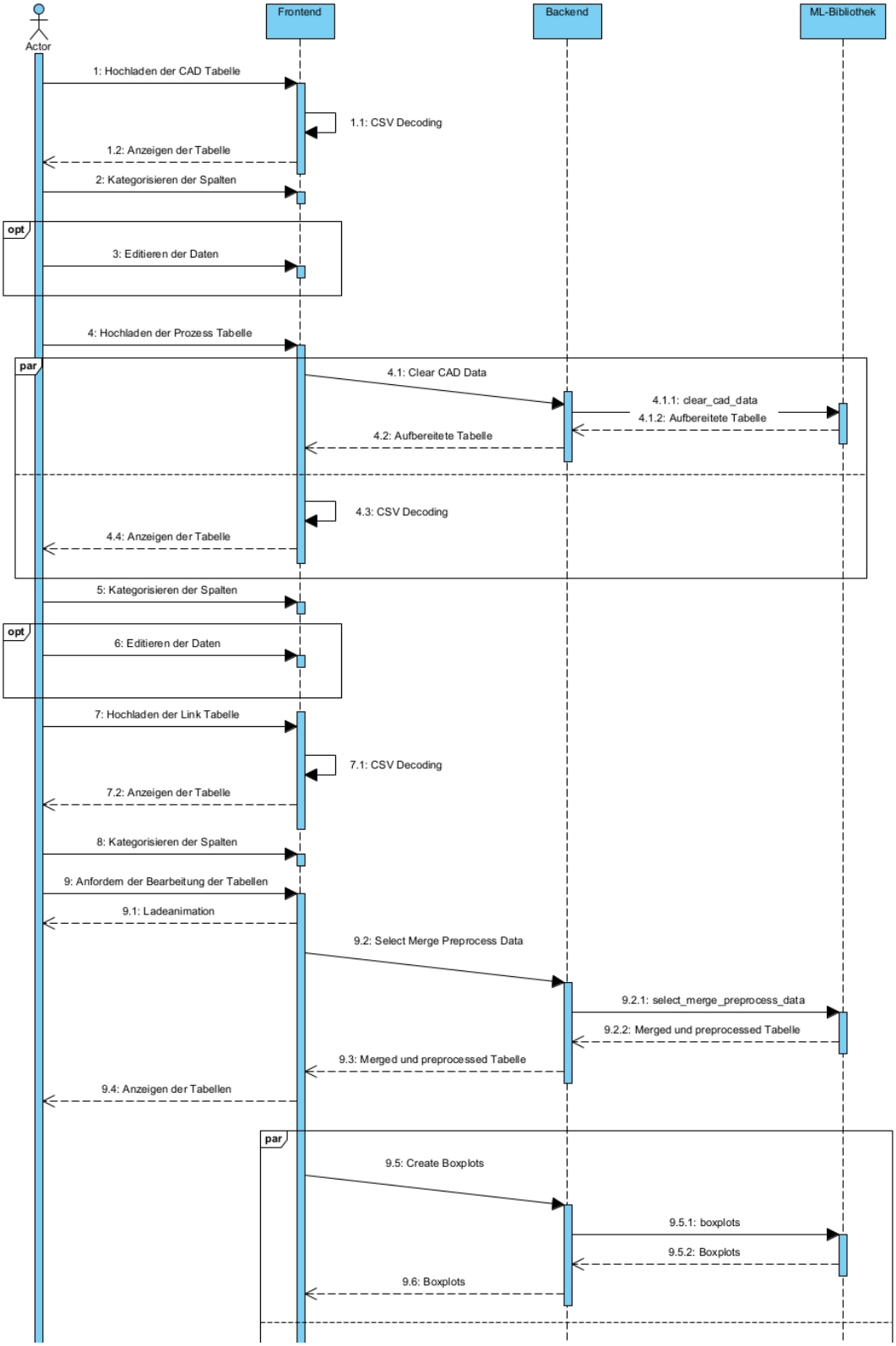
Die in der Tabelle 4.1 dargestellte Reihenfolge der Funktionen lässt sich auf die Abfolge der Aufrufe übertragen. Um nach *Select Merge Preprocess Data* korrekte Werte zu erhalten, müssen die CAD-Daten zuvor mit *Clear CAD Data* transformiert werden. Sowohl die merged Tabelle als auch die preprocessed Tabelle bilden danach die Grundlage für die nachfolgenden Funktionen. Ebenso wichtig ist auch das Ergebnis der *Gower Distance* Funktion. Für *Elbow Plot* und *Find Clusters K-Medoids* ist eine Eingabe der maximalen Clusteranzahl durch die User erforderlich. Für *Silhouette Plot*, *Agglomerative Clustering*, *K-Medoids Clustering* und *First Clusters* ist zusätzlich eine vorherige exakte Eingabe von Clustergrößen oder -anzahl wichtig. Die Bewertung der Cluster durch *Evaluate Clustering*, *Feature Importance* und *Plot Results 2D* erfordert wiederrum die zuvor berechneten Clusterlabels.

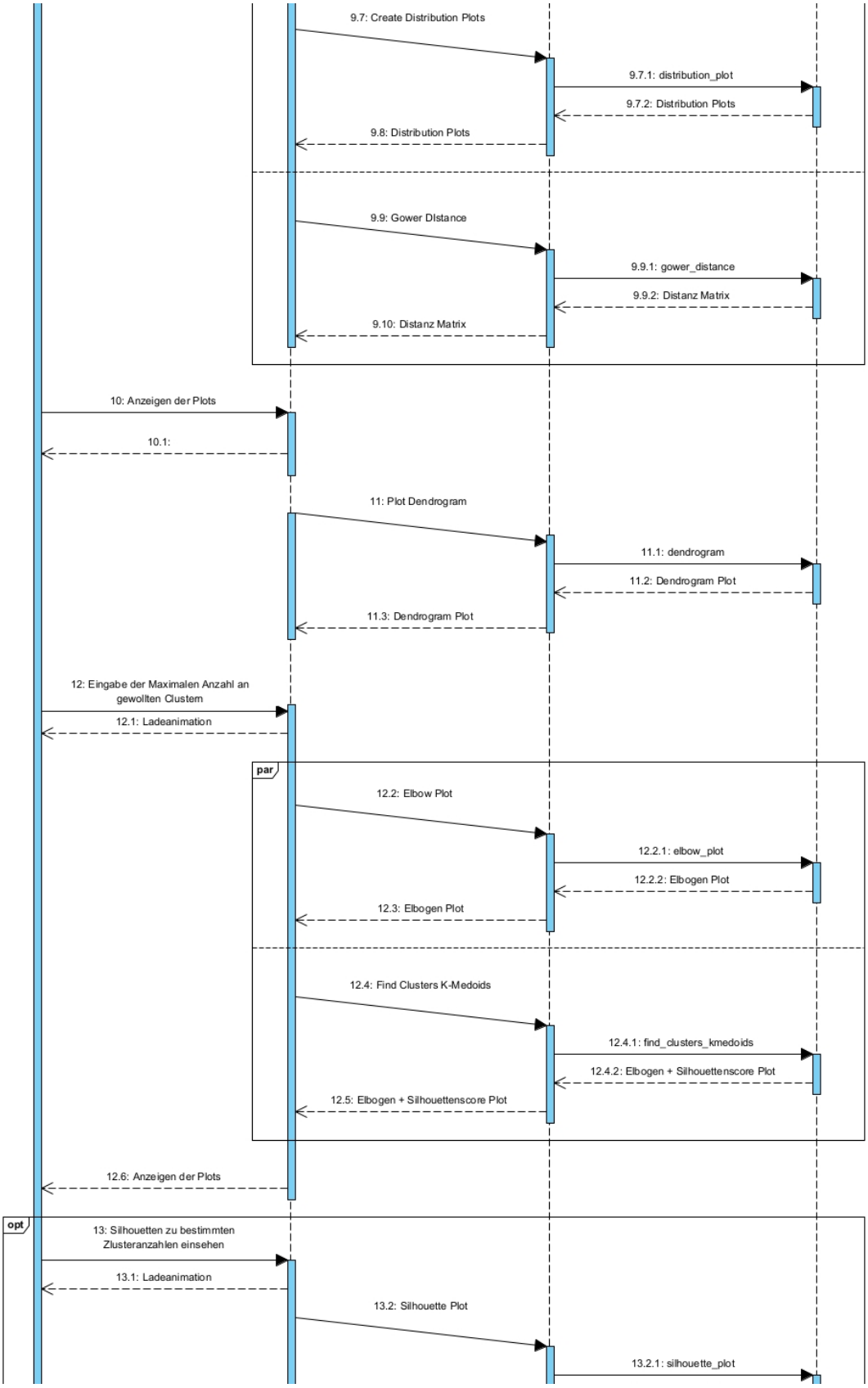
Die effizienteste Reihenfolge, um Wartezeiten zu minimieren

Viele dieser Methoden bauen auf den Ergebnissen der vorherigen auf oder können aufgrund der benötigten Eingabedaten erst ab einem bestimmten Punkt aufgerufen werden. Wenn jedoch Funktionen gleiche oder ähnliche Eingabedaten haben, können diese auch parallel durch den asynchronen Server aufgerufen werden. Da einige Funktionen aufgrund großer Datenmengen längere Verarbeitungszeiten aufweisen können, sollte die Verarbeitung möglichst ohne Beeinträchtigung der Nutzer im Hintergrund verlaufen. Dies wird teilweise durch größere Nutzereingaben erleichtert. Für *Select Merge Preprocess Data* müssen alle Tabellen, also CAD-Daten, Prozessdaten und die Verbindungsdaten der beiden Tabellen, sowie eine Markierung der Spalten, als Schlüsselspalte, numerisch, kategorisch oder binär vorliegen. Die CAD-Daten müssen jedoch zuvor mit *Clear CAD Data* transformiert werden. So kann festgelegt werden, dass die Nutzer zuerst die CAD-Daten hochladen und die Spalten kennzeichnen

müssen. Während dies anschließend mit den Prozessdaten und der Verbindungstabelle durchgeführt wird, kann mit JavaScript Promises *Clear CAD Data* im Hintergrund unbemerkt aufgerufen und das Ergebnis im Frontend überschrieben werden. Nachdem alle Tabellen hochgeladen und alle Spalten markiert wurden, ist eine kurze Bearbeitungszeit der Methode *Select Merge Preprocess Data* unvermeidlich. Danach können jedoch wieder andere Funktionen im Hintergrund aufgerufen werden, während die User die merged Daten einsehen können. Die Funktionen *Create Boxplots*, *Create Distribution Plots* und *Gower Distance* sind alle unabhängig voneinander und benötigen die merged Daten. Daher können unmittelbar nach Erhalt der Ergebnisse von *Select Merge Preprocess Data* parallele Anfragen an das Backend für die drei Funktionen gestellt werden. Dabei ist es am effizientesten, die unwichtigen Spalten bereits bei der Dateneingabe entsprechen markieren zu können. So können nun für alle als nicht irrelevant gekennzeichneten Spalten Plots erzeugt werden, ohne dass eine erneute Eingabe der User erforderlich ist. Mit den, aus den Plots gewonnenen Erkenntnissen können weitere Spalten manuell als unwichtig aussortiert werden. Nach dem Filtern der Spalten kann die maximal gewünschte Anzahl von Clustern pro Verfahren eingegeben werden, während mit *Plot Dendrogram* ein Dendrogramm erstellt wird. Dieses kann nach der Ladezeit zusätzlich als Entscheidungshilfe dienen. Nachdem alle Entscheidungen über die maximale Anzahl von Clustern getroffen wurden, können parallel mit *Elbow Plot* und *Find Clusters K-Medoids* Diagramme erstellt werden. Diese sollten Aufschluss über die bevorzugte Anzahl von Clustern geben. Von hier aus haben die Nutzer zwei Möglichkeiten. Sie können bereits eine Auswahl über die zu verwendenden Clustering-Verfahren und die Anzahl der Cluster treffen. Zum anderen können zunächst mithilfe von Silhouetten Plots bestimmte Clustergrößen näher betrachtet werden. Für die Silhouetten Plots müssen zunächst die gewünschten Clustergrößen eingegeben und anschließend mit der Funktion *Silhouette Plot* die entsprechenden Grafiken erzeugt werden. Weiterhin können nach Angabe des Clustering-Verfahrens und der Clusteranzahl Kombinationen der jeweiligen Funktionen *Agglomerative Clustering*, *K-Medoids Clustering* und *First Clusters* parallel gestartet werden. Welche Kombination dieser Funktionen aufgerufen wird, hängt von der Auswahl des Verfahrens ab. Nachdem die Ergebnisse der verschiedenen Clustering-Verfahren vorliegen, können alle Methoden zur Validierung der Cluster gleichzeitig aufgerufen werden. So können die User die verschiedenen Daten nach Clusterlabel geordnet betrachten, während im Hintergrund *Evaluate Clustering*, *Feature Importance* und *Plot Results 2D* berechnet werden. Die Ergebnisse der Validierung ermöglichen es den Nutzern, die verschiedenen Entscheidungen, wie die Kategorisierung der Spalten oder die Anzahl der Cluster, zu überdenken und den Prozess an dieser Stelle erneut zu beginnen.

Diese Reihenfolge wird in dem Sequenzdiagramm 4.5 einmal veranschaulicht.


Actor
Frontend
Backend
ML-Bibliothek
1: Hochladen der CAD Tabelle
1.1: CSV Decoding
1.2: Anzeigen der Tabelle
2: Kategorisieren der Spalten
opt
3: Editieren der Daten
4: Hochladen der Prozess Tabelle
par
4.1: Clear CAD Data
4.1.1: clear_cad_data
4.1.2: Aufbereitete Tabelle
4.2: Aufbereitete Tabelle
4.3: CSV Decoding
4.4: Anzeigen der Tabelle
5: Kategorisieren der Spalten
opt
6: Editieren der Daten
7: Hochladen der Link Tabelle
7.1: CSV Decoding
7.2: Anzeigen der Tabelle
8: Kategorisieren der Spalten
9: Anfordern der Bearbeitung der Tabellen
9.1: Ladeanimation
9.2: Select Merge Preprocess Data
9.2.1: select_merge_preprocess_data
9.2.2: Merged und preprocessed Tabelle
9.3: Merged und preprocessed Tabelle
9.4: Anzeigen der Tabellen
par
9.5: Create Boxplots
9.5.1: boxplots
9.5.2: Boxplots
9.6: Boxplots


9.7: Create Distribution Plots
9.7.1: distribution_plot
9.7.2: Distribution Plots
9.8: Distribution Plots
9.9: Gower DIstance
9.9.1: gower_distance
9.9.2: Distanz Matrix
9.10: Distanz Matrix
10: Anzeigen der Plots
10.1:
11: Plot Dendrogram
11.1: dendrogram
11.2: Dendrogram Plot
11.3: Dendrogram Plot
12: Eingabe der Maximalen Anzahl an gewollten Clustern
12.1: Ladeanimation
par
12.2: Elbow Plot
12.2.1: elbow_plot
12.2.2: Elbogen Plot
12.3: Elbogen Plot
12.4: Find Clusters K-Medoids
12.4.1: find_clusters_kmedoids
12.4.2: Elbogen + Silhouettenscore Plot
12.5: Elbogen + Silhouettenscore Plot
12.6: Anzeigen der Plots
opt
13: Silhouetten zu bestimmten Clusteranzahlen einsehen
13.1: Ladeanimation
13.2: Silhouette Plot
13.2.1: silhouette_plot

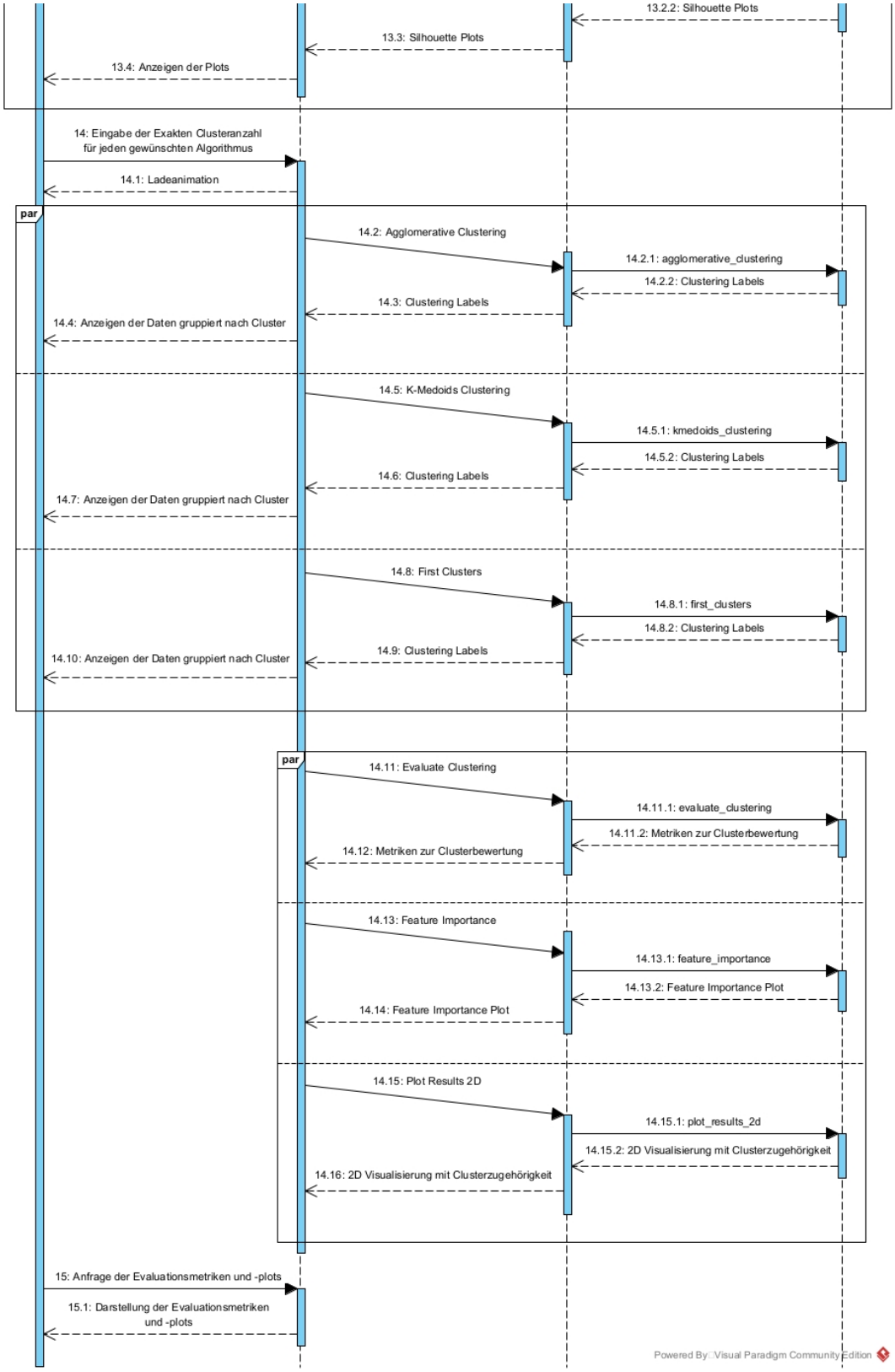


Abbildung 4.5: Sequenzdiagramm Abfolge der Anfragen

Reaktion des Frontends auf längere Wartezeiten

Trotz der vielen parallelen Schritte kommt es immer noch zu unvermeidbaren Wartezeiten, wenn Nutzereingaben unmittelbar vor komplexen Anfragen erfolgen müssen. In solchen Fällen muss ein Weg gefunden werden, gleichzeitig zu kommunizieren, dass die Daten in Bearbeitung sind und die Nutzer von doppelten oder unzulässigen Aktionen abzuhalten. Eine bekannte und einfache Möglichkeit, die erste Anforderung umzusetzen, wäre eine Ladeanimation. Durch beispielsweise einen sich kontinuierlich drehenden Kreis wird deutlich, dass gerade etwas bearbeitet wird. Eine ähnliche Animation wäre das sogenannte Skeleton, mit dem das Laden von Plots angezeigt werden kann. Beim Editieren von Tabellen durch das Backend sollte die Anwendung jedoch nicht einfach durch eine Animation ersetzt werden, sondern weiterhin, wenn auch eingeschränkt, bedienbar bleiben. Dies kann durch Promises und die asynchrone Architektur von JavaScript erreicht werden. Während asynchron auf die Antwort gewartet wird, können weitere Anfragen der User in den Callstack gelangen. Um jedoch zu verhindern, dass Anfragen doppelt gestellt werden oder durch Race-Conditions veraltete Daten vorliegen, muss die Bearbeitung der entsprechenden Daten durch die Nutzer für diese Zeit deaktiviert werden. Schließlich muss auch kommuniziert werden, wenn die Anfrage durchgeführt und die Daten angepasst wurden. Bei komplexeren Anfragen könnten die Nutzer durch eine Meldung am Rand des Fensters informiert werden. Bei weniger komplexen Berechnungen würde ein kurzer Wechsel von der Ladeanimation zu den angeforderten Daten oder einem Häkchen ausreichen.

Konkret sollte die Bearbeitung der CAD-Tabelle, während *Clear CAD Data* sowie die CAD-, Prozess- und Verbindungsdaten, während *Select Merge Preprocess Data* kurzzeitig unterbunden werden. Zusätzlich sollte für letztere Option eine Ladeanimation für die merged Tabelle erzeugt werden. Nachdem die merged Tabelle angezeigt werden kann, können für die beiden folgenden Diagramme Skeletons generiert werden. Nach Auswahl der wichtigen Spalten kann ein Skeleton für das Dendrogramm zusammen mit einer Eingabemöglichkeit für die maximalen Clustermengen erscheinen. Für die folgenden Diagramme werden erneut Skeletons benötigt, bis diese berechnet und gesendet sind. Zuletzt, nach Auswahl der endgültigen Parameter für die Cluster, kann eine Ladeanimation, wie beispielsweise ein sich drehender Kreis, angezeigt werden. Wenn die Ergebnisse der Anfragen eintreffen, wird eine Meldung und ein Link zu den entsprechenden Tabellen angezeigt.

5 Abschluss

Für das Forschungsprojekt PrEvelOp wurde durch den FIR e. V. eine Open-Source-Bibliothek in Python entwickelt, um Ähnlichkeiten in der Produktion von Bauteilen mit Funktionen des Machine Learnings zu erkennen. Damit soll kleinen und mittleren Unternehmen geholfen werden, die Varianz in den Produktionsprozessen zu reduzieren, ohne dass Produktvarianten gestrichen werden müssen. Um diese Funktionen zu Test- und zu Demonstrationszwecken übersichtlich zu verteilen, wurde ein Frontend mit Node.Js geschrieben. Dieses hilft auch Personen mit nur geringer Erfahrung in den Bereichen Datascience und Machine Learning, die Funktionen zur Ähnlichkeitsanalyse zu bedienen. Die Aufgabe dieser Arbeit bestand darin, eine Schnittstelle zwischen den Python-Funktionen und dem Node.Js-Frontend zu entwickeln und diese in beide Prozesse zu integrieren. Diese Schnittstelle sollte in der Lage sein große Datenmengen schnell, effizient und vor allem zuverlässig zu übertragen.

5.1 Zusammenfassung der Ergebnisse

Die erarbeiteten Ergebnisse lassen sich anhand der Struktur der Arbeit zusammenfassen. Zunächst wurden die zugrundeliegenden Technologien ausgewählt. Die Verwendung von HTTP bietet viele Vorteile, wie die feste automatische Fehlerverarbeitung und die große Abdeckung an HTTP-fähigen Geräten. Durch die Verwendung bestimmter Anforderungen der REST-Architektur, wird eine saubere Art der Programmierung erzwungen, die hilft, die API für Erweiterungen offen zu halten. Darüber hinaus hat eine Client-Server-Architektur den positiven Nebeneffekt, dass die API von allen HTTP- und JSON-fähigen Frontends angesprochen werden kann. Ein Vorteil von JSON, ist der geringe Overhead und damit eine relativ geringe Größe. Außerdem kann JSON schnell in JavaScript und verschiedenen Python-Bibliotheken serialisiert werden. Der Engpass, der durch die Kodierung und den Versand von großen Datenmengen entsteht, wird somit möglichst minimiert. Schließlich wurde FastAPI als Framework ausgewählt, da es JSON schnell und direkt übersetzen kann, was die Datenübertragung erheblich erleichtert. Dabei wird die Struktur des JSON-Objektes sehr genau überprüft, wodurch Fehler vermieden werden. FastAPI weist zudem, wie der Name bereits aussagt, eine hohe Performance auf. Diese wird durch asynchrone Funktionen und den Uvicorn-Server, mit dem die API gestartet wird erreicht. Zusammen sorgen diese Technologien für ein robustes, erweiterbares und vor allem schnelles Backend.

Für den Aufbau der API, wurden einige Architekturanforderungen umgesetzt. So wurde für die Ein- und Ausgabe ein streng definiertes und geprüftes JSON-Format gewählt. Dadurch sollte möglichst wenig zusätzlicher Overhead erzeugt und gleichzeitig Eingabefehler vermieden werden. Um in einer asynchronen Serverumgebung mehrere Anfragen gleichzeitig verarbeiten zu können, muss die Verarbeitung der Anfragen asynchron erfolgen. Um dabei Redundanzen im Code zu vermeiden, erfolgt die Erzeugung asynchroner Pfade in einem Software-Pattern.

Die REST-Architektur sieht eine Trennung der Verantwortlichkeiten vor. Diese wurde durch eine Datei- und Ordnerstruktur umgesetzt, die den Code zunächst nach der Funktion innerhalb der API und dann nach der Funktion außerhalb der API trennt. Durch diese Architektur wurde die API wartbar und widerstandsfähig gegen mögliche Eingabefehler konzipiert und umgesetzt.

Damit die Funktionen tatsächlich auch angesprochen werden können, muss auch die Kommunikation mit der API in das Frontend integriert werden. Um dabei Wartezeiten zu vermeiden, wurde eine geschickte Reihenfolge definiert, bei der die Anfragen, sofern möglich, durch JavaScript Promises im Hintergrund ausgeführt werden. So können Daten unbemerkt berechnet werden, während die Nutzer Eingaben für die nächste Anfrage vornehmen. Ist eine Bearbeitung im Hintergrund nicht möglich, da noch auf Nutzereingaben gewartet werden muss, werden entsprechende Ladeanimationen eingesetzt. So wird ein rotierender Kreis für größere Datenmengen und Skeletons für Bilder verwendet. Zusätzlich wird die Bedienung dahingehend eingeschränkt, dass während der Wartezeit keine Anfragen erneut gesendet werden können. Damit ist nun eine effiziente und nutzerfreundliche Kommunikation zwischen Machine Learning Verfahren und einem Node.Js Frontend möglich.

5.2 Ausblick

Die Implementierung der Schnittstelle und die Konzeption und Integration der Kommunikation ermöglichen eine einfache Weiterentwicklung vieler Aspekte des Forschungsprojektes. Insbesondere das Frontend profitiert von der Anbindung an die berechnenden Funktionen. Dieses kann nun, mithilfe der Anfragen, einen visuellen Aufbau aus den tatsächlich berechneten Werten ermitteln. Dadurch kann das Frontend schneller programmiert und auch unter realen Bedingungen getestet werden, ohne dass dafür Beispieldaten kopiert werden müssen. Auch für die im FIR e. V. entwickelte Bibliothek bringt die API Vorteile. So können zum Beispiel die im Backend generierten Plots auf ihre Lesbarkeit im Frontend überprüft und gegebenenfalls angepasst werden. Außerdem kann die Bibliothek nun nicht nur über Python, sondern auch HTTP genutzt werden. Dies und die durch FastAPI automatisch generierte Dokumentation, ermöglichen die Entwicklung von vielen unterschiedlichen und speziell angepassten Frontends zur Nutzung der Schnittstelle.

Somit bietet diese Schnittstelle auch Fachfremden einen einfachen Zugang zu wertvollen Machine Learning Funktionen. Durch die so ermöglichte Kombination von Expertenwissen kann die Wirtschaftlichkeit in der Produktion gesteigert werden.

A Literaturverzeichnis

[Bra14] Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159, March 2014.

[Ele24] Introduction — electron, 21.12.2024.

[Eri24] Jeffrey Erickson. What is json? *Oracle*, 04.04.2024.

[Gru18] Wilfried Grupe. *XML: Grundlagen — Technologien— Validierung — Auswertung.* mitp Professional. mitp, Frechen, 2018 edition, 2018.

[Lat23] Malhar Lathkar. *High-Performance Web Apps with FastAPI : The Asynchronous Web Framework Based on Modern Python / by Malhar Lathkar.* Apress, Berkeley, CA, 1st ed. 2023 edition, 2023.

[LCCY12] Boci Lin, Yan Chen, Xu Chen, and Yingying Yu. Comparison between json and xml in applications based on ajax. In *2012 International Conference on Computer Science and Service System*, pages 1174–1177. IEEE, 2012.

[Rel19] Kunal Relan. *Building REST APIs with Flask : Create Python Web Services with MySQL / by Kunal Relan.* Apress, Berkeley, CA, 1st ed. 2019 edition, 2019.

[Rub17] Daniel Rubio. *Beginning Django : Web Application Development and Deployment with Python / by Daniel Rubio.* Apress, Berkeley, CA, 2017.

[Sev12] C. Severance. Discovering javascript object notation. *Computer (Long Beach, Calif.)*, 45(4):6–8, 2012.

[Wöl23] Nicole Wölfel. Websockets: Technischer einblick und performance-vergleich. *Computer Science Blog @ HdM Stuttgart*, 01.02.2023.

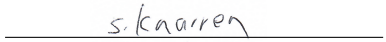
B Tabellenverzeichnis

C Abbildungsverzeichnis

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit mit dem Titel „Implementierung einer Schnittstelle zwischen Python und Node.Js zur bidirektionalen Übergabe von großen Datenmengen“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführung, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war. Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzubewahren und auf Verlangen dem Prüfungsamt des Fachbereiches Medizintechnik und Technomathematik auszuhändigen.

Name: Sasha Knarren
Aachen, den 22. Dezember 2024



Sasha Knarren