Information Retrieval – Dokumentation

bei Sascha Szott

im Sommersemester 2025

Vorgelegt von:

Linus Breitenberger
lb205@hdm-stuttgart.de
Matrikelnummer: 43789

Hochschule der Medien Stuttgart

26. Juli 2025

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation und Projektziele

Das vorliegende Projekt entstand im Rahmen des Moduls „Information Retrieval“ und diente der praktischen Anwendung der im Kurs vermittelten theoretischen Grundlagen. Die zentrale Aufgabenstellung bestand darin, eine eigenständige Suchanwendung auf Basis der etablierten Open-Source-Technologie Apache Solr zu konzipieren und umzusetzen.

Für die Umsetzung wurde die Pokémon API (pokeAPI.co) als Datenquelle ausgewählt. Auf dieser Grundlage entstand die Anwendung „Solr Pokédex“. Im Rahmen des Projekts wurden sowohl übergeordnete Ziele als auch konkrete funktionale Anforderungen definiert. Ein zentrales Ziel war die Entwicklung einer vollständigen Indexierungspipeline, die in der Lage ist, Daten automatisiert von der Quelle abzurufen, zu bereinigen und für Solr entsprechend aufzubereiten. Darüber hinaus sollte ein robustes und erweiterbares Solr-Schema entworfen werden, das die Struktur und Eigenschaften des gewählten Datensatzes sinnvoll abbildet. Für eine realistische und aussagekräftige Suchumgebung war die Indexierung eines relevanten Datenbestands vorgesehen, der mindestens 1000 einzigartige Einträge umfasst. Abgerundet wurde das Projektziel durch die Entwicklung einer simplen Weboberfläche, welche eine einfache und benutzerfreundliche Interaktion mit der Suchmaschine ermöglichen sollte.

1.2 Funktionale Kernanforderungen

Zur Erreichung der genannten Ziele musste die Anwendung bestimmte funktionale Anforderungen erfüllen. Dazu gehörte die Unterstützung verschiedener Suchanfragetypen, darunter eine klassische Keywordsuche über zentrale Textfelder, eine Phrasensuche zur gezielten Suche nach exakten Wortfolgen, eine Wildcardsuche mit Platzhaltern für flexible Suchmuster sowie eine facettierte Suche, die das Filtern von Ergebnissen nach Kriterien wie Pokémon-Typ oder Generation ermöglicht.

Um die Benutzerfreundlichkeit weiter zu erhöhen, wurden außerdem Funktionen zur Fehlerbehandlung und Ähnlichkeitssuche integriert. So sollte das System in der Lage sein, bei fehlerhaften Eingaben passende Korrekturvorschläge zu liefern („Meinten Sie…?“) und zusätzlich thematisch verwandte Inhalte zu einem Suchergebnis anzuzeigen („More like this“).

Über die funktionalen Kernanforderungen hinaus wurden einige optionale Erweiterungen als sogenannte „Stretch Goals“ formuliert. Dazu zählte unter anderem eine Autocompletion-Funktion, die während der Eingabe bereits passende Suchvorschläge anbietet, sowie ein Highlighting-Mechanismus, der die gesuchten Begriffe direkt in der Ergebnisvorschau visuell hervorhebt.

1.3 Vorstellung der Suchmaschine Solr Pokédex

Die im Rahmen dieses Projekts entwickelte Anwendung trägt den Namen „Solr Pokédex“ und ist eine spezialisierte Suchmaschine für Pokémon. Sie bietet einen umfassenden Index, der alle 1025 Pokémon der Generationen eins bis neun abdeckt.

Jedes Pokémon wird als eigenständiges Dokument in Apache Solr gespeichert und mit einer Vielzahl von detaillierten Metadaten angereichert. Diese Daten wurden sorgfältig ausgewählt, um sowohl eine gezielte Suche nach Fakten als auch eine explorative Volltextsuche zu ermöglichen. Zu den zentralen indexierten Feldern gehören:

- **Stammdaten:** Name, Pokédex-ID, Typ(en), Generation, Größe und Gewicht.
- **Fähigkeiten und Kampfwerte:** Alle erlernbaren Fähigkeiten sowie die Basiswerte für Lebenspunkte (HP), Angriff, Verteidigung etc.
- **Beschreibender Text:** Ein separates Feld namens `flavor_text` enthält die offiziellen Beschreibungen aus den Spielen. Mit seinem größeren Textumfang bildet dieses Feld die ideale Grundlage für eine freie Volltextsuche, die über die Suche nach reinen Fakten hinausgeht.

Die Interaktion mit der Suchmaschine erfolgt über eine mit Flask entwickelte Weboberfläche, die unter `http://localhost:5000` erreichbar ist. Die gesamte Anwendung ist containerisiert und lässt sich mittels Docker Compose und einem bereitgestellten Installationsskript (`install.sh`) unkompliziert einrichten und starten.

1.4 Verwendeter Technologie-Stack

Die Architektur des „Solr Pokédex“ basiert auf einer Auswahl bewährter Open-Source-Technologien, die gezielt für ihre jeweilige Aufgabe im Projekt eingesetzt wurden. Der Stack lässt sich in die Bereiche Backend, Frontend, Datenquelle und Deployment unterteilen. Die prozentuale Verteilung der Programmiersprachen im Projekt (siehe Abbildung 1) spiegelt diese Aufteilung wider.

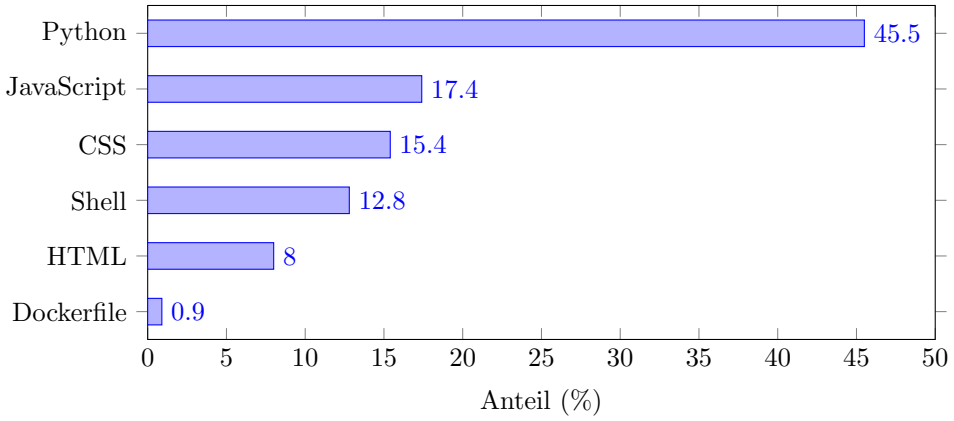

Abbildung 1: Sprachverteilung im Codebestand

Backend und Datenverarbeitung (Python, Apache Solr) Das Herzstück der Anwendung bildet das Backend, das primär in **Python** implementiert ist. Python wurde aufgrund seiner exzellenten Bibliotheken für Webentwicklung und Datenverarbeitung sowie seiner einfachen Lesbarkeit gewählt.

- **Apache Solr:** Als Suchserver-Technologie wurde Solr eingesetzt, da es in der Lehrveranstaltung als Standard vorgegeben war. Eine freie Wahl zwischen verschiedenen Suchmaschinen bestand daher nicht, auch wenn Alternativen wie OpenSearch oder Elasticsearch ebenfalls interessante Optionen gewesen wären. Dennoch überzeugt Solr durch eine hohe Performance, eine flexible Schema-Definition und eine mächtige Query-Syntax, was es zu einer geeigneten Grundlage für die Indexierung und komplexe Abfrage der Pokémon-Daten macht. Die Konfigurationen des Solr-Cores befinden sich im Verzeichnis `solr/configsets/`.
- **Flask**: Das leichtgewichtige Web-Framework Flask dient als Brücke zwischen dem Frontend und dem Solr-Server. Im Rahmen der Lehrveranstaltung wurde eine einfache Basisanwendung auf Grundlage von Flask bereitgestellt, welche ich für meine Anwendung entsprechend angepasst und erweitert habe. Die Datei `web/web_app.py` verarbeitet die HTTP-Anfragen der Benutzeroberfläche, konstruiert die entsprechenden Solr-Queries und gibt die Ergebnisse als gerenderte HTML-Seite zurück.
- **Datenakquise:**

Zu Beginn entwickelte ich ein einzelnes, umfassendes Skript namens `fetcher_v2.py`, das alle erforderlichen Funktionen zur Befüllung der Suchmaschine in sich vereinte. Mit der Zeit und der Implementierung zusätzlicher Features wuchs dieses Skript jedoch kontinuierlich an, bis es schließlich unübersichtlich und schwer wartbar wurde. Aus diesem Grund entschied ich mich für eine Refaktorierung und teilte das ursprüngliche Skript in mehrere spezialisierte Module auf:

- `main.py`: Das Hauptskript, das den gesamten Datenabfrage- und Indexierungsprozess orchestriert
- `api_client.py`: Verwaltet die Kommunikation mit der Pokemon API
- `data_processor.py`: Verarbeitet und transformiert die Pokemon-Daten für die Indexierung
- `solr_indexer.py`: Übernimmt das Setup des Solr-Schemas und die Dokumentenindexierung
- `config.py`: Enthält Konfigurationseinstellungen und das Logging-Setup

Das ursprüngliche Skript `fetcher_v2.py` fungierte als zentrale Komponente zur Befüllung der Suchmaschine und übernahm sämtliche Aufgaben von der Konfiguration des Solr-Schemas über den Abruf der Daten von der PokeAPI

bis hin zu deren Verarbeitung und finaler Indexierung. Es war so konzipiert, dass es auch auf einem frischen Solr-Core ohne manuelle Vorbereitung funktionsfähig war. Die Daten wurden systematisch bereinigt, angereichert und in eine für Solr optimierte Struktur überführt. Ein integrierter Batching-Mechanismus und eine detaillierte Protokollierung gewährleisteten dabei sowohl Effizienz als auch Transparenz im gesamten Indexierungsprozess. Eine detailliertere Beschreibung der einzelnen Komponenten und deren Funktionsweise folgt in den nachstehenden Kapiteln.

Frontend (HTML, CSS, JavaScript) Die Benutzeroberfläche wurde mit klassischen Web-Technologien realisiert, um eine einfache und reaktionsschnelle User Experience zu gewährleisten.

- **HTML und CSS:** Die Struktur der Webseite ist in der Template-Datei `web/templates/index.html` definiert. Das Styling erfolgt über eine separate CSS-Datei (`web/static/style.css`).
- **JavaScript:** Für die dynamische Interaktivität auf der Client-Seite kommt pures JavaScript `web/static/js/main.js` zum Einsatz. Eine zentrale Funktion ist die Darstellung der Detailansicht eines Pokémon. Bei einem Klick auf ein Suchergebnis wird kein neuer Seitenaufruf ausgelöst. Stattdessen wird ein modales Fenster (Modal) über die bestehende Seite gelegt, das die Detailinformationen des ausgewählten Pokémon anzeigt. Dieser Ansatz verbessert die Benutzererfahrung, da der Kontext der Suchergebnisse erhalten bleibt.

Datenquelle Als externe Datengrundlage dient die **Pokémon API (pokeAPI.co)**. Sie bietet eine umfangreiche und gut strukturierte Sammlung von Pokémon-Daten im JSON-Format, die sich ideal für die Verarbeitung und Indexierung eignete.

Deployment und Automatisierung (Docker, Shell) Um eine einfache und reproduzierbare Einrichtung der Anwendung zu garantieren, wurde auf Containerisierung und Skripting gesetzt.

- **Docker und Docker Compose:** Die gesamte Anwendung, inklusive des Solr-Servers und der Flask-App, wird durch die Datei `docker-compose.yml` als Multi-Container-Anwendung definiert. Dies isoliert die Komponenten und vereinfacht das Deployment erheblich.
- **Shell-Skripting:** Das Skript `install.sh` automatisiert den gesamten Setup-Prozess: Es richtet die Python-Umgebung ein, installiert Abhängigkeiten, startet die Docker-Container und initiiert die erstmalige Datenindexierung.

2 Datengrundlage und Datenakquise

2.1 Die Datenquelle: Pokémon-API (pokeAPI.co)

Beschreibung der API, ihrer Struktur und warum sie für das Projekt geeignet ist.

2.2 Analyse der Datenstruktur und relevanter Endpunkte

Welche Endpunkte (z.B. /pokemon, /type) haben Sie genutzt? Wie sieht das JSON-Format aus?

2.3 Datenakquise und Vorverarbeitung mit `fetcher_v2.py`

2.4 Automatisierter Datenabruf via Python-Skript

Erläutern Sie die Funktionsweise des Skripts.

2.5 Datenbereinigung und Transformation für die Indexierung

Wie haben Sie die JSON-Daten aufbereitet, damit Solr sie verarbeiten kann? (z.B. Flachen von Strukturen, Auswahl relevanter Felder).

2.6 Technische Besonderheiten: Implementierung eines Rate-Limits

Erwähnen Sie, dass Sie die API-Richtlinien respektieren und wie Sie das technisch umgesetzt haben.

3 Systemarchitektur und Konzeption

3.1 Überblick der containerisierten Gesamtarchitektur

Fügen Sie hier ein Diagramm ein, das das Zusammenspiel von Docker, Flask-App und Solr-Container zeigt. Beschreiben Sie die Architektur.

3.2 Entwurf des Solr-Indexschemas (`solr/configsets/`)

3.3 Definition zentraler Feldtypen

Welche Feldtypen (z.B. text_de, string, pint) haben Sie definiert und warum?

3.4 Struktur des Index: Definierte Felder

Listen Sie die wichtigsten Felder Ihres Index auf (z.B. `name`, `primary_type`, `all_abilities`, etc.) und beschreiben Sie deren Zweck.

3.5 Nutzung von `copyField` für eine übergreifende Keywordsuche

Erklären Sie, wie `copyField` funktioniert und warum Sie es für die einfache Suche eingesetzt haben.

4 Implementierung der Kernkomponenten

4.1 Indexierungspipeline

Beschreiben Sie den Prozess vom Start des Fetchers bis zum fertigen Dokument in Solr.

4.2 Entwicklung der Webanwendung mit Flask (`web/web_app.py`)

Erläutern Sie die Struktur der Flask-App: Backend-Routen und Frontend-Templates. Fügen Sie hier Screenshots der UI ein.

4.3 Implementierung der Suchfunktionalitäten

Gehen Sie auf die einzelnen Suchtypen ein, wie sie in der README beschrieben sind (Keyword, Phrase, Facetten, etc.) und wie Sie diese mit Solr-Anfragen umgesetzt haben.

4.4 Automatisierung des Setups (`install.sh`)

Erklären Sie kurz den Zweck und die Funktionsweise des Installationsskripts.

5 Evaluation und Optimierung

5.1 Funktionale Tests der implementierten Suchanfragetypen

Wie haben Sie sichergestellt, dass die Suche wie erwartet funktioniert? Zeigen Sie Beispiele.

5.2 Bewertung und Optimierung des Relevanzrankings

Haben Sie das Ranking angepasst (z.B. durch Boosting von Feldern)? Diskutieren Sie die Ergebnisse.

5.3 Diskussion der optionalen Features

Falls implementiert, beschreiben Sie hier Highlighting und Autocompletion. Falls nicht, erwähnen Sie, warum nicht und dass es Teil des Ausblicks ist.

6 Fazit und Ausblick

6.1 Zusammenfassung der Projektergebnisse

Fassen Sie die erreichten Ziele und das finale Produkt noch einmal kurz zusammen.

6.2 Reflektion der Herausforderungen und Lösungsansätze

Was waren die größten Schwierigkeiten im Projekt (z.B. Schema-Design, Daten-Parsing, Flask-Anbindung) und wie haben Sie diese gelöst?

6.3 Mögliche Erweiterungen und zukünftige Optimierungen

Welche Ideen haben Sie für die Zukunft? (z.B. weitere Datenquellen, komplexere Filter, Benutzer-Accounts, etc.)

A Auszug aus dem Solr-Schema

Fügen Sie hier relevante Teile Ihrer ‘managed-schema‘ ein, z.B. als Code-Block.

B Relevante Code-Auszüge

Zeigen Sie hier wichtige Snippets aus `fetcher_v2.py` oder `web_app.py`.

C Screenshot der Benutzeroberfläche

Ein vollseitiger Screenshot der fertigen Anwendung.

Abbildungsverzeichnis

Listings