Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik
Department of Computer Science

Bachelorarbeit

im Bachelor-Studiengang Wirtschaftsinformatik

Entwicklung einer Schnittstelle für die Anbindung von austauschbaren Datenquellen an KI-Algorithmen

von

Laurenz Anton Dilba

Erstprüfer:	Prof. Dr. Matthias Bertram
Zweitprüfer:	Prof. Dr. Wolfgang Heiden
Unternehmen:	CONET Solutions GmbH
Eingereicht am:	6. Dezember 2022

Erklärung

Hiermit erkläre ich wahrheitsgemäß, dass ich den vorliegenden Bericht selbst angefertigt habe. Der Bericht gibt die tatsächlich durchgeführten Arbeiten wieder. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht. Vertrauliche Informationen sind nicht enthalten.

Datum	Unterschrift Studierender	Unterschrift Betreuer

Inhaltsverzeichnis

Abkürzungsverzeichnis

AJAX Asynchronous JavaScript and XML
API Application Programming Interface
HTML Hypertext Markup Language
KI Künstliche Intelligenz
UI User Interface
BL Business Logic
RAM Read-Access Memory
RDBMS relationalen Datenbankmanagementsystemen
HTTP Hypertext Transfer Protocol
JSON JavaScript Object Notation
AMQP Advanced Message Queuing Protocol
BERT Bidirectional Encoder Representations from Transformers
ID Identification
VM Virtuelle Maschine
URL Uniform Resource Locator
REST Representational State Transfer

1 Einleitung

text

1.1 Motivation und Hintergrund

text

1.2 Problemstellung

text

1.3 Aufbau

text

2 Grundlagen

2.1 Python API mit Flask

Python ist eine um 1991 von Guido van Rossum entwickelte Programmiersprache. Bei der Entwicklung von Python wurde ein besonderer Fokus auf die Lesbarkeit von Code gesetzt. Dank der simplifizierten Syntax im Vergleich zu anderen höheren Programmiersprachen wie Java oder C#, ist Python auch in Bereichen, wie in der Mathematik oder der Wissenschaft ein häufig genutztes Werkzeug. Python bietet ebenfalls die Möglichkeit, von anderen Entwicklern bereitgestellte Bibliotheken in das eigene Projekt zu integrieren.[1]

Flask ist eine der verfügbaren Bibliotheken, die ein Framework für die Implementierung eines Webbasierten Application Programming Interface (API) bereitstellt. Eine API dient dazu, Funktionen und Routen zu definieren, um die Kommunikation zwischen dem Frontend und dem Backend herzustellen. Das Flask Framework ist im Gegensatz zu anderen Frameworks sehr klein. Dies ermöglicht ein schnelles aufsetzen und entwickeln. Da Flask nur die nötigsten Grundlagen für eine API mitliefert, ist der Code besser lesbar und damit für andere Entwickler besser wartbar.[2]

Die Flask API wird für die Anbindung des Frontends an die Datenbank, sowie die Anbindung an die Kommunikationsschnittstelle von RabbitMQ verwendet. Sie nimmt die Daten oder die Eingaben des Nutzers entgegen und vermittelt sie an den richtigen Dienst, damit sie von einer KI-Schnittstelle ausgewertet werden können. Anschließend kann die API angefragt werden, ob es bereits Antworten von einer Künstliche Intelligenz (KI) zu der vorher geschickten Anfrage gab. Falls die API die Auswertung der KI erhalten hat, wird diese ans Frontend geschickt, um sie dort anzeigen zu können.

2.2 Angular Frontend

Eine grundlegende Website wird klassisch mit Hypertext Markup Language (HTML) und JavaScript erstellt. Um eine moderne Website zu entwickeln, die ihren Inhalt nicht beim ersten Aufrufen lädt, sondern erst dann, wenn er benötigt wird, müssen Konzepte wie Asynchronous JavaScript and XML (AJAX) verwendet werden. Angular ist ein von Google gebaut und gepflegtes Open Source Framework, welches das Entwickeln von komplexen webbasierten Anwendungen vereinfachen soll. Angular bietet im Gegensatz zu anderen Webframeworks wie React und Vue.js eine vollumfängliche Bibliothek, mit der nahezu alle Aspekte in der Web Entwicklung abgedeckt werden können.[3]

In Angular wird die Programmiersprache TypeScript verwendet. Diese ist eine Erweiterung der Programmiersprache JavaScript und implementiert Konzepte wie feste Typisierung von Variablen. Weitere Konzepte wie Dependency Injection oder die Trennung von Business Logic (BL) und User Interface (UI) ermöglichen eine schnelle Entwicklung von komplexen Systemen.

Das Frontend wird für die Ein- und Ausgabe der Daten verwendet. Der Nutzer kann auf der Webseite seine Suchanfrage in ein Textfeld schreiben und anschließend auf den Server

[1] Josheph, 2021.
[2] Grinberg, 2018.
[3] Moiseev u. a., 2018.

hochladen. Im nächsten Schritt wird die Möglichkeit bereitgestellt, die eingegeben Daten automatisiert zu bearbeiten und zu manipulieren. Im gleichen Zug wird die Eingabe des Nutzers in ein für die KI verständliches Format konvertiert. Im letzten Schritt kann der Nutzer die Anfrage an das Backend schicken, dass mit der Analyse der Eingabe begonnen werden soll. Das Frontend fängt daraufhin an beim Backend in regelmäßigen Abständen nach Antworten der KI zu fragen. Wenn Antworten vorhanden sind, können diese in einer Liste visualisiert werden.

2.3 Redis API Cache

Redis ist eine In-Memory Key-Value Datenbank. Im Gegensatz zu relationalen Datenbankmanagementsystemen (RDBMS) wie MySQL oder PostgreSQL werden in Redis keine festen Tabellenstrukturen hinterlegt. Redis gehört damit zur Kategorie der NoSQL Datenbanken (Not Only SQL). Key-Value Stores sind kein Ersatz für eine relationale Datenbank, bieten aber für bestimmte Bereiche große Vorteile. Durch das Fehlen von komplexen Strukturen innerhalb der Datenbank, kann Redis Anfragen weitaus schneller als andere Datenbanksysteme bearbeiten. Da Redis im Read-Access Memory (RAM) ausgeführt wird, werden die Daten grundsätzlich nicht persistent gespeichert. ACID (Atomicity, Consistency, Durability and Isolation) Konformität wird mit Redis ebenfalls nicht gewährleistet. Für den Einsatzzweck als Cache in einer Cloud Umgebung ist Redis allerdings sehr gut geeignet.[4]

Innerhalb des Redis Key-Value Stores werden alle relevanten Daten gespeichert, die ein Nutzer während seiner Benutzung der Software produziert. Dort werden ebenfalls die Zwischenergebnisse abgespeichert, die die KI während der Analyse erstellt.

2.4 MySQL Datenbank für Services und Logs

MySQL ist ein um 1995 erschienenes Open-Source RDBMS. MySQL ist eines weitverbreitetsten und schnellsten Datenbanksysteme in seiner Kategorie.[5]

In relationalen Datenbanken werden Daten strukturiert in Tabellenform abgespeichert. Einzelne Tabellen können Verlinkungen und Referenzen auf andere Tabellen haben, damit die Zusammengehörigkeit der Daten beschrieben werden kann, ohne Daten redundant speichern zu müssen. In MySQL, wie auch anderen RDBMS, werden Tabellenstrukturen und Daten persistent abgespeichert. In-Memory Datenbanken wie Redis können Daten über Umwege auch persistent speichern, jedoch müssen dafür größere Anpassung an der Konfiguration von Redis vorgenommen werden.

Das RDBMS MySQL wird unter Anderem für die Speicherung der Logs, die der Flask Server während der Verarbeitung von Requests oder Nachrichten an die KI produziert, verwendet. Ein weiterer Einsatzzweck der MySQL Datenbank ist die Speicherung der im System registrieren KI-Services. Ein Dienst kann über die Flask API im System registriert oder deregistriert werden. Das Frontend kann sich im Anschluss eine Auflistung der verfügbaren Services vom Backend ziehen.

[4] Paksula, 2010.
[5] DuBois, 2008.

2.5 Kommunikation mit RabbitMQ

Damit eine Kommunikation zwischen unabhängigen Programmen möglich wird, muss es einen Zwischendienst geben, der die Nachrichten von von Programm zum anderen transportiert. Beider Kommunikation zwischen einer Website und einer API wird das Hypertext Transfer Protocol (HTTP) verwendet. Dieses stellt sicher, dass die Information, ob die Nachrichten am anderen Ende angekommen sind, vorhanden sind. Sollte eine Nachricht nicht angekommen sein, hat der Absender die Möglichkeit die Nachricht erneut zu schicken. Problematisch wird diese Herangehensweise, wenn die Antwortzeit sehr lang wird oder ungewiss ist, ob überhaupt eine Antwort kommen wird.

RabbitMQ ist ein eine nachrichtenorientierte Middleware, die die Kommunikation zwischen zwei oder mehreren Programmen durch das Advanced Message Queuing Protocol (AMQP) ermöglicht. Im Gegensatz zu einer direkten Kommunikation zwischen Client und Server wie bei HTTP, wird in RabbitMQ eine Queue implementiert, in der alle Anfragen gesammelt werden. Jeder Client kann Nachrichten in die Queue reinschreiben. Diese Nachrichten werden dort so lange gespeichert, bis sie von einem Dienst ausgelesen werden. Durch diese Herangehensweise wird eine asynchrone Kommunikation zwischen Client und Server ermöglicht. Da RabbitMQ frei von den Handshakes des HTTP ist, sind die Schreib- und Lesezeiten deutlich schneller.[6]

Die Middleware RabbitMQ wird für die Kommunikation zwischen der Flask API und den KI-Services genutzt. Der im Frontend vom Nutzer eingegebene Text-Input wird an die Flask API geschickt. Die Flask API modifiziert den Text im Anschluss so, dass es mittels der JavaScript Object Notation (JSON) über den RabbitMQ Service in die Queue geschrieben werden kann. Jeder KI-Service hat eine Queue einprogrammiert, aus der die Nachrichten ausgelesen werden. Diese Nachrichten können dann verarbeitet und im Anschluss in eine Response-Queue geschrieben werden. Das Flask Backend kann diese Response-Queue auslesen und die einzelnen Antworten dann zusammenbauen.

2.6 KI-Service

Die KI-Services sind alleinstehende Programme, die die Aufgabe haben, Nachrichten anzunehmen, sie zu transformieren, zu analysieren und anschließend ein oder mehrere Ergebnisse zurückzugeben.

Um die Nachrichten empfangen und die Ergebnisse zurücksenden zu können, muss in jedem Service eine AMQP Verbindung zu RabbitMQ hergestellt werden.

Im Prototypen zur Anbindung von austauschbaren Datenquellen an KI-Algorithmen wurde ein Service zur Textähnlichkeitssuche implementiert. Dieser nimmt nutzt das Bidirectional Encoder Representations from Transformers (BERT) Modell von Google. Beim Starten des Services werden alle Einträge in einer Elasticsearch Datenbank mithilfe der künstlichen Intelligenz analysiert und in semantische Vektoren konvertiert. Der Service kann anschließend vom Nutzer eingegebene Anfragen mit dem gleichen BERT Modell analysieren und den daraus entstandenen Vektor mit den Vektoren in der Datenbank ab-

[6] Ionescu, 2015.

gleichen. Nach einem erfolgreichen Suchdurchlauf werden die semantisch ähnlichsten Einträge über RabbitMQ wieder an das Backend zurückgegeben.

2.7 Logs visualisieren in Grafana

Grafana ist ein von Torkel Ödegaard in 2014 entwickeltes Open-Source Datenvisualisierungsprogramm. Grafana kann zeitbasierte Daten in verschiedenen Arten von Grafen und Diagrammen anzeigen.[7]

Eines der möglichen Panels für ein Dashbaord ist das Log-Panel. Dort werden die Log Nachrichten aus einer Datenbank angezeigt und mit einer Farbe, abhängig vom Schweregrad markiert. Als Datenquelle können unter Anderem zeitbasierte Datenbanken wie InfluxDB und Prometheus oder RDBMS wie MySQL verwendet werden.

Im implementierten Prototypen wurde eine MySQL verwendet, in der die zu Loggende Nachricht, der Schweregrad, ein Zeitstempel und die User Identification (ID) gespeichert werden. Diese Daten werden verwendet, um die Logs im Log-Panel von Grafana chronologisch anzeigen zu lassen.

2.8 Deployment mit Docker

Docker ist eine Software zur Virtualisierung von Containern. Ein Container beschreibt eine in sich geschlossene Umgebung, in der ein Programm ausgeführt werden kann. Alle benötigten Dateien, Parameter und Umgebungsvariablen werden beim Starten des Containers mitgegeben. Damit kann sichergestellt werden, dass ein Programm, welches innerhalb eines Docker Containers ausgeführt wird, sich in jeder Umgebung gleich verhält. Eine Unabhängigkeit vom Host-Betriebsystem wird dadurch gewährleistet. Im Gegensatz zu einer Virtuelle Maschine (VM) muss für die Ausführung eines Docker Containers kein komplettes Betriebssystem virtualisiert werden. Das Hochfahren einzelner Container ist deutlich schneller und ressourcenschonender als die Implementierung einzelner VMs.[8]

Des Weiteren können über das Docker Compose Plugin mehrere Container gleichzeitig hochgefahren werden, sodass mit einer einzigen Kommandozeileneingabe eine komplette Softwarearchitektur hochgefahren werden kann.

Docker wird für das Deployment der einzelnen Komponenten des Prototypens verwendet. Für Redis, MySQL, RabbitMQ, Grafana und Elasticsearch können die benötigten Images, die eine Bauanleitung darstellen aus dem Docker Hub heruntergeladen und genutzt werden. In einem Docker Image sind auch alle für die Ausführung des Programms benötigten Dateien gepackt. Docker Hub ist eine Plattform zur Verteilung von offiziellen Docker Images, von der automatisch alle Images runtergeladen werden, die lokal nicht vorhanden sind.

Für das Angular Frontend und das Flask Backend müssen die Images erst manuell gebaut werden, bevor sie als Container gestartet werden können. Dafür bietet die Docker sogenannte Dockerfiles an, in der die benötigten Konfigurationen hinterlegt werden können.

[7] Chakraborty u. a., 2021.
[8] Anderson, 2015.

3 Methodik

text

3.1 Design Science Research

text[9]

3.2 Evaluationsmethode

text

[9] Frauchiger, 2017.

4 Projektergebnisse

In diesem Kapitel wird die prototypische Implementierung der Schnittstelle für die Anbindung von austauschbaren Datenquellen an KI-Algorithmen beschrieben.

4.1 Softwarearchitektur

Ein grundlegender Dienst, der Daten mit einer KI verbindet, kann mithilfe eines einzigen Python-Scripts erstellt werden. Die Herausforderung an einer praxistauglichen Anwendung, die gleichzeitig von mehreren Usern genutzt werden kann, liegt im Architekturdesign der Software. Eine praxistaugliche Anwendung muss neben den funktionalen Anforderungen auch noch weitere nicht funktionale Anforderungen erfüllen. Die drei wichtigsten nicht funktionalen Anforderungen sind Performance, Skalierbarkeit und Verfügbarkeit. Alle drei Anforderungen können mit einem lokal ausgeführten Skript nicht erfüllt werden.

Damit Nutzer mit der Software interagieren können, wird ein Frontend benötigt. Ein zentral gehostetes, webbasiertes Frontend kann von einem Nutzer über eine einfache Uniform Resource Locator (URL) im Webbrowser aufgerufen werden. Für die anzuzeigenden Daten im Frontend wird eine Verbindung zum Backend benötigt. Diese wird über eine HTTP Verbindung zur mit Flask gehosteten Representational State Transfer (REST) API bereitgestellt.

Um die Anforderung der Skalierbarkeit erfüllen zu können, ist die REST-API komplett zustandslos implementiert worden. Eine API ohne Zustände speichert keine Zwischenstände zu den Anfragen einzelner Nutzer. Bei jeder Anfrage an die API müssen alle Informationen im Request bereitgestellt werden, die die API zum bearbeiten der Anfrage benötigt. Dies bietet die Möglichkeit bei steigender Nutzerzahl mehrere parallel betriebene Instanzen der API hochzufahren. Dadurch ist eine horizontale Skalierung gewährleistet. Horizontal skalierbare Instanzen innerhalb der Software Architektur sind in Abbildung 1 mit zwei hintereinander gestapelten Rechtecken visualisiert.

Da die Kommunikation zwischen dem Frontend, der API und den KI-Services asynchron läuft, muss das Flask Backend trotz seiner Zustandslosigkeit Transformationsanleitungen und Ergebnisse der KI-Services zwischenspeichern, bis sie im Frontend benötigt werden. Um die Performanceanforderungen erfüllen zu können, können nicht alle Zwischenstände in einer MySQL Datenbank gespeichert werden. Die Lese- und Schreibgeschwindigkeit kann bei steigender Nutzerzahl problematisch werden. Um dem Entgegenzuwirken wird ein Redis Key-Value Store als Cache betrieben. Die zwischengespeicherten Daten werden nach dem ersten Aufruf wieder gelöscht, weswegen eine persistente Speicherung nicht notwendig ist. In-Memory Datenbanken speichern und führen ihre Queries direkt im RAM aus, wodurch Anfragen im Vergleich zu einer MySQL Datenbank deutlich schneller ausgeführt werden.

Im Flask Backend werden alle Routen und die meisten Funktionen abgekapselt in einem Funktion Wrapper ausgeführt. Dieser fungiert als eine Art Sandbox, in der auftretende Fehler nicht zum Programmabsturz führen, sondern behandelt und geloggt werden können. Alle Logs werden persistent in einer MySQL Datenbank gespeichert. Mit dem Dienst Grafana können diese Logs angezeigt werden.

Die Laufzeit von KI-Services kann sehr stark vom verwendeten KI-Modell, der zu durchsuchenden Datenmenge, wie auch der vom Nutzer gesendeten Eingabe abhängen. Bei einer synchronen Kommunikation zwischen dem Flask Backend und dem Service können sehr lange Wartezeiten entstehen. Wenn der KI-Service ebenfalls eine REST-Schnittstelle implementieren würde, könnten es bei einem HTTP Request zum Timeout der Anfrage führen. Aufgrund der schwanken Laufzeit muss eine asynchrone Kommunikationsstruktur, wie RabbitMQ mit dem AMQP implementiert werden.

Die einzelnen Services können mit einem Eintrag in der MySQL Datenbank registriert werden. Für die Registrierung muss lediglich der Name und der im Frontend anzuzeigende Name des Services hinterlegt werden. Die Registrierung eines Dienstes kann durch den Aufruf einer Route in der API durchgeführt werden.

Der im Prototypen implementierte KI-Service nutzt das BERT Modell von Google zum konvertieren der Nutzereingaben in semantische Vektoren. Es wird ebenfalls eine Elasticsearch Datenbank betrieben, in der alle zu Durchsuchenden Einträge gespeichert sind. Im Gegensatz zu einer MySQL Datenbank, kann in einer Elasticsearch Datenbank zu jedem Eintrag ein semantischer Vektor gespeichert werden. Der KI-Service kann mithilfe der Kosinusähnlichkeitssuche den semantischen Vektor der Eingabe mit den Vektoren der Datenbank vergleichen und so die semantisch ähnlichstes Texte herausfiltern. Die gefunden Einträge werden über RabbitMQ im Anschluss wieder an das Flask Backend geschickt, damit sie dort vom Frontend ausgelesen werden können.

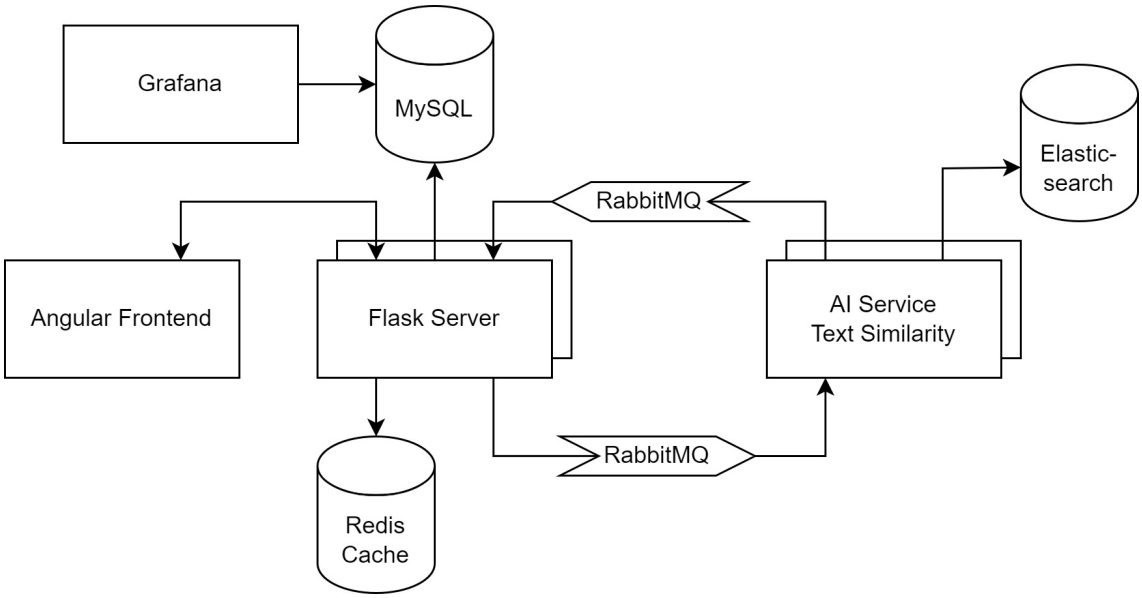

Abbildung 1: Architekturdiagramm

4.2 REST-API mit Flask

text

4.2.1 Aufbau und Implementierung der REST-API

text

4.2.2 Nutzeridentifizierung mit JWT

text

4.2.3 Caching mit Redis Datenbank

text

4.2.4 Fehlerbehandlung

text

4.2.5 Event Logging

text

4.3 Visualisierung der Logs in Grafana

text

4.4 Webseite mit Angular

text

4.4.1 Aufbau des User Interfaces

text

4.4.2 Funktionen der Komponenten

text

4.4.3 Kommunikation zur API

text

4.5 Kommunikation zwischen API und Services mit RabbitMQ

text

4.5.1 RabbitMQ vs. REST-API

text

4.6 Implementierung des KI-Services

text

4.6.1 Dynamische Registrierung neuer Services

text

4.6.2 Interpretation der Eingabe mit BERT

text

4.6.3 Cosinusähnlichkeitssuche in Elastic Search

text

5 Evaluation

text

5.1 Performanceanalyse

text

5.2 Skalierbarkeit

text

5.3 Ergebnisse des Code-Reviews

text

6 Fazit

6.1 Fazit

6.2 Einschränkungen

6.3 Ausblick

7 Literaturverzeichnis

ANDERSON, C., 2015. Docker [software engineering]. *Ieee Software*. Jg. 32, Nr. 3, S. 102–c3.

CHAKRABORTY, M.; KUNDAN, A.P., 2021. Grafana. In: *Monitoring Cloud-Native Applications*. Springer, S. 187–240.

DUBOIS, P., 2008. *MySQL*. Pearson Education.

FRAUCHIGER, D., 2017. Anwendungen von Design Science Research in der Praxis. In: *Wirtschaftsinformatik in Theorie und Praxis*. Wiesbaden: Springer Fachmedien Wiesbaden, S. 107–118.

GRINBERG, M., 2018. *Flask web development: developing web applications with python*. O'Reilly Media, Inc.

IONESCU, V.M., 2015. The analysis of the performance of RabbitMQ and ActiveMQ. In: *2015 14th RoEduNet International Conference-Networking in Education and Research (RoEduNet NER)*. IEEE, S. 132–137.

JOSHEPH, T., 2021. Python. *Python Releases for Windows*. Jg. 24.

MOISEEV, A.; FAIN, Y., 2018. *Angular Development with TypeScript*. Simon und Schuster.

PAKSULA, M., 2010. Persisting objects in redis key-value database. *University of Helsinki, Department of Computer Science*. Jg. 27.