

Anleitung Full-Stack App für Sprachübungen

Inhalt

1. Einleitung 6

1.1. Ausgangslage 6

1.2. Problemstellung 6

1.3. Idee und Relevanz 6

1.4. Zielsetzung 7

1.4.1. Nebenziel 7

1.5. Abgrenzung 8

2. Theoretische Grundlagen 9

2.1. Web-Scraping-Technologie 9

2.1.1. Web Scraping 9

2.1.2. Python 10

2.1.3. BeautifulSoup 10

2.1.4. Scrapy 10

2.1.5. Selenium 10

2.2. Datenbanktechnologie 11

2.2.1. Daten 11

2.2.2. Datenbank 11

2.2.3. MongoDB 11

2.2.4. Dokument – MongoDB 12

2.2.5. Sammlung – MongoDB 12

2.2.6. MongoDB-Compass 12

2.2.7. Prisma 12

2.2.8. JSON 13

2.3. Webtechnologien 13

2.3.1. Next.js 14 13

2.3.2. HTML 14

2.3.3. Tailwind CSS 14

2.3.4. JavaScript 14

2.3.5. Visual Studio Code 14

3. Methodisches Vorgehen 15

3.1.1. Analyse 15

3.1.2. Recherche 15

3.1.3. Umsetzung 15

3.1.4. Ergebnisse 16

3.1.5. Fazit und Reflexion 16

4. Analyse 16

4.1. Applikation 16

4.1.1. Login 16

4.1.2. Lernsession 18

4.1.3. Worterfassung 20

4.2. Datenbankstruktur 22

4.2.1. Datentypen 22

4.2.2. Lernbereich 23

4.2.3. Felder 23

4.2.4. Verwaltungsbereich 23

4.3. Anforderungen 23

4.3.1. Erweiterung des Datenbankvolumens 23

4.3.2. Hinzufügen neuer Lernkategorie 24

4.3.3. Anpassen des Lernkarteninhalts 24

5. Entwicklung / Umsetzung 24

5.1. Web-Scraping-Prozess 24

5.1.1. Skripts 25

5.1.2. Methodenauswahl 25

5.2. Datensammlung 26

5.2.1. JSON-Datei 35

5.2.2. Implementierung 36

5.2.3. Sprichwörter 36

5.3. Lernkategorie 37

5.3.1. Scraping 38

5.3.2. Frontend 38

5.3.3. Backend 39

5.4. Lernkarten 39

5.4.1. Visuelle Anpassung 40

6. Ergebnisse 40

6.1. Web-Scraping-Skripts 40

6.2. Lernkategorie 41

6.3. Lernkarte 41

7. Fazit 43

7.1. Zusammenfassung 43

7.1.1. Quellenanpassung 44

7.1.2. Scraping 44

7.1.3. Worttyp 44

7.1.4. Designanpassungen 44

7.1.5. Erweiterung des Nutzerprofils 45

7.2. Erkenntnisse 45

7.3. Ausbau 45

7.3.1. Erweiterung der Skripts 45

7.3.2. Erweiterung der Datenbank 46

7.3.3. Neue Lernfunktionen 46

7.3.4. Erweiterung der Benutzeroberfläche 46

8. Reflexion 46

8.1.1. Allgemeines Vorgehen 46

8.1.2. Verbesserungspotenzial 47

8.1.3. Erwartung vs. Realität 47

8.1.4. Persönliche Erfahrungen 47

8.1.5. Positive Aspekte 48

8.1.6. Lernpunkte 48

8.1.7. Herausforderungen 48

Deklaration 49

Quellen 50

Anhang 54

Verwendung von KI-gestützten Tools 54

Anleitung 54

Setup 55

Sprichwörter – Skript 56

Redewendung – Skript 57

# Ausganglage

Eine Full-Stack-App für das Üben und Lernen der deutschen Sprache mit Übersetzungen in Französisch ermöglicht das Üben und Lernen von Wörtern und Redewendungen auf eine vielfältige Art und Weise.

Die Full-Stack-App wurde mit dem modernen Webframework Next.js 14 entwickelt. Im Backend wird zur Datenverwaltung eine MongoDB-Datenbank eingesetzt.

Das Design des Frontends wurde mithilfe von Tailwind CSS entworfen.

Die Full-Stack-App verfügt über eine Login-Funktion und mehreren Anmeldemöglichkeiten.

**Neuer Worttyp**

Ein weiteres Nebenziel wäre das Hinzufügen eines neuen Worttyps (Konjunktionen). In der Lernkategorie «Wortbedeutungen» befinden sich aktuell 11 verschiedene Worttypen. Die Datenbank soll durch den neuen Worttyp «Konjunktionen» erweitert werden. Dieses Nebenziel soll die Lernmaterialien zusätzlich erweitern.

## Datenbanktechnologie

In den folgenden Unterkapiteln wird die notwendige Grundlage für die Technologien im Zusammenhang mit der Datenbank und dem Backend bereitgestellt.

### Daten

Daten sind Informationen, die verarbeitet, gespeichert und analysiert werden können. Es gibt verschiedene Arten von Daten und Datengrösse. Die Speicherung von Daten erfolgt meist durch ein Datenbankmanagementsystem.

### Datenbank

Eine Datenbank ist ein organisierter Speicher von strukturierten, aber auch unstrukturierten Informationen. Normalerweise wird eine Datenbank von einem Verwaltungssystem (DBMS) gesteuert, welches für die Sicherheit und Datenintegrität verantwortlich ist. Eine Datenbank ist zuständig für das Speichern, Verwalten und Abfragen von Daten (*Was ist eine Datenbank?*, 2024).

Es gibt verschiedene Typen von Datenbanken, dazu gehören relationale, objektorientierte, verteilte sowie NoSQL-Datenbanken. Typischerweise bestehen Datenbanken hauptsächlich aus Tabellen (bei relationalen Datenbanken), Sammlungen (bei dokumentorientierten Datenbanken) oder anderweitigen Strukturen. Daten innerhalb der Datenbank können mit den sogenannten «CRUD»-Operationen bedient werden.

### MongoDB

MongoDB ist eine dokumentenorientierte NoSQL-Datenbank (nicht-relational). Sie ist kostenlos und Open-Source verfügbar. Die Vorteile welche MongoDB mit sich bringt, sind eine hohe Leistung und Verfügbarkeit sowie eine einfache Skalierbarkeit. Normalerweise verwenden herkömmliche (relationale) Datenbanken Tabellen und Zeilen für das Speichern ihrer Daten. MongoDB speichert Daten in Form von Sammlungen («collections») und Dokumenten («documents») anstelle von Tabellen und Zeilen. MongoDB verfügt über eine Vielzahl von Funktionen, darunter beispielsweise eine Aggregations-Pipeline für komplexe Abfragen und Datenanalyse (*Was Ist MongoDB?*, o. J.).

### Dokument – MongoDB

Dokumente («Documents») sind eigentlich die Grundbausteine einer MongoDB-Datenbank und können als einen Datensatz verstanden werden. Sie haben einen ähnliche Struktur wie ein JSON-Objekt, aber in MongoDB werden sie in einem binären Format namens «BSON» gespeichert. Somit ist die Datenstruktur eines Dokuments in Form von Feld-Wert-Paaren aufgebaut. Deshalb können sie leicht angepasst werden, ohne dass die gesamte Struktur der anderen Datensätze ändern zu müssen. Zudem besitzt jedes Dokument einen einzigartigen Identifikator namens \_id.

### Sammlung – MongoDB

Innerhalb einer MongoDB Datenbank befinden sich viele verschiedene Dokumente und diese werden in Sammlungen, sogenannten «Collections», gespeichert. Diese Collections funktionieren ähnlich wie Tabellen in relationalen Datenbanken, jedoch ohne eine fest vorgegebene Struktur (Schema-Vorgabe). Das bedeutet, dass jedes Dokument in einer Collection unterschiedliche Felder enthalten kann, was eine flexible Datenmodellierung ermöglicht (*Collections - MongoDB Manual v7.0*, 2024).

### MongoDB-Compass

MongoDB Compass ist eine grafische Benutzeroberfläche (GUI) für MongoDB. Mithilfe des MongoDB-Compass wird das Arbeiten mit der Datenbank vereinfacht. Es erleichtert das Durchsuchen und Verwalten der Datenbank. Anhand der grafischen Benutzeroberfläche können Abfragen optimiert und Daten besser analysiert werden. Zudem verfügt MongoDB-Compass über Werkzeuge zur Schema-Visualisierung, zur Erstellung von Aggregationspipelines, zur Indexverwaltung und zur Performance-Analyse.

[Download](https://www.mongodb.com/try/download/shell)

# [MongoDB Compass Tutorial](https://www.youtube.com/watch?v=xodSVriuYm4)

### Prisma

Prisma ist ein Open-Source ORM (Object-Relational Mapping) Framework. Es wurde spezifisch für TypeScript und Node.js entwickelt. Prisma wird als Schnittstelle zwischen der Webanwendung und Datenbank eingesetzt. Durch die automatisierten Migrationswerkzeuge und Datenmodellierungen vereinfacht Prisma die Arbeit für Entwickler\*innen.

Prisma besteht aus drei Hauptkomponenten:

* **Prisma Client:** Verkörpert den Hauptkern von Prisma, welches ein automatisch generierter und typsicherer Query-Builder ist. Er ermöglicht es, effiziente Datenbankabfragen zu generieren.
* **Prisma Migrate:** Ist für die Migration der Datenbank zuständig. Sie verwaltet das Datenbankschemata über den Verlauf der Zeit.
* **Prisma Studio:** Ist eine grafische Benutzeroberfläche (GUI), welche die Daten der Datenbank visualisiert. Die Daten können angezeigt und auch bearbeitet werden.

[Dokumentation](https://www.prisma.io/docs/orm/prisma-client/queries/relation-queries)

[Send database queries with Prisma Client](https://www.youtube.com/watch?v=LggrE5kJ75I)

### JSON

JSON (JavaScript Object Notation) ist ein Datenformat, welches für den Datenaustausch zwischen Servern und Webanwendungen verwendet wird. JSON wird wegen seiner klaren und einfachen Struktur sowohl von Menschen als auch von Maschinen leicht gelesen, geschrieben und analysiert. JSON kann in zwei grundlegende Bereiche unterteilt werden, nämlich in Objekte und in Arrays. Ein Objekt ist eine Sammlung von Schlüssel- oder Wert-Paaren und ein Array ist eine geordnete Liste von Werten. Sie können beide als JSON (Datenformat) gespeichert werden.

## Technologien

In den folgenden Unterkapiteln werden die grundlegenden Webtechnologien beschrieben, welche im Zusammenhang mit der Applikation und dem Frontend stehen.

### Next.js 14

Next.js 14 ist ein React-Framework, das die Erstellung und Optimierung von Webanwendungen vereinfacht und beschleunigt. Im Grunde ist das Framework wie ein Baukasten aufgebaut, wo Entwickler grundlegende und vorgefertigte Softwarekomponenten für den Aufbau ihrer Webanwendung verwenden können. Es enthält viele Bibliotheken, Tools und anderweitige Komponenten, die eine Basis bereitstellen. Die Version 14 bringt neue Funktionen und Verbesserungen mit sich, was einen positiven Einfluss auf die Performance und Bedienbarkeit hat (Vercel, Inc., 2024).

Dokumentation: <https://nextjs.org/docs>

Mögliche Tutorials auf Youtube:

[Next Js 14 Complete Course 2024](https://www.youtube.com/watch?v=p0iWbtHPel4)

### Tailwind CSS

Tailwind CSS ist ein CSS-Framework, welches bekannt für einen schnellen und effizienten Aufbau von Benutzeroberflächen ist. Utility-First ist eines der Hauptmerkmale von Tailwind CSS, dabei findet das Styling innerhalb des HTML-Codes statt. Mit einer Reihe von vordefinierten Utility-Klassen ist es Entwickler möglich rasch und flexibel Komponenten spezifisch für ihr Projekt zu erstellen. Dies ist ein wichtiger Vorteil der Zeit und Code spart, da keine separate CSS-Datei geführt werden muss.

Tutorial

[Tailwind CSS Full Course 2023](https://www.youtube.com/watch?v=tS7upsfuxmo&t=14s)

<https://play.tailwindcss.com/>

<https://play.tailwindcss.com/MNMK1mUmEn>

Pading and margin: <https://play.tailwindcss.com/KslLNSrEbq>

Einfach: <https://play.tailwindcss.com/NVPwT8Ji17>

### FontAwesome

FontAwesome ist eine beliebte Bibliothek für Icons und Logos, welche leicht integriert werden kann und sehr anpassungsfähig ist. Sie bietet Tausende von Icons und Logos in verschiedene Grössen, Farben, Stilen und Kategorien an. Diese beiden Technologien leisten einen grossen Beitrag im Bereich Design und Ästhetik der Sprachapp (*Font Awesome*, 2024).

### JavaScript

JavaScript ist eine dynamische Programmiersprache und wird in der Webentwicklung eingesetzt. Zusammen mit HTML und CSS gehört JavaScript zu den Grundtechnologien der Webentwicklung und des heutigen Internets. JavaScript zeichnet sich durch ihre Interaktivität und dynamischen Inhalten aus. Beispielsweise ermöglicht es auf Ereignisse einer Webseite zu reagieren, Inhalte zu aktualisieren, interaktive Elemente zu gestalten sowie Animationen zu implementieren.

### Visual Studio Code

Visual Studio Code (VS Code) ist ein Quellcode-Editor, der eine Vielzahl von Frameworks sowie Programmiersprachen unterstützt. VS Code hat eine schnelle und leistungsfähige Bearbeitungsfunktion und kann sowohl für alltägliche Aufgaben als auch für komplexere Entwicklungsprojekte eingesetzt werden. Er zeichnet sich durch seine Anpassbarkeit sowie Erweiterbarkeit aus (Microsoft, 2024).

Tutorials

[How to get started with VS Code](https://www.youtube.com/watch?v=EUJlVYggR1Y)

[Getting started with Visual Studio Code](https://code.visualstudio.com/docs/introvideos/basics)

### Github

Ergänzen

[Introduction to Git in VS Code](https://code.visualstudio.com/docs/sourcecontrol/intro-to-git)

# Analyse der App

## Frontend

### Login

Die Applikation verfügt über vier verschiedene Möglichkeiten, sich in die Full-Stack-App einzuloggen: Google, GitHub, Discord und ein allgemeines E-Mail-Login. Für das Einloggen ist ein gültiges Konto bei den jeweiligen Anbietern erforderlich.

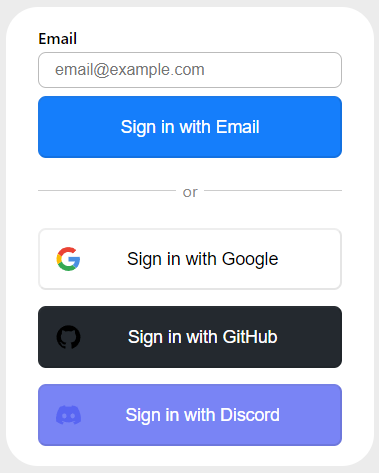


Abbildung 1 Login-Optionen

**Authentifizierung**

Wenn die Benutzer\*innen bei einem der vorhandenen E-Mail-Provider den «Anmelden»-Button anwählen, werden sie von der Applikation zu dem jeweiligen E-Mail-Provider (Google, GitHub, Discord oder allgemeines Login) weitergeleitet. Benutzer\*innen werden dann aufgefordert, der Applikation bestimmte Berechtigungen zu gewähren. Durch die Bestätigung der Benutzer\*innen werden sie automatisch bei dem jeweiligen E-Mail-Provider angemeldet, falls dies noch nicht bereits geschehen ist.

Nach der Zustimmung werden die Benutzer\*innen zurück zur Applikation geleitet, zusammen mit einem Authorization Code. Die Applikation sendet den erhaltenen Authorization Code zusammen mit der client\_secret-Information zurück an den E-Mail-Provider, um ihn gegen ein Access Token (Zugangstoken) einzutauschen. Die client\_id und client\_secret sind Teil des OAuth-Authentifizierungprozess. Sie werden verwendet, um die Anwendung sicher mit dem E-Mail-Provider (z.B. Discord, GitHub) zu identifizieren.

Mit dem Access Token (Zugangstoken) kann die Applikation nun Benutzerinformationen von dem E-Mail-Provider abfragen und kontrollieren, ob sie übereinstimmen. Wenn der Prozess der Authentifizierung gültig ist, erstellt die Anwendung automatisch eine Sitzung für die Benutzer\*innen, womit sie nun eingeloggt sind und mit dem Lernen beginnen kann.

**Discord & GitHub**

Ein entscheidender Vorteil der NextAuth-Bibliothek ist, dass sie über viele verschiedene Login «Provider» verfügt und diese leicht implementierbar sind. Für Discord und Github mussten im Vorhinein die Anmeldeinformationen erstellt werden. Ohne diese beiden Informationen (client\_id und client\_secret) ist es nicht möglich den Nutzer mit NextAuth und OAuth zu authentifizieren. Die client\_id und client\_secret sind Teil des OAuth-Authentifizierungprozess. Sie werden verwendet, um die Anwendung sicher mit dem Provider (z.B. Discord, GitHub) zu identifizieren.

Um diese Anmeldeinformationen (client\_id und client\_secret) zu erlangen, muss bei den jeweiligen Providern (GitHub und Discord) eine neue «OAuth App» eingerichtet werden. Nach der Erstellung liefern die beiden Provider die entsprechenden Informationen für den Loginprozess. Diese Informationen können kopiert und in der «.env.local»-Datei des Projekts eingefügt werden.

### Layout

**Lernkarte**

Mit Layout ist die Inhaltsdarstellung der Lernkarte gemeint. Der Fokus der Anpassungen liegt auf den Texten innerhalb der jeweiligen Lernkarten. Die Lernkarten sind unterschiedlich aufgebaut und werden demzufolge nach individuell angepasst.

Es sollte auf ein einheitliches Design der verschiedenen Lernkarten geachtet, dennoch können sie sich von anderen in Bezug auf Schriftgrösse und Struktur sowie Aufbau der Lernkarte unterscheiden.

Beispielsweise kann bei einer Lernkarte die Schriftgrösse anders sein als bei den anderen, aufgrund von Platzverhältnissen. Hinzukommen weitere Frontendanpassungen in Bezug auf das generelle Design im Frontend Bereich, um die Benutzeroberfläche ansprechender und intuitiver zu gestalten.

### Lernsession

Die Applikation verfügt über einen Lernbereich, den sogenannte «Übungsstapel», in dem den Benutzern vier Lernkategorien zur Auswahl stehen sind: «Wortbedeutungen», «Präpositionen», «Präpositionen & Verben» und «Sprichwörter».

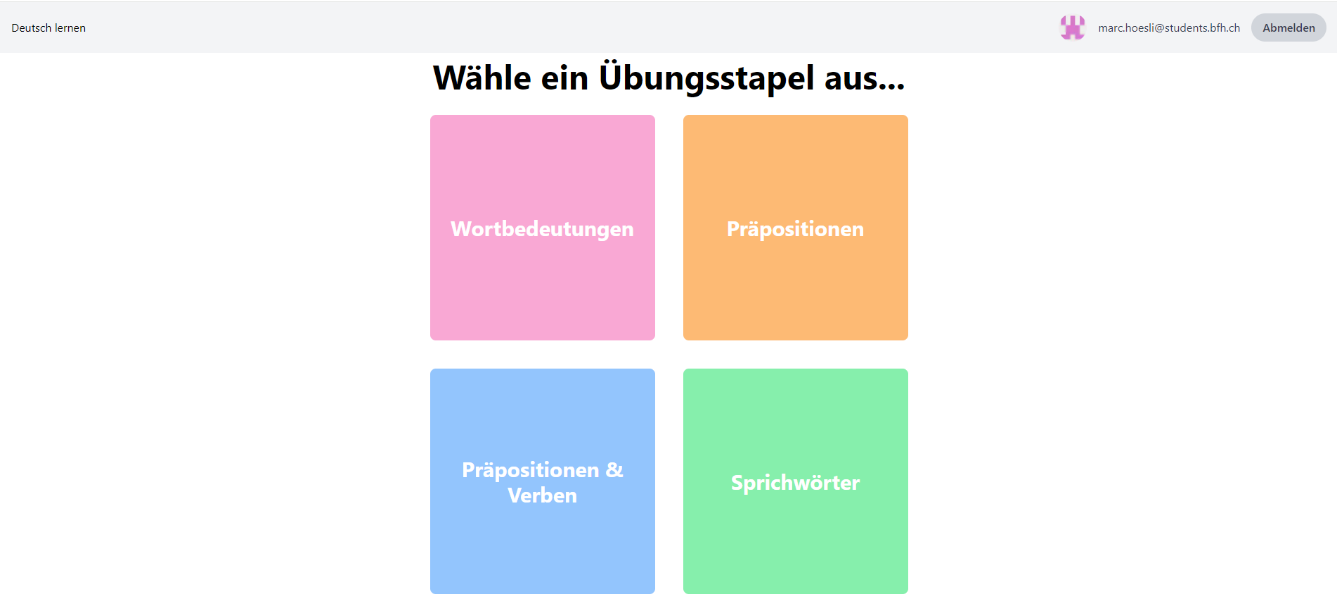


Abbildung 2 Lernkategorien

In der Abbildung 2 werden die vier Lernkategorie nochmals bildlich dargestellt. Jeder dieser Lernkategorien hat verschiedene Wörter, Sätze und Aufgabenstellung, die erlernt werden können.

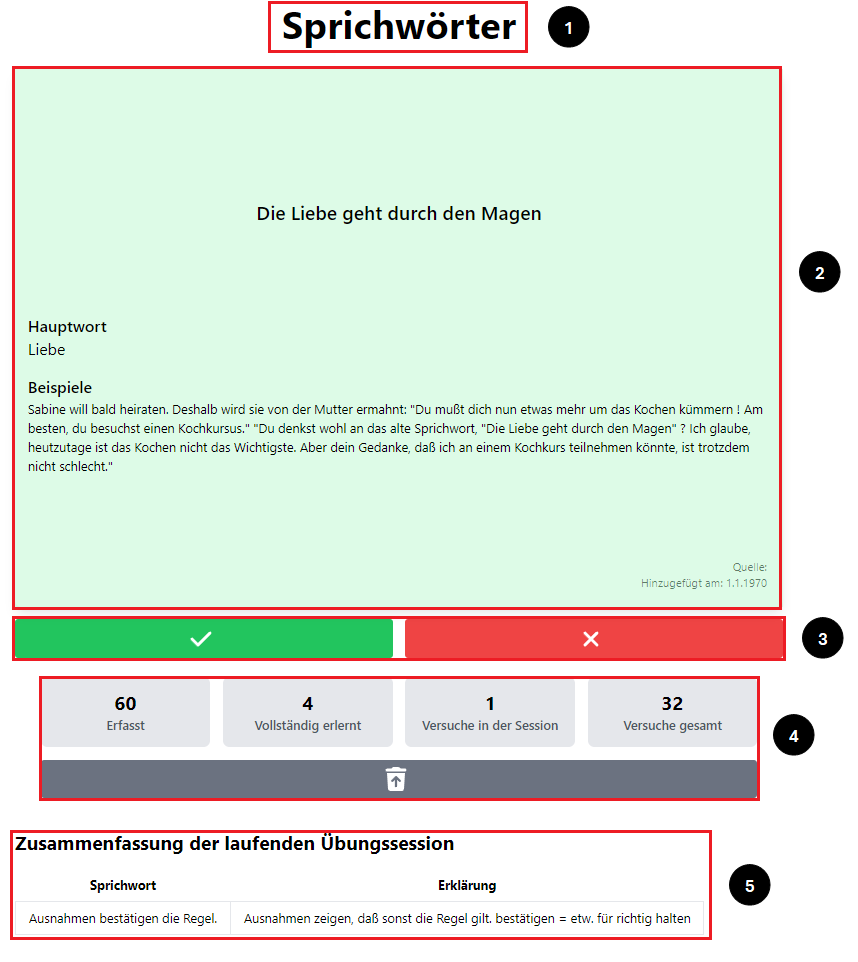


Abbildung 3 Lernkarte - Sprichwörter

Die Wörter innerhalb der Lernkategorie werden alle als Karteikarten präsentiert, die eine Vorder- und Rückseite haben. Im folgenden Abschnitt werden die Logik und der Aufbau der Lernkategorie «Sprichwörter» mithilfe der Abbildung 3 näher vorgestellt.

1. **Lernkategorie**

Die Lernkategorie wird in allen Bereichen als Überschrift oberhalb der Lernkarte angezeigt. Sie dient als Information und weist die Benutzer\*innen darauf hin, in welcher Lernkategorie sie sich aktuell befinden.

1. **Lernkarte**

Die Sprichwörter oder der Lerninhalt wird als «Lernkarte» präsentiert. Alle zu lernenden Informationen befinden sich innerhalb dieser Lernkarte. Mit einem Mausklick auf die Lernkarte wird diese umgedreht, um die Lösung anzuzeigen. Die Lernkategorien haben zwar den gleichen Aufbau, dennoch können sich die Lernkarten bezüglich des Lerninhaltes ein wenig unterscheiden.

1. **«Richtig»- oder «Falsch»-Button**

Mithilfe eines roten und eines grünen Buttons können die Karten nach dem Umdrehen bestätigt werden, wobei der grüne Button für «richtig» und der rote Button für «falsch» steht. Nach dem Anwählen eines dieser Buttons erscheint eine weitere Lernkarte, bis der Stapel vollständig erlernt wurde. Der Lernstand wird dementsprechend auch automatisch angepasst. Das ist das Lernprinzip und die Logik hinter dem Karteikartensystem.

1. **Lernstand**

«Erfasst» gibt an, wie viele Wörter sich insgesamt in der jeweiligen Lernkategorie befinden. Eine Lernkarte muss zweimal als korrekt angewählt werden, um als «vollständig erlernt» eingestuft zu werden. Bei den Versuchen wird zwischen den «Versuche in der Session» und den «Versuche gesamt» unterschieden. Die laufenden Versuche beziehen sich auf die aktive Lernsession. Die gesamthaften Versuche beziehen sich wiederum auf die Versuche aus mehreren Lernsessions.

Durch das Auslösen des grauen Buttons («Reset») kann der Lernstand zurückgesetzt werden und ermöglicht es den Benutzer\*innen, die Lernsession erneut zu beginnen. Der Lernstand wird automatisch zurückgesetzt, wenn alle Lernkarten als «vollständig erlernt» eingestuft sind.

1. **Zusammenfassung der Lernsession**

Die Zusammenfassung liefert Informationen zur aktiven Lernsession. Jede Lernkarte, die die Benutze als korrekt oder als falsch markiert haben, wird in der Zusammenfassung vermerkt und zusammengefasst. Die Lernkarten werden in der Zusammenfassung in Form einer Tabelle angezeigt.

### Worterfassung

Der dritte Bereich der Applikation ist die «Worterfassung». Mithilfe der Worterfassung können neue Wörter (Daten) in die Datenbank hinzugefügt werden. Da die Applikation über vier verschiedene Lernkategorien verfügt, gibt es für jede dieser Kategorien ein entsprechendes Formular oder Felder, um neue Wörter hinzuzufügen. Im folgenden Abschnitt werden der Aufbau und die Logik der Worterfassungskategorie «Präpositionen & Verben» mithilfe der Abbildung 4 näher vorgestellt.



Abbildung 4 Worterfassung - Präposition & Verben

1. **Erfassungskategorie**

Die Kategorie wird in allen Bereichen als Überschrift oberhalb der Lernkarte angezeigt. Sie dient als Information und weist die Benutzer\*innen darauf hin, in welche Erfassungskategorie sie sich aktuell befinden.

1. **Erfassungsformular**

Für das Erfassen neuer Wörter steht ein Formular mit Felder zur Verfügung, das mit der MongoDB-Datenbank verknüpft ist. Es müssen alle Felder ausgefüllt werden, um das Wort oder den Satz erfolgreich hinzufügen zu können. Die Formularfelder können sich je nach Erfassungskategorie unterscheiden.

1. **Hinzufüge-Button**

Wenn alle Felder korrekt ausgefüllt sind, kann der blaue Button ausgelöst werden, um die Daten an die MongoDB-Datenbank zu senden. Wenn die Daten erfolgreich übermittelt wurden, erscheint die Meldung «Erfolgreich hinzugefügt!». Falls die Felder unkorrekt oder unvollständig ausgefüllt wurden, erhalten die Benutzer\*innen die Meldung «Fülle bitte alle Felder aus!».

Der graue Button ist der Reset-Button und setzt das gesamte Deck der Lernsession wieder auf null. Wenn man bereits Wörter gelernt hat, jedoch noch nicht am Ende der Lernsession ist, wird der Stand beim Betätigen dieses Buttons zurückgesetzt. Das Abfallsymbol könnte theoretisch durch ein "Reverse-Pfeil-Symbol" ersetzt werden, da Abfallsymbole eher für das Löschen als für das Zurücksetzen bekannt sind.

Implementieren

     Beispielsymbole:

Ein Bild, das Symbol, Logo, Schrift, weiß enthält.

Automatisch generierte Beschreibung      Ein Bild, das Logo, Symbol, Silhouette enthält.

Automatisch generierte Beschreibung

## Datenbankstruktur

Für die gesamte Datenverwaltung wird mit den Datenbanktechnologien von MongoDB, MongoDB Compass und Prisma gearbeitet.

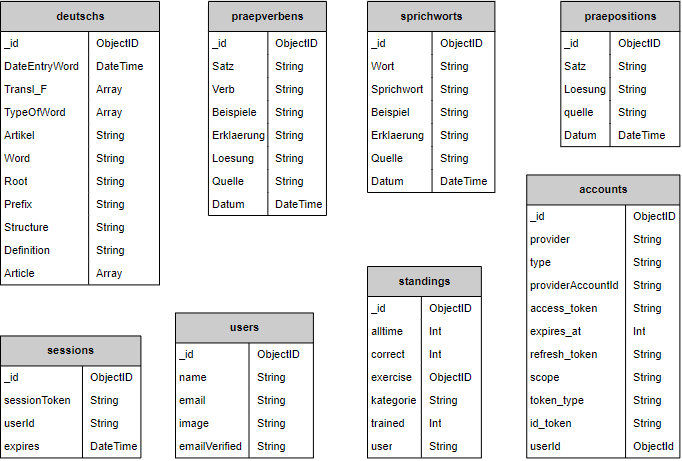


Abbildung 5 Datenbankschema - MongoDB

Insgesamt befinden sich acht Collections in der MongoDB-Datenbank. Diese Collections bilden das Datenbankschema in Abbildung 5. Jede dieser Collection hat verschiedene Felder sogenannte «Fields». Diese Felder haben jeweils einen Wert, sogenannte «Values». Die Werte innerhalb einer Collection sowie innerhalb eines Felds können sich jeweils unterscheiden.

### Datentypen

Die verwendeten Werte in diesem Datenbankschema sind *Strings*, *Array*, *Int*, *DateTime* und *ObjectID*. Ein *String* ist der am häufigsten verwendete Datentyp, um Textdaten zu speichern. Ein *Array* wird mit eckigen Klammern dargestellt und kann mehrere Werte in einem Feld speichern. Ein *Int*, auch *Integer* genannt, ist ein Datentyp, der Werte wie Ganzzahlen speichert.

Die *DateTime* ist ein Datentyp, der verwendet wird, um Datum- und Zeitinformationen zu speichern. Die *ObjectID* ist ein weiterer Datentyp, der als Primärschlüssel für ein Dokument innerhalb einer Collection verwendet wird. Der Wert in einem *ObjectID*-Feld ist einzigartig und wird automatisch von MongoDB generiert (*ObjectID - BSON Types*, 2024).

### Lernbereich

Die Collections *deutschs***,** *praepositions***,** *praepverbens***,** und *sprichworts*bilden den Lernbereich im Backend. Sie sind kohärent mit den vier Lernkategorien Wortbedeutungen (*deutschs)*, Präpositionen (*praepositions*), Präpositionen & Verben (*praepverbens*) und Sprichwörter (*sprichworts*), die den Lernbereich im Frontend darstellen.

### Felder

Das Feld *\_id*ist dafür verantwortlich, dass jedes Dokument innerhalb der Collection einen einzigartigen und einmaligen Identifikator besitzt. Die Felder *DateEntryWord*und*Datum*liefern Informationen darüber, zu welchem Zeitpunkt das Dokument (Datensatz) in die Datenbank aufgenommen oder implementiert wurde. Das Feld *Quelle* liefert Information zur Quellenangabe.

### Verwaltungsbereich

Die weiteren vier Collections *sessions*, *users*, *standings*und *accounts*sind für die Verwaltung der Lernsession sowie der Daten der Benutzer\*innen zuständig. Die Collection s*essions* ist für das Verwalten der Daten der angemeldeten Benutzer\*innen zuständig. Die Collection *users* enthält die Profildaten von den Benutzer\*innen, wie zum Beispiel die E-Mail-Adresse vom Loginbereich. Die Collection *standings* ist der Zustandsspeicher, in dem die Daten der Lernsession von den jeweiligen Benutzer\*innen gespeichert werden. Die Collection *accounts* ist ähnlich wie die Collection *users* und verwalten die Accountdaten.

### Querys

MongoDB-Compass benutzen, um die Query-Abfragen zu tätigen.

In dieser [Quelle](https://www.mongodb.com/docs/compass/current/query/filter/#query-your-data) werden die meist verwendeten Query-Funktionen kurz erklärt und mit Beispielen weiter erläutert. Zudem gibt es in MongoDB-Compass selbst eine KI-Funktion ("Generate Query"). Dafür müssen Sie mit einem Atlas-Account angemeldet sein, den Sie einfach erstellen können. Mit "Generate Query" können Sie durch Eingabe eines Befehls (Prompt) die gewünschten Daten abfragen.

KI-Funktion dokumentieren.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

**Abbildung 3**

Alle Nomen die keinen Artikel vorweisen



Alle Wörter ausgeben welche auf «ung», «heit», «keit», «nis», «tum», «tät» und «schaft» enden, die nicht als Nomen kategorisiert sind

****

**Einzelabfrage** -> Alle Wörter ausgeben welche auf «haft», «ig», «mut», «bar», «sam», «wert», «lich», «reich», «arm», «voll», «isch», «frei» und «los» enden, die nicht als Adjektive kategorisiert sind

****

Alle «adj.» ausgeben, welche innerhalb «Transl\_F.Transl\_F» (französisches Wort) vorkommen und nicht als Adjektive kategorisiert sind.



Alle Wörter ausgeben, welche den Partikel «sich» enthalten (z.B. sich entfalten), die nicht als “Reflexives Verb” kategorisiert sind



Alle Wörter ausgeben welche als «vi.» (intransitive Verben) markiert sind, die nicht zu einer Verbform gehören, sondern als einen anderen Worttypen (z.B. Nomen, Adjektiv etc.) kategorisiert sind.



Alle «vt.» (transitiven Verben) ausgeben die mit «vt.» markiert sind aber nicht als «Transitives» Verb kategorisiert ist.



Spezifische Suchabfrage, wenn nur ein bestimmtes Wort gesucht wird.



**4.3.1. Datenbankbereinigung**

Verschaffen Sie sich mittels Prisma Studio und MongoDB-Compass ein Gesamtüberblick über die ganze Datenbankstruktur.

****

**Abbildung 4**

**Datensatz**

In der Abbildung 4 wird der Aufbau eines Datensatzes, ein Wort in der MongoDB-Datenbank, gezeigt.

Jedes Wort innerhalb der Datenbank wird als ein Datensatz betrachtet und folgt einem einheitlichen System. Jeder Datensatz hat die gleiche Struktur und ist somit gleich aufgebaut. Dennoch teilen nicht alle Datensätze die gleichen Inhalte. Nicht nur Datenbanken befolgen ein bestimmtes System, sondern auch Sprachen. Jede Sprache hat gewisse Regeln, wie Wörter gegliedert, geschrieben und aufgebaut sind.

**Aggregationspipeline**

Für die Analyse der Datensätze innerhalb der Datenbank wurde mithilfe von MongoDB-Compass eine sogenannte «Aggregationspipeline» erstellt. Eine Aggregationspipeline zeigt, wie Daten in einer Datenbank analysiert werden können. Sie ermöglicht es, Daten durch eine Reihe von Verarbeitungsschritten, sogenannte Stufen («stages»), zu leiten. Jeder dieser Stufen wendet eine bestimmte Operation auf die Daten an. In diesem Schritt

Danach erstellen Sie Abfragen mithilfe von Aggregationspipelines an die Datenbank.

Kategorien (TypeOfWord) befinden.

Ein Bild, das Text, Screenshot, Schrift, Design enthält.

Automatisch generierte Beschreibung

**Abbildung 5 Aggregationspipeline**

Die Aggregationspipeline kann in die Stufen

«$unwind», «$group» und «$sort» unterteilt werden.

In der ersten Stufe «unwind», wird ein Array-Feld zerlegt und aufgeteilt. Ein Array ist eine einfache Sammlung von Elementen (Werten oder Variablen), welche in einer bestimmten Reihenfolge gespeichert sind.

Mit $unwind wird somit für jedes Element in diesem Array ein separates Dokument erstellt. Das Feld TypeOfWord (Array) wird aufgeteilt, sodass jedes Dokument nur einen Worttyp (z.B. Nomen, Verb, Adjektiv) enthält.

n der zweiten Stufe «group», werden die Dokumente nach einem bestimmten Schlüssel (Worttyp) gruppiert.

Mit \_id: «$TypeOfWord.TypeOfWord» wird ngegeben, dass die Gruppierung auf der Basis des «TypeOfWord.TypeOfWord»-

eldes erfolgt. Das Feld count wird dann erwendet, um die Anzahl der Dokumente in eder Gruppe zu zählen.

n der letzten Stufe «sort» werden die Dokumente nach dem Feld count in bsteigender Reihenfolge (-1) sortiert. Was omit bedeutet, dass die Worttypen mit den meisten Wörtern zuerst angezeigt werden.

Die Aggregationspipeline gab die Anzahl der orttypen und gleichzeitig auch die Anzahl

Ein Bild, das Text, Screenshot, Dokument, Zahl enthält.

Automatisch generierte Beschreibung

**Abbildung 6 Ergebnis Aggregationspipeline**

**Versionskontrolle**

Mit dem Befehl «npm list» wird eine Ausgabe erstellt, welche die bereits installierten Pakete und Versionen von den eingesetzten Technologien aufzeigt. Gibt die Konsole das Statement «empty» oder «unknown command» als Antwort auf den Befehl, ist das gewünschte Paket oder die vorhandene Version nicht installiert und müsste daher noch installiert werden. Dieser Schritt ist notwendig, um zu überprüfen, ob das Projekt über die nötigen Pakete und Technologien verfügt.

Mit dem Befehl «npm i next@latest» die Next.js Version aktualisieren. Der Begriff «i» steht für Installation und «latest» steht für die, zu diesem Zeitpunkt, aktuellste Version von Next.js.

## Anforderungen

Anhand der durchgeführten Grundlagenanalyse werden in diesem Abschnitt die Anforderungen beschrieben. Sie dienen als Leitlinie für die spätere Entwicklung und technischen Umsetzung.

# Web Scraping

## Web-Scraping-Prozess

Da zu Beginn der Entwicklungsphase keine Vorkenntnisse im Bereich Web Scraping vorhanden waren, wurden Recherchearbeiten durchgeführt. Hierfür wurde die Literatur «Web Scraping with Python» von Ryan Mitchell studiert (Mitchell, 2024).

Tutorial von «freeCodeCamp.org» auf Youtube als Einstieg und Unterstützung in die Thematik genutzt (freeCodeCamp.org, 2020). Dieser Prozess war notwendig, um ein grundlegendes Wissen zu erwerben. Für die Entwicklung der Skripte wurden zusätzlich KI-gestützte Tools wie GitHub Copilot verwendet.

### Skripts

*Skripts* sind kleine Programme, die eingesetzt werden, um eine oder mehrere spezifische Aufgaben automatisch auszuführen. Der Umfang, die Komplexität oder die Aufgaben eines Skripts können sehr stark variieren (Seiten-Werk, o. J.).

In diesem Fall war das Ziel der Skripts, die gewünschten Daten von den Quellen automatisch zu extrahieren und somit den manuellen Zeitaufwand zu reduzieren. Für den Web-Scraping-Prozess wurden drei separate Skripts erstellt. Dabei wurde darauf geachtet, dass die Wiederverwendung von Code gewährleistet wird und dass alle Skripte somit eine ähnliche Struktur verfolgen.

### Methodenauswahl

Für die Entwicklung der Skripte wurde die Python-Bibliothek «BeautifulSoup» eingesetzt. Der Entscheidungsgrund für BeautifulSoup basierte auf den Eigenschaften der Quellen. BeautifulSoup eignet sich sehr gut für Scraping-Aufgaben im Zusammenhang mit Daten aus statischem HTML und für kleinere Projekte. Statische Webseiten sind dadurch erkennbar, dass sie oftmals gut strukturiert sind und wenig dynamische Inhalte (JavaScript) aufweisen, wodurch auch die Komplexität des Aufbaus der gesamten Seite geringer ist.

Die definierten Quellen in diesem Projekt treffen auf diese Beschreibung zu. Die Webseiten bestehen hauptsächlich aus textbasierten Daten und enthalten nur wenige dynamische Inhalte. Aus diesem Grund haben die Quellen einen grossen Einfluss auf den Scrapingprozess und somit auch auf den Einsatz der Python-Bibliothek.

Scrapy wurde nicht verwendet, da die Quellen keine hohe Datenmenge aufweisen. Scrapy wird hauptsächlich bei sehr grossen und komplexen Projekten eingesetzt. Da es sich in diesem Fall um eine kleineres Projekt handelt, wurde auf den Einsatz von Scrapy verzichtet.

Selenium wurde ebenfalls nicht verwendet, da die Daten sowie die Quellen selbst nicht viele dynamischen Inhalten oder Nutzerinteraktionen aufwiesen. Selenium ist bekannt für das Arbeiten mit JavaScript und sehr viel dynamischen Inhalten. Da die Quellen in diesem Projekt sehr textbasiert sind, kam der Einsatz von Selenium nicht in Frage.

Bei der Entscheidungswahl wurden auch persönliche Präferenzen berücksichtigt. Beispielsweise gibt es keinen regulären Massstab ab wann eine Datenmenge als hoch oder als niedrig gilt (Palakollu, 2020).

# Webscraping

## Idee

Die Idee ist es mithilfe von Web Scraping-Techniken neue Daten automatisch in die Datenbank einzubinden. Web Scraping ist der Prozess, bei dem Daten von Webseiten extrahiert werden, durch den Einsatz von automatisierten Tools. Diese Technik ermöglicht es, grosse Mengen an Informationen aus dem Internet zu sammeln, ohne sie manuell kopieren und einfügen zu müssen.

In der Regel werden Programme oder Skripte eingesetzt, die die Struktur einer Webseite analysieren, die gewünschten Daten identifizieren und diese dann in einer strukturierten Form speichern, wie beispielsweise in einer Datenbank. Die Verwendung von Web Scraping bietet eine automatisierte Datensammlung, Effizienzsteigerung, Erweiterung des Datenbankvolumens und Förderung der Lernmaterialien.

Web Scraping ermöglicht es, automatisiert aktuelle und relevante Daten aus dem Internet zu sammeln. Dies ist besonders nützlich, um die Datenbank kontinuierlich mit neuen und aktuellen Lernmaterialien zu aktualisieren, ohne dass dafür manuelle Eingriffe notwendig sind.

## Web-Scraping-Technologie

In den folgenden Unterkapiteln wird die theoretische Grundlage des Web Scraping erklärt.

### Web Scraping

Web Scraping bedeutet im Grunde eigentliche das automatisierte Sammeln von Daten aus dem Internet, welches fast so alt ist wie das Internet selbst. Erreicht wird dieser Prozess durch das Schreiben und Erstellen eines automatisierten Programms, welches den Webserver abfragt, Daten anfordert und diese Daten dann analysiert, um schlussendlich die benötigten Information zu extrahieren (Mitchell, 2024, Preface).

**Typologie**

Web Scraping ist kein neuer Begriff und wurde im Verlaufe der Zeit auch als *Screen Scraping, Web Harvesting, Data Mining* oder anderen Formen bezeichnet. Der Bereich ist sehr breit gefächert, deshalb auch die verschiedenen Begriffe, welche alle auf Web Scraping zurückzuführen sind (Mitchell, 2024, Preface).

Es gibt verschiedene Methoden und Arten wie Daten mittels Web Scraping extrahiert werden können. Einerseits bieten gewisse Webseiten APIs an, die genutzt werden können, andererseits gibt es bereits browserbasierte Tools, wo mithilfe eines Web Scraping Plug-Ins Daten extrahiert, werden können. Trotzdem stehen die Bibliotheken in Python an vorderster Front. Sie bilden die Grundlage für das klassische Web Scraping. Die beliebtesten Web Scraping Bibliotheken sind Scrapy, Selenium und BeautifulSoup (Mitchell, 2024, Preface).

### Python

Python ist eine sehr beliebte und vielfältige Programmiersprache. Sie ist bekannt für ihre Einfachheit und Lesbarkeit, somit kann sie von Anfänger\*innen als auch von fortgeschrittenen Entwickler\*innen genutzt werden. Python wird in vielen verschiedenen Anwendungsgebieten eingesetzt, darunter Webentwicklung, Datenanalyse, maschinelles Lernen und künstliche Intelligenz. Sie zeichnet sich durch eine klare und einfache Syntax sowie ein grosses Angebot an Bibliotheken aus, was die Entwicklung von Software effizient und produktiv macht.

Entwickler\*innen schätzen besonders, dass der Zyklus von Bearbeitung, Tests und Debugging sehr schnell ist. Da in Python keinen Kompilierungsschritt erforderlich ist wird somit automatisch die Produktivität gesteigert. Ein weiterer Vorteil ist, dass Fehler in Python-Programmen nicht zu einem Absturz des Systems führen, sondern Ausnahmen auslösen, die leicht zu identifizieren und zu beheben sind (*What Is Python?*, 2024).

### BeautifulSoup

BeautifulSoup ist eine Bibliothek in Python, welche für das Extrahieren von Daten aus HTML- und XML-Dateien verwendet werden kann. Ein wichtiger Bestandteil von BeautifulSoup ist der sogenannten «Parser». Ein Parser zerlegt Daten einer HTML-Datei in kleinere Einheiten, welche anschliessend in einer strukturierten Form (meistens in Form eines Syntaxbaumes) wiedergegeben werden. Mithilfe des Parsers, ermöglicht BeautifulSoup das Durchsuchen der Dokumentstruktur sowie das Extrahieren und Verarbeiten von Daten, basierend auf ihren «HTML-Tags». Die Python-Bibliothek wird im Bereich des Web Scraping angewendet (Leonard Richardson Revision 546, 2024).

### Scrapy

Scrapy ist ein Anwendungsframework, was von Python entwickelt wurde, um strukturierten Daten effizient extrahieren zu können. Scrapy wird bei grossen und komplexen Projekten eingesetzt. Es ist bekannt für Effizienz und Geschwindigkeit bei grossen Datenmengen (*Scrapy at a glance — Scrapy 2.11.1 documentation*, o. J.).

### Selenium

Selenium ist ein Tool für das Automatisieren von Webanwendungen. Selenium wird auch im Bereich Web Scraping, bei umfangreichen Webseiten, die viel Nutzerinteraktionen aufweisen, eingesetzt. Deshalb kann es sehr gut bei Webseiten verwendet werden, welche viel JavaScript und dynamische Inhalte enthalten (*The Selenium Browser Automation Project*, o. J.).

Hinzufügen von Konjunktionen

Diese Seite (<https://www.colanguage.com/de/konjunktionen-im-franz%C3%B6sischen>) bietet zu jeder Konjunktion auch ein entsprechendes Beispiel an. Ich habe bereits erfolgreich die Daten von der Seite extrahiert.

## Datensammlung

Im folgenden Abschnitt werden die entwickelten Skripte mit ihrer Codebasis näher beschrieben.

Abbildung 6 Skript – Konjunktion



Abbildung 7 Import-Befehle

Mit dem Befehl **import** können Bibliotheken, spezielle Funktionen oder anderweitige Pakete in das Skript importiert werden. Der Importbefehl in diesem Skript verfolgte ein ähnliches Prinzip wie das eines ein Frameworks. Anstatt spezielle Funktionen von Grund auf selbst zu programmieren, konnten diese dank dem Importbefehl implementiert und somit Zeit gespart werden.

* **import requests**

Mit diesem Befehl wurde das Modul für HTTP-Anfragen importiert (*Python Requests Module*, o. J.).

* **from bs4 import BeautifulSoup**

Mit diesem Befehl wurde die Bibliothek BeautifulSoup importiert. Sie war für das HTML-Parsing zuständig, also für das Aufteilen der HTML-Struktur in kleine Teile (Mitchell, 2024, Chapter 1).

* **import json**

Mit import *json* wurde das Datenformat JSON importiert. Dieses Format wurde für das Speichern der Daten in einem gültigen Datenformat (JSON) benötigt (*Python JSON*, o. J.).

* **from datetime import datetime**

Mit dieser Funktion wurde die aktuelle Zeit und das aktuelle Datum importiert. Dieser Import wurde eingesetzt, um die Zeit der Quellenangabe zu bestimmen (*Python Dates*, o. J.).

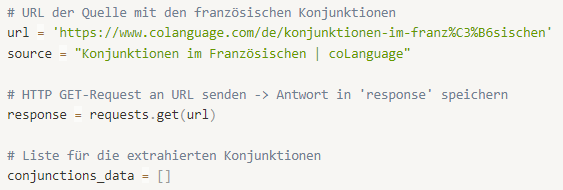


Abbildung 8 Verknüpfung Webseite

In diesem Codeabschnitt wurde die Webseite sowie die Quellenangabe definiert. Danach wurde ein GET-Request und eine leere Liste erstellt.

* **url = 'Link zur gewünschten Webseite'**

In der Variable «URL» wurde die Webseite definiert. Von diesem Ort aus hat das Programm die Daten der Zielseite entnommen (*Konjunktionen im Französischen | coLanguage*, o. J.).

* **source = "Konjunktionen im Französischen | coLanguage"**

Die Variable «source» enthielt den Titel der Webseite.

* **response = requests.get(url)**

Um Zugriff auf die Daten zu erlangen, musste immer im Vorhinein eine HTTP-Anfrage erstellt werden. Somit wurde eine GET-Anfrage an die URL gesendet, was die Antwort direkt speicherte (*Python Requests Get Method*, o. J.).

* **conjunctions\_data = []**

In dieser Variable wurde mithilfe der beiden eckigen Klammer (« [] ») eine leere Liste erstellt. Die leere Liste wurde dafür verwendet, die Daten, welche von der Webseite extrahiert werden, zu speichern.

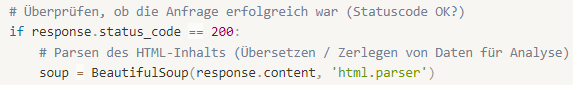


Abbildung 9 HTML-Parser

Mit der «*if*»-Anweisung wurde überprüft, ob der GET-Request (Anfrage) erfolgreich ist (*Python Requests.Response Object*, o. J.).

* **if response.status\_code == 200**

Die definierte Variable *response* erstellte einen GET-Request an die gewünschte Webseite. Mit der Zahl 200 wurde überprüft, ob der erstellte GET-Request gültig ist. Der Statuscode 200 in HTTP, steht für OK (*200 OK - HTTP | MDN*, 2023).

* **soup = BeautifulSoup(response.content, 'html.parser')**

In der Variable *soup* wurde die Bibliothek von BeautifulSoup aufgerufen. Innerhalb der Klammern der Bibliothek BeautifulSoup wurde der Inhalt der Webseite aufgerufen und gleichzeitig «geparst». Parsen ist ein Prozess wo die HTML-Struktur, besser gesagt die Daten der HTML-Struktur, in kleine Datenstücke zerlegt werden (Mitchell, 2024, Chapter 1).

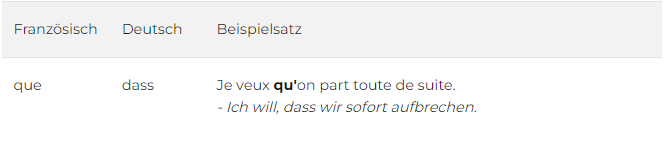


Abbildung 10 Konjunktion von Webseite

In der Abbildung 10 ist ein Ausschnitt einer Konjunktion abgebildet, der einen kleinen Teil der Webseite darstellt, die gescraped wurde. Die Abbildung 11 zeigt den reinen HTML-Code der Konjunktion aus Abbildung 10.

Somit sind der HTML-Code aus Abbildung 11 und die Konjunktion aus Abbildung 10 dementsprechend genau dasselbe. Der Ausschnitt aus der Abbildung 10 präsentiert die Sicht der alltäglichen Nutzer\*innen der Webseite, während Abbildung 11 die Sicht des Entwickertools-Modus darstellt.



Abbildung 11 HTML-Struktur

(Chrome-)Entwicklertools sind Werkzeuge für Webentwickler\*innen, die direkt im Browser eingebaut sind. Mithilfe der Entwicklertools (Ctrl + Shift + I) kann der HTML-Code von der Webseite im Browser angezeigt und manipuliert werden. Der Einsatz der Entwicklertools war essenziell für die Entwicklung der Skripts im Scrapingprozess.

Danach wurde im Skript festgelegt, an welchen Stellen und Orten innerhalb der Webseite Daten entnommen werden sollen. Durch das Aufrufen des HTML-Codes mithilfe der Entwicklertools wurde die HTML-Struktur der gesamten Webseite angezeigt. Somit konnten die HTML-Elemente leicht lokalisiert sowie identifiziert werden. Durch das Identifizieren der HTML-Elemente wurde im Skript klar definiert, wo sich die Daten befinden und wo auf diese zugegriffen werden können (*Chrome-Entwicklertools | Chrome DevTools*, o. J.).

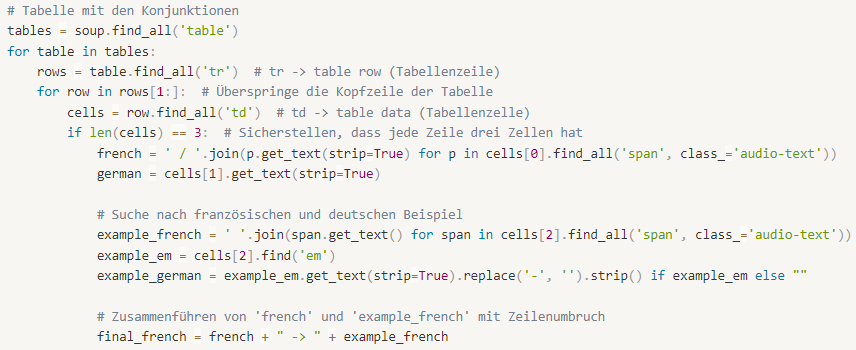


Abbildung 12 Scrapingprozess

In diesem Codeabschnitt wurden die Daten mithilfe von Schleifen Schritt für Schritt in der HTML-Struktur lokalisiert. Der Scraping-Bereich wurde anhand der HTML-Elemente festgelegt. Pro Tabellenzeile befinden sich jeweils vier Wörter (Daten), die extrahiert wurden. Es wurden weitere Schleifen eingebaut, um die Tabellenzeile aufzuteilen und jedes Wort separat zu extrahieren.

* **tables = soup.find\_all('table')**

Es wurde eine Variable namens *tables* erstellt, um nach allen Tabellen im HTML-Code zu suchen. Mit *.find\_all()* wurde die Suchfunktion eingeleitet. Die Suchfunktionen sucht nach dem definierten HTML-Element in den Klammern, welches *'table'* ist. Der englische Begriff «table» steht für Tabelle (Mitchell, 2024, Chapter 2).

* **rows = table.find\_all('tr')**

Es wurde eine weitere Variable namens *rows* erstellt, um alle Zeilen innerhalb der Tabelle zu finden. Durch die Suchfunktion *.find\_all()* wurde nach dem definierten HTML-Element in der Klammer (*'tr'*) gesucht. Der Begriff «tr» bedeutet übersetzt «table row» und steht für Tabellenzeile (Mitchell, 2024, Chapter 2).

* **for row in rows[1:]**

Die eingeleitete «*for*»-Schleife durchsuchte alle Zeilen der Tabelle sowie Tabellenzeilen. Jedoch wurde der Startpunkt «[1:]» der Schleife um eine Zeile versetzt. In Python und generell in der Informatik ist der Startpunkt die 0. Durch die Definition der «1» in der eckigen Klammer wurde die erste Zeile in diesem Fall immer übersprungen. Wie in der Abbildung 10 klar erkennbar ist, handelte es sich bei der ersten Zeile um die Kopfzeile, welche keinen Nutzen im Scrapingprozess bringt. Somit durchlief das Skript alle Zeilen der Tabelle bis auf die Kopfzeile.

* **cells = row.find\_all('td')**

Es wurde eine weitere Variable namens *cells* erstellt, um alle Zellen innerhalb der Tabellenzeilen zu finden. Durch die Suchfunktion *.find\_all()* wurde nach dem definierten HTML-Element in den Klammern (*'td'*) gesucht. Der Begriff «td» bedeutet übersetzt «table date» und steht für Tabellenzelle (Mitchell, 2024, Chapter 2).

* **if len(cells) == 3**

Erneut wurde eine «*if*»-Anweisung eingeleitet, welche sicherstellte, dass jede Zeile innerhalb der Tabelle genau drei Zellen enthält. Die Funktion *len* überprüft jeweils die Länge oder besser formuliert die Anzahl eines Objekts. Somit wurde das Objekt *cells* überprüft. Wie in der Abbildung 10 dargestellt, befinden sich in jeder Tabellenzeile jeweils drei Zellen (*Python Len() Function*, o. J.).

* **french = ' / '.join(p.get\_text(strip=True) for p in cells[0].find\_all('span', class\_='audio-text'))**
* **german = cells[1].get\_text(strip=True)**

Die Variable *french* durchsuchte die erste Zelle der jeweiligen Tabellenzeile. [0] war der Startpunkt der Tabellenzeile und verkörperte die erste Zelle. Das HTML-Tag «span» und die Klasse «audio-text» wurden angegeben, um die Daten korrekt extrahieren zu können.

Die Funktion *.get\_text()* entfernte alle «Tags» aus dem Dokument und gab nur den reinen Text (Unicode-Zeichenfolge) zurück. Die Funktion war sehr hilfreich, da es für den Scrapingprozess nicht relevante Inhalte im HTML-Dokument ausblendete. Mit *strip* wurden zusätzliche Leerzeichen, welche nicht benötigt werden, entfernt (Mitchell, 2024, Chapter 2).

Die Funktion *.join()* fügte die gesammelten Daten aus der ersten Tabellenzelle zu einem einzigen String zusammen. String ist ein Datentyp in Python der Zeichenfolgen, also Buchstaben und Text, darstellt (*Python String Join() Method*, o. J.).

Die Variable *german* durchsuchte die zweite Zelle der Tabellenzeile.

In der Abbildung 10 ist klar erkennbar, dass sich in der ersten Zelle das französische Wort, in der zweiten Zelle das deutsche Wort und in der dritten die beiden Beispiele dazu befanden.

* **example\_french = ' '.join(span.get\_text() for span in cells[2].find\_all('span', class\_='audio-text'))**
* **example\_german = example\_em.get\_text(strip=True).replace('-', '').strip() if example\_em else ""**

Die Variablen *example\_french* und *example\_german* durchsuchten die dritte Zelle und somit war [2] der Endpunkt der Tabellenzeile. In der dritten Zelle befanden sich zwei Datensätze, weshalb noch spezifischere Angaben definiert wurden, um diese beiden separat zu extrahieren. Durch die Spezifikation des «em»-Tags bei der Variable *example\_german* konnten die französischen sowie die deutschen Beispiele separat extrahiert werden (Mitchell, 2024, Chapter 2).

* **final\_french = french + " -> " + example\_french**

Die Variable *final\_french*fügte die beiden Objekte *french* (französische Konjunktion) und *example\_french* (französisches Beispiel) als ein Objekt zusammen.

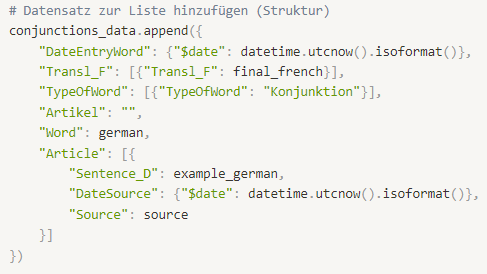


Abbildung 13 Datenstruktur

In diesem Codeabschnitt wurde die Struktur für der extrahierten Daten festgelegt, damit sie später in das richtige Datenformat (JSON) konvertiert und gespeichert werden konnten. Die Datenstruktur wurde von der *schema.prisma*-Datei übernommen. Die *schema.prisma*-Datei gab die Struktur vor und stellte somit ein Abbild der MongoDB-Datenbank dar. Die Variablen, mit denen Daten extrahiert oder generiert wurden, sind in die passenden Felder der Datenstruktur im Skript zugeordnet worden (*Prisma Schema Overview | Prisma Documentation*, o. J.).

* **conjunctions\_data.append()**:

Mit der Funktion *.append()* konnten die gesammelten Daten in die leere Liste (*conjunctions\_data*) hinzugefügt werden, die zuvor (Abbildung 8) erstmalig erstellt wurde (*Python List Append() Method*, o. J.).

Die Datenstruktur des Skripts musste mit der Struktur der *schema.prisma*-Datei übereinstimmt, sonst wäre die Implementierung der Daten in die MongoDB-Datenbank nicht möglich gewesen.

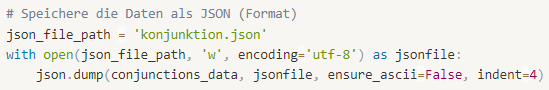


Abbildung 14 Datenspeicherung

In diesem Codeabschnitt wurde eine JSON-Datei erstellt, um die extrahierten Daten zu speichern.

* **json\_file\_path = 'konjunktion.json'**

In der Variable*json\_file\_path*wurde der Dateiname definiert. Die Datei wurde im aktuellen Verzeichnis auf dem lokalen Gerät gespeichert.

* **with open(json\_file\_path, 'w', encoding='utf-8') as jsonfile**

Mit der Funktion *open* wurde das File *konjunktion.json* aus der Variable *json\_file\_path*geöffnet. Der zweite Parameter *'w'*steht für «write», was übersetzt «schreiben» bedeutet (*Python File Write*, o. J.). Der dritte und letzte Parameter gab die Kodierung (*utf-8*) vor. Die Datei wurde im JSON-Format für das Schreiben geöffnet. Dies bedeutet, dass mit diesen Grundvoraussetzung neue Daten in die Datei geschrieben oder hinzugefügt werden können (*Unicode HOWTO*, o. J.).

* **json.dump(conjunctions\_data, jsonfile, ensure\_ascii=False, indent=4)**

Mit der Funktion *.dump()* wurden die gespeicherten Daten in der Liste (*conjunctions\_data*) formatiert und gesichert (*Python JSON*, o. J.).

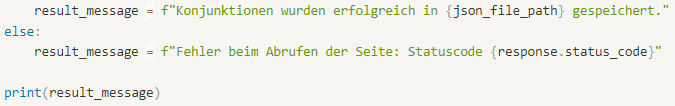


Abbildung 15 Abschlussmeldung

Im letzten Codeabschnitt wurde durch *print(result\_message)* eine Meldung ausgegeben, die den Erfolg oder Misserfolg des Scrapingvorgangs anzeigt. Wenn das Skript erfolgreich durchgeführt werden konnte, wird eine JSON-Datei mit den extrahierten Daten erstellt. Falls das Skript eine andere Meldung ausgibt, wurde ein Fehler begangen.

### JSON-Datei

Die Skripte konnten auf die Quellen zugreifen, die gewünschten Daten extrahieren und korrekt strukturieren. Die extrahierten Daten wurden von den Skripten konvertiert und in eine JSON-Datei gespeichert. Die JSON-Datei konnte auf dem lokalen Gerät im Hauptverzeichnis geöffnet werden. Dank der vorgegebenen Datenstruktur von Prisma wurde der Aufbau im Skript festgelegt, dass die Datenstruktur der extrahierten Daten in der JSON-Datei dieselbe ist wie die in Prisma. Nun waren die extrahierten Daten korrekt strukturiert und im richtigen Datenformat gespeichert. In der Abbildung 16 wird ein Beispiel eines Datensatzes im JSON-Format präsentiert, welches später in die MongoDB-Datenbank implementiert wurde.



Abbildung 16 Datensatz im JSON-Format

### Implementierung

Der Worttyp «Konjunktion» war bereits in der Applikation sowie im Datenbankschema eingebaut, sodass dieser Schritt übersprungen und direkt mit der Implementierung der Daten fortgefahren werden konnte. Eine Verbindung zur MongoDB-Datenbank wurde über das Programm MongoDB-Compass hergestellt. Voraussetzung für eine erfolgreiche Implementierung der Daten war, dass die Daten dieselbe Struktur wie das Datenbankschema vorweisen und entweder dem JSON- oder CSV-Datenformat entsprechen. Da beide Punkte bei der Entwicklung berücksichtigt wurden, konnte durch die «Add Data»-Methode in MongoDB-Compass, die generierte JSON-Datei in die Datenbank eingefügt werden. In der Abbildung 17 wird der Datensatz gezeigt, der sich in der MongoDB-Datenbank befindet.

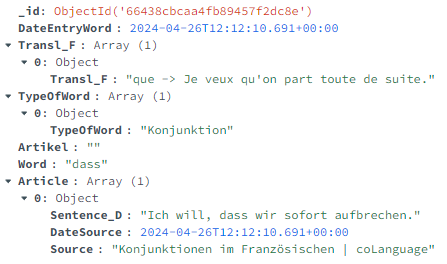


Abbildung 17 Datensatz - MongoDB-Datenbank

Mit der Abfrage *{"TypeOfWord.TypeOfWord": "Konjunktion"}* wurden die gesamte Collection (*deutschs*) nach dem Worttypen «Konjunktion» gefiltert. Mithilfe dieser Filterabfrage in MongoDB-Compass wurde überprüft, ob die Implementierung der Daten erfolgreich war.

### Sprichwörter

Wie bereits erwähnt, wurden für den Scrapingprozess insgesamt drei Skripte entwickelt. Ein weiteres Skript wurde auch für die Lernkategorie «Sprichwörter» erstellt. Da diese Lernkategorie bereits in der Applikation vorhanden war, konnte die Datenstruktur direkt in das Skript übernommen werden. Der Fokus lag somit rein auf dem Sammeln der Daten, da keine weiteren Arbeiten am Frontend oder Backend-Bereich unternommen werden mussten.

Bei der Entwicklung der Skripte wurde grosser Wert auf einen einheitlichen Aufbau gelegt. Dadurch konnte der Code der jeweiligen Skripte wiederverwendet werden, was Zeit sparte und sich als effiziente Vorgehensweise erwies. Aus diesem Grund wird nicht jedes einzelne Skript im Detail vorgestellt, da der gesamte Ablauf und Vorgang bereits im Konjunktionsskript beschrieben wurde.

**Scrapingvorgang**

Dennoch ist es wichtig zu erwähnen, dass sich die Skripte im Scrapingvorgang, also in der Logik, wie Daten von der Webseite extrahiert werden und in der Datenstruktur unterscheiden. Da jede Webseite anders aufgebaut ist, müssen dementsprechend im Skript auch die spezifischen HTML-Angaben angepasst werden, um einen erfolgreichen Scrapingvorgang zu gewährleisten. Zudem ist die Datenstruktur der Lernkarte bei jeder Lernkategorie unterschiedlich strukturiert, weshalb dies im Skript ebenfalls berücksichtigt wird. Diese zwei Punkte unterscheiden sich jeweils bei allen Skripten, während der Rest nach der gleichen Logik aufgebaut ist.

**Datenvergleich**

Bei der Lernkategorie «Sprichwörter» kam ein weiterer kleiner Schritt hinzu, nämlich der Vergleich der aktuellen Daten mit den aus dem Skript extrahierten Daten. Um Redundanzen in einer Datenbank zu vermeiden, musste sichergestellt werden, dass keine doppelten Einträge vorhanden sind. Redundanz bedeutet, dass dieselben Daten doppelt oder mehrfach vorkommen und somit überflüssig sind. Solche überflüssige Daten möchte man in einer Datenbank grundsätzlich vermeiden (*Was ist Redundanz in der Informatik?*, 2022).

Um dies zu prüfen, wurde die gesamte Collection («*sprichworts»*) mit allen Daten aus der Lernkategorie Sprichwörter exportiert. Beim Exportieren der vorhandenen Daten wurde darauf geachtet, dass das JSON-Format verwendet wird. Somit gab es zwei Dateien, eine mit den bereits vorhandenen Daten und eine mit den neu «gescrapten» Daten. Diese beiden Dateien wurden dann mithilfe eines Onlinetools im Browser verglichen, damit es zu keinen Redundanzen in der Datenbank führt (*JSON Diff - The semantic JSON compare tool*, o. J.). Die Daten der «gescrapten» Datei wurden kontrolliert sowie angepasst und anschliessend in die Datenbank hinzugefügt.

## Lernkategorie

Die letzte der insgesamt drei Quellen beinhaltete deutsche Redewendungen. Da in der Applikation keine Lernkategorie für deutsche Redenwendungen vorhanden war, wurde entschieden, eine neue zu erstellen. Bei den Lernkategorien «Wortbedeutungen» und «Sprichwörter» ging es hauptsächlich darum, neue Daten zu sammeln und in die Datenbank zu integrieren, da sowohl das Frontend als auch Backend bereits entwickelt waren. Im Gegensatz dazu wurden bei der letzten Quelle nicht nur neue Daten gesammelt, sondern auch eine komplett neue Lernkategorie erstellt.

Bei der Erstellung der Lernkategorie «Redewendungen» wurde darauf geachtet, dass das Layout, die Struktur und die Logik mit den bereits vorhandenen Lernkategorie übereinstimmen, um die Einheitlichkeit innerhalb der Applikation und der Datenbank gewährleisten zu können.

### Scraping

Wie bei den beiden vorherigen Skripten wurde auch hier wieder auf die Wiederverwendbarkeit des Codes und auf einen einheitlichen Aufbau geachtet. Es wurden die spezifischen HTML-Angaben sowie die Datenstruktur angepasst, um den Scrapingvorgang und die Implementierung der Daten erfolgreich durchführen zu können.

## Neue Kategorie hinzufügen

### Frontend

Ein neuer Lernbereich wurde für das Üben der Redewendungen und ein Erfassungsbereich für das Hinzufügen neuer Redewendungen angelegt.

Für den Entwicklungsprozess im Frontend wurde die Lernkategorie «Sprichwörter» als Vorlage genutzt. Durch die Nutzung dieser Vorlage konnte sichergestellt werden, dass die Verknüpfung von Frontend und Backend übereinstimmten. Es war wichtig, dass die neuen Daten nicht nur in die Datenbank implementiert, sondern auch im Frontend richtig angezeigt und präsentiert werden. Dabei wurde darauf geachtet, dass das Layout einheitlich ist und das CSS der gleiche Stil wie bei den anderen Lernkategorien verfolgte.

Zusätzlich war es essenziell, die Logik und Funktionen mit einzubauen, damit die Daten nicht nur angezeigt, sondern auch geübt werden können. Anpassungen, die unternommen wurden, betrafen die Datenstruktur, bestimmte Pfade, Funktionen und Verknüpfungen. Dasselbe Prinzip wurde natürlich auch für die Worterfassung angewendet, wobei dort auch die «Sprichwörter» als Hilfsmittel dienten. Nach der Entwicklung der Worterfassung wurde dieser Bereich direkt getestet und mit der Datenbank (Backend) verglichen, um sicherzustellen, dass die eingegeben Daten einwandfrei gesendet und in die Datenbank korrekt aufgenommen wurden.

### Backend

Um die gesammelten Daten in der Datenbank erfolgreich zu speichern, musste eine neue Collection erstellt und zusätzlich mit der korrekten Datenstruktur definiert werden. In MongoDB-Compass wurde eine neue Collection namens «*redewendungs*» angelegt. In dieser Collection wurden die gesammelten Daten aus dem Skript eingefügt und gespeichert.

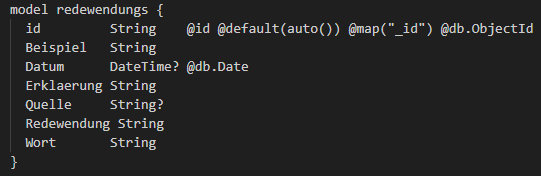


Abbildung 18 Datenstruktur in Prisma

Im Vorfeld wurde in Prisma ein neues Modell («model») namens «*redewendungs*»erstellt*.* Innerhalb dieses Modells wurde die Datenstruktur für die neue Lernkategorie «Redewendungen» festgelegt. Diese Struktur wurde in der bereits vorhandenen *schema.prisma-*Datei definiert. Das Schema aus der *schema.prisma-*Datei konnte direkt in die MongoDB-Datenbank übertragen werden, wodurch keine manuellen Schritte in der MongoDB-Datenbank mehr notwendig waren. Das Schema von Prisma (*schema.prsima-*Datei) wurde automatisch in die Datenbank aufgenommen. Wie bei den Skripten und im Frontend wurde auch bei der Struktur auf die Einheitlichkeit geachtet.

## Lernkarten

Jede Lernkategorie verfügt über eine Lernkarte mit einer Vorder- und Rückseite. Da es unterschiedliche Lernkategorien gibt, variierten auch die Textinhalte, Aufgabenstellung und Lösungen. Daher wurden einige Lernkategorien mehr angepasst als andere.

### Visuelle Anpassung

Jede Anpassung wurde individuell vorgenommen und konnte nicht direkt auf die anderen Lernkategorien übertragen werden. Betroffen waren die Lernkategorien «Wortbedeutungen», «Präpositionen» und «Präposition & Verben». Grössere Textabschnitte oder Inhalte mit vielen Absätzen wurden hauptsächlich durch Aufzählungszeichen umstrukturiert, um einen bessere und optimalere visuelle Übersicht zu gewährleisten. Weitere Anpassungen fanden in der Lernkategorie «Präpositionen» statt, wobei die Aufgabenstellung auf der Vorderseite und die Präsentation der Lösung auf der Rückseite angepasst sowie verändert wurden.

# Ergebnisse

## Web-Scraping-Skripts

Im Scrapingprozess konnten erfolgreich drei Web-Scraping-Skripte erstellt und gezielt eingesetzt werden. Die gewünschten Daten von den jeweiligen Webseiten wurden mithilfe der entwickelten Skripte extrahiert und korrekt in eine JSON-Datei umgewandelt. Die Implementierung der lokal gespeicherten JSON-Dateien in die MongoDB-Datenbank verlief einwandfrei. Durch eine Add-Funktion in MongoDB-Compass konnten diese hinzugefügt werden.

Insgesamt wurden 1570 neue Datensätze gesammelt und in die MongoDB-Datenbank implementiert. Darunter befanden sich 32 Konjunktionen, 560 Sprichwörter und 988 Redewendungen. Die Lernkategorie «Wortbedeutungen» hatte zu Beginn 1908 Datensätze und jetzt 1940. Die Lernkategorie «Sprichwörter» verzeichnete zu Beginn 60 Datensätze und nach der Implementierung 620. Die Lernkategorie «Redewendungen» wurde neu erstellt, somit hatte sie zu Beginn keine Daten vorhanden und enthält nun 988 Datensätze in der Datenbank.

Dies bietet den Benutzer ein völlig neues Lernerlebnis. Durch neue Daten werden auch gleichzeitig neue Lernoptionen geschaffen. Benutzer\*innen können somit mehr und länger lernen, wenn sie das möchten, da durch die Skripte das Datenbankvolumen erweitert wurde.

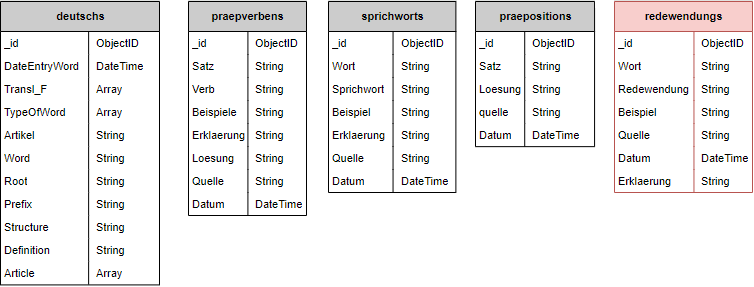


Abbildung 19 redewendungs

Das Erstellen einer neuen Collection und der Datenstruktur verlief reibungslos. In Prisma wurde ein neues Modell («model») angelegt, welches die neue Collection sowie die Datenstruktur des Lernbereichs festlegte.

Das Schaffen einer komplett neuen Lernkategorie bietet den Benutzer\*innen eine zusätzliche Lernoption. Ursprünglich gab es vier verschiedene Lernbereiche, durch die Entwicklung der Lernkategorie «Redewendung» wurde dieser Bereich erweitert. Diese Erweiterung bringt mehr Spielraum und Abwechslung für Benutzer\*innen während ihrer Lernsession.

Abbildung 22 Lernkarte - Präpositionen

In der Lernkategorie «Präpositionen» wurde die Vorder- und Rückseite der Lernkarte angepasst. Die Aufgabenstellung sowie die Lösung der gesuchten Präpositionen wurden untereinander aufgelistet.

Diese Änderungen tragen zu einer klareren Übersicht bei und steigern gleichzeitig die Benutzerfreundlichkeit der Full-Stack-App.

# Ausbau

Es gibt verschiedene Möglichkeiten, dieses Projekt weiterzuentwickeln. In diesem Abschnitt wird ein Ausblick für Weiterentwicklungsmöglichkeiten beschrieben.

### Erweiterung der Skripts

Durch die Web-Scraping-Technologie könnten weitere, spezifischere Skripte erstellt werden, um neue Daten sammeln. Diese Skripte könnten angepasst oder sogar optimiert werden, um die die Effizienz und die Geschwindigkeit für die Datensammlung zu steigern. Ein Ziel wäre es, verschiedene Quellen zu definieren, die wiederum neue Lerninhalte generieren.

### Erweiterung der Datenbank

Eine weitere sinnvolle Erweiterung wäre das Anlegen neuer Collections in der MongoDB-Datenbank. Dadurch könnten im Frontend gleichzeitig neue Lernkategorien gebildet werden, was den Benutzer\*innen noch mehr Auswahl bei Lernkategorien bieten würde. Gleichzeitig könnten erneut neue Daten mittels Web-Scraping gesammelt und in die neu angelegten Collection eingefügt werden.

### Neue Lernfunktionen

Eine weitere Ausbaumöglichkeit wäre im Frontendbereich, indem die Lerninhalte der Lernkategorien erweitert werden. Die Aufgabenstellungen gewisser Lernkategorien könnte angepasst werden, beispielsweise durch die Einführung eines neuen Lernmodus mit Multiple-Choice oder anderen Aufgabenstellungen, bei denen die Benutzer\*innen ihre Antworten eingeben müssen. Somit könnte den Benutzer\*innen eine Abwechslung vom klassischen Karteikarten-Prinzip geboten werden.

### Erweiterung der Benutzeroberfläche

Im Frontendbereich könnten interaktive Elemente eingebaut werden, um die Lernsession und den Lernprozess für die Benutzer\*innen attraktiver zu gestalten. Die interaktiven Elemente könnten beispielsweise dann eingesetzt werden, wenn eine persönliches Ziel erreicht wird oder der Lernstapel abgeschlossen ist. Eine zusätzlich Erweiterung wäre auch ein sogenannter «Fortschrittstracker», der die Benutzer\*innen informiert, wie weit sie schon sind, wie oft und wie lange sie in einem bestimmten Zeitraum gelernt haben. Fortschritt motiviert und durch einen Fortschrittstracker könnte dies festgehalten und überwacht werden. Zudem könnte im Frontend das Nutzerprofil erweitert werden, sodass die Benutzer\*innen Anpassungen an Ihren Profilen vornehmen können.

### Verbesserungspotenzial

Obwohl ich einen Zeitplan für meine Bachelorthesis und für dieses Projekt festgelegt hatte, hätte dieser spezifischer und detaillierter sein können. Ein detaillierterer Zeitplan hätte eine bessere Verteilung der Arbeiten ermöglicht. Zudem hätte mein Zeitmanagement im Verlauf des Projekts generell besser sein können, da der Arbeitsaufwand gegen Ende eher mehr zunahm. Dennoch fand ich die allgemeine Planung meiner Arbeitsschritte sehr sinnvoll und gewisse externe Faktoren lassen sich im Vorhinein nicht wirklich einplanen.

Ausprobieren, hat mir grossen Spass bereitet.

# Verwendung von KI-gestützten Tools

|  |  |  |
| --- | --- | --- |
| **ChatGPT – 4o** | | |
| **S. 2 Zeile 1 - 37** | Übernommen aus ChatGPT  (22.05.2024) | Prompt: Bitte korrigiere mir das ganze Dokument auf grammatikalische und orthografische Fehler. |
| **S.6, Zeile 1 - S.48, Zeile 27** | Übernommen aus ChatGPT  (22.05.2024) | Prompt: Bitte korrigiere mir das ganze Dokument auf grammatikalische und orthografische Fehler. |

# Anleitung

Um die Full-Stack-App starten zu können, muss der entsprechende Ordner der Full-Stack-App verfügbar sein. Sie müssen entweder den Ordner, in dem sich die Full-Stack-App befindet, von GitHub herunterladen oder klonen. Die Full-Stack-App wird über GitHub verwaltet...

**Grundvoraussetzung**

1. Download- und Zugangsrechte für Full-Stack-App anfordern (falls der Ordner schon zur Verfügung steht, kann dieser Schritt übersprungen werden und zum Setup weitergegangen werden)
2. Mit Einladungslink das komplette Repository (<https://github.com/callmemarcleo/language2-workspace>) herunterladen
   1. Auf den Link gehen › grüner Button "Code" anwählen › ZIP herunterladen
   2. Personen welche mehr Erfahrung vorweisen, können das Repository einfach Klonen und selbst starten
3. File Explorer (Windows) öffnen und der heruntergeladene ZIP-Ordner (von Github-Repository) «entzippen»

## Setup

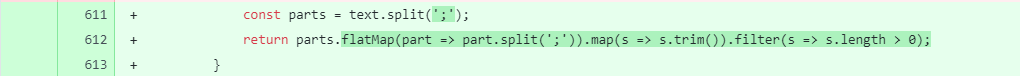
* + - 1. Die aktuellste Version von Node.js (<https://nodejs.org/en>) herunterladen.
      2. Der heruntergeladene Ordner (language2-workspace) an einem bestimmten Ort speichern.
      3. Im File Explorer (Windows) in diesen Ordner navigieren. Sie müssen sich im Ordner befinden
      4. Dann in der Explorerleiste (dort wo der Pfad angegeben ist) bitte rein klicken (alles sollte jetzt blau markiert sein) und "cmd" in die Explorerleiste eingeben und Enter drücken
      5. Es öffnet sich die Eingabeaufforderung (schwarzes CMD-Fenster). In der Eingabeaufforderung sollte derselbe Pfad angezeigt werden wie vorhin im File Explorer. Es ist sehr wichtig, dass Sie sich im richtigen Pfad und Verzeichnis befinden, ansonsten können die weiteren Schritte nicht durchgeführt werden. Somit sollte jetzt ihr Pfad in der Eingabeaufforderung auf "\language2-workspace" enden. Wenn das der Fall ist, können Sie mit Schritt 6 weiterfahren. Wenn nicht müssen sie es nochmals über den File Explorer mit dem Befehl "cmd" in der oberen Explorerleiste probieren.
      6. In der Eingabeaufforderung
      7. npm install + ENTER drücken:› ladet alle notwendigen Pakete herunter die in diesem Projekt benötigt werden
      8. npx prisma generate + ENTER drücken: stellt Verknüpfung zu Prisma Studio her
      9. Ersetzen Sie in der Datei pages/api/auth/lib/mongodb.js die Variable «MONGODB\_URI» durch «DATABASE\_URL»

Ein Bild, das Text, Screenshot, Software, Webseite enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

10. In der Datei pages/deutsches.js den Split-Parameter auf ';' ändern.



11. in den Dateien pages/Worterfassung.js und pages/index.js bei jeder Auswahlbox style={{ width: '300px', height: '150px' }} hinzufügen.



12. [MongoDB Compass Download (GUI) | MongoDB](https://www.mongodb.com/try/download/compass) herunterladen, installieren und öffnen.

13. Die Datenbank-URL aus .env kopieren und in MongoDB Compass einfügen, um sich zu verbinden.

14. Einen Query erstellen, um alle reflexiven Verben mit zusätzlichen Einträgen zu filtern.

Ein Bild, das Text, Webseite, Schrift, Software enthält.

Automatisch generierte Beschreibung

15. Die gefilterten Daten exportieren und eine Sicherungskopie (Backup) erstellen.

Erstelle eine kopie(backup)

16. Die angezeigten Daten mit dem Query aus der Datenbank löschen.

17. Die exportierte JSON-Datei in ChatGPT hochladen und prüfen lassen, dass nur ein Verbtyp enthalten ist. ChatGPT soll den passenden Typ festlegen.

18. Die überarbeitete JSON-Datei von ChatGPT herunterladen und in die Datenbank importieren.

19.Tailwind material installieren und den Import in pages/deutsch.js einfügen.

Ein Bild, das Screenshot, Schrift, Text, Reihe enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

20. Mit der Funktion dialog ein Popup erstellen, das den Filter enthält.

21. npm run dev + ENTER drücken: startet die Anwendung

1. Jetzt sollte in diesem Fenster einen URL namens http://localhost:3000 erscheinen
2. Link kopieren und im Browser einfügen (die Adresse könnte man auch manuell im Browser eingeben) und somit öffnet sich die Anwendung
3. Einloggen und dann sind sie angemeldet und können loslegen

## 