Webengineering
Projektbericht

Fachhochschule Bielefeld
Campus Minden
Studiengang Informatik
Prof. Dr.-Ing. Grit Behrens
B. Sc. Florian Fehring

Beteiligte Personen:

Name	**Matrikelnummer**
Daniel Fast	1217924
Marcel Sander	1260933
Willi Schäfer	1051182
Amer Shurbaji	1138155
Abdurakhman Vaysert	1228838

03. Februar 2023

1 Einleitung

1.1 Beschreibung

Viele Daten aus dem Internet werden im JSON Format ausgegeben. Dieses JSON Format ist nicht immer sofort und einfach verständlich für jeden Nutzer, außerdem können Datenbanken nicht ohne weiteres jede JSON-Datei einlesen. Die JSON-Dateien enthalten oft weitere Informationen, die nicht für jeden Nutzer brauchbar sind oder die nicht in jede Datenbank eingefügt werden können.
Damit die Daten aus den JSON-Dateien verständlicher sind und auch in Datenbanken eingefügt werden können, soll ein Tool entwickelt werden, welches die JSON-Dateien einliest und deren Informationen sinnvoll in eine leserliche Tabelle umwandelt. Diese Tabelle kann dann auch in Datenbanken gespeichert werden.

1.2 Motivation

Das zu entwickelnde Tool wird als Webservice angeboten. Es kann JSON-Dateien entweder als reinen Text, als Datei oder von einer Schnittstelle einlesen. Die Informationen aus der Datei werden eingelesen und als Tabelle angezeigt. Diese Tabelle ist übersichtlich und für jeden Nutzer sofort ersichtlich. Zusätzlich kann eine Datenbank, welche die ausgegebenen Spalten enthält, die Informationen abspeichern.
Die zu verarbeitenden Informationen können vor dem Parsen durch eine Config-Datei angegeben werden, damit nur die benötigten Informationen in eine Tabelle umgewandelt werden. Die Config speichert Informationen zur Verarbeitung der JSON-Datei, wie die Zieldatenbank, die relevanten Attribute aus der JSON-Datei und die Quelle der Daten. Zusätzlich kann die Config-Datei genutzt werden, um sich wiederholende Arbeitsprozesse, wie z.B. der tägliche Abruf der Wetterdaten, zu verarbeiten.
Bei Nutzung des Tools ohne Config, wird die gesamte JSON-Datei zur Veranschaulichung in einer Tabelle dargestellt.

Der Fokus des Projekts liegt auf der sinnvollen Gestaltung der Benutzeroberfläche sowie dem Import von unterschiedlichen JSON-Dateien. Es sollen alle möglichen JSON-Dateien korrekt eingelesen, in eine Tabelle umgewandelt und in einer Datenbank abgespeichert werden.

2 Projektorganisation

2.1 Annahmen

Für die Projektorganisation und -durchführung verwenden wir verschiedene Programme.
Zum Verfassen der Spezifikation nutzen wir Google Docs. Hier können wir einfach verteilt und gleichzeitig die Spezifikation bearbeiten.
Zur Organisation verwenden wir Discord und die Spezifikation als Vorlage. Zur verteilten Entwicklung nutzen wir GitHub. Das Projekt wird von GitLab gecloned und in einem eigenen Repository entwickelt und zum Schluss wieder in das ursprüngliche Repository gepusht und gemerged. Backend und Webfrontend besitzen je ein eigenes GitLab Repository. In Postman wird die REST-API entwickelt. Das Backend wird in IntelliJ IDEA entwickelt, da diese IDE viele nützliche Funktionen zur Entwicklung von Server und Client bietet. Die Entwicklung von dem Webfrontend wird in Visual Studio Code durchgeführt. Der Server hostet einen Payara Server, welcher auf Java basiert und mit Java 11 kompatibel ist. Für das Webfrontend wird das Framework SWAC genutzt.
Wir achten bei der Entwicklung auf Softwarequalität und Erweiterbarkeit. Der Code wird kommentiert, bei Java auch mit JavaDoc. Die Klassen sind gruppiert in unterschiedliche Packages.

2.2 Verantwortlichkeiten

Leitung

Übernimmt die Leitung des Projekts als Scrum Master und Product Owner. Führt die Meetings an, schreibt Protokoll, sammelt Ideen und trifft die letzte Entscheidung über potenzielle Features.

Frontend-Entwickler

Entwickelt grafische Benutzerschnittstellen, insbesondere das Layout einer Anwendung. Erstellen von GUI-Mockups.

Backend-Entwickler

Implementiert die funktionale Logik der Anwendung. Hierbei werden zudem diverse Datenquellen und Frameworks integriert und für die Anwendung bereitgestellt. Erstellt Domain Model und ER-Modell.

Rollenzuordnung

Name	**Rolle**
Daniel Fast	Leitung, Backend-Entwickler
Marcel Sander	Frontend-Entwickler
Willi Schäfer	Backend-Entwickler
Amer Shurbaji	Frontend-Entwickler
Abdurakhman Vaysert	Backend-Entwickler

2.3 Grober Projektplan

Aufwandsabschätzung

Der Aufwand wurde durch den Workload des Praktikums im Modul Webengineering aus dem Modulhandbuch geschätzt.

Phase	**Relative Entwicklungszeit**	**Absolute Entwicklungszeit in Std. pro Person**
Definition	10 %	30
Entwurf	25 %	75
Codierung	35 %	105
Testen	30 %	90

Meilensteine

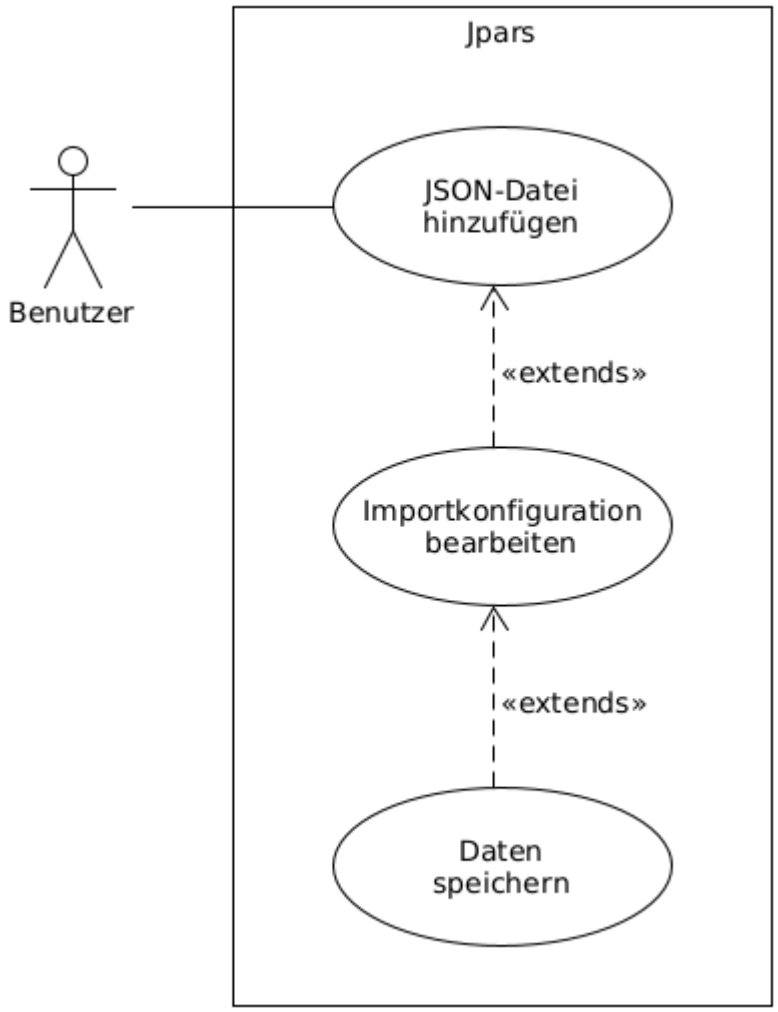
Meilenstein	**Aufgaben**
1	• Funktionierende Umgebung • Vertraut machen mit Umgebung • Detailliertes Einarbeiten in die Frameworks und Umgebung • REST-Schnittstelle definieren • Hello World Programm, einfache Datenübertragung von Frontend zu Backend zu Datenbank • Mehrere JSON-Objekte in verschiedenen Spalten in der Datenbank einfügen
2	• Config erstellen • Config speichern und laden • Frontend kann JSON-Datei durch mind. eine Möglichkeit (Text/Drag&Drop/URL) an das Backend übertragen • JSON-Datei mit einer Verschachtelungsebene in die Datenbank schreiben und lesen • Backend mit Parser, der die JSON-Datei lesen kann • Frontend mit zwei Fensterns, links die JSON und Config Auswahl, rechts die entstehende Tabelle
3	• Frontend erweitern und benutzerfreundlicher gestalten • Frontend zeigt die gelesenen Daten aus der JSON-Datei als Tabellen an • Config-Datei durch Fenster in der Weboberfläche editierbar, Attribute können ausgewählt werden • JSON-Dateien auf unterschiedliche Arten einlesen und verarbeiten können • Prüfung, ob Daten in die Datenbank eingefügt werden können • Datenbankstruktur kann angepasst / erweitert werden • Soll-Kriterien implementieren
Abgabe	• Testen • Puffer um Arbeit aufzuholen

3 Anforderungen

3.1 Funktionale Anforderungen

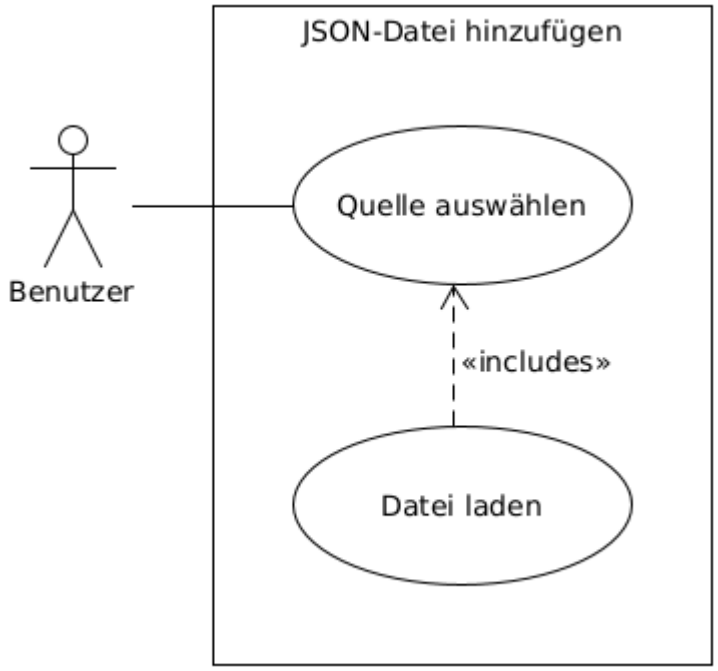
3.1.1 Allgemeine Funktionsweise

Use-Case-Diagramm zu der allgemeinen Funktionsweise der JPars-Software. Grundlegend ist die Software in drei Bereiche unterteilt, die näher erläutert werden.



3.1.2 JSON-Datei hinzufügen

Der Benutzer fügt dem System eine JSON-Datei hinzu, die verarbeitet werden soll.



Quelle auswählen:

(User Story ID 2)
Der Benutzer wählt eine Quelle für die JSON-Datei aus.

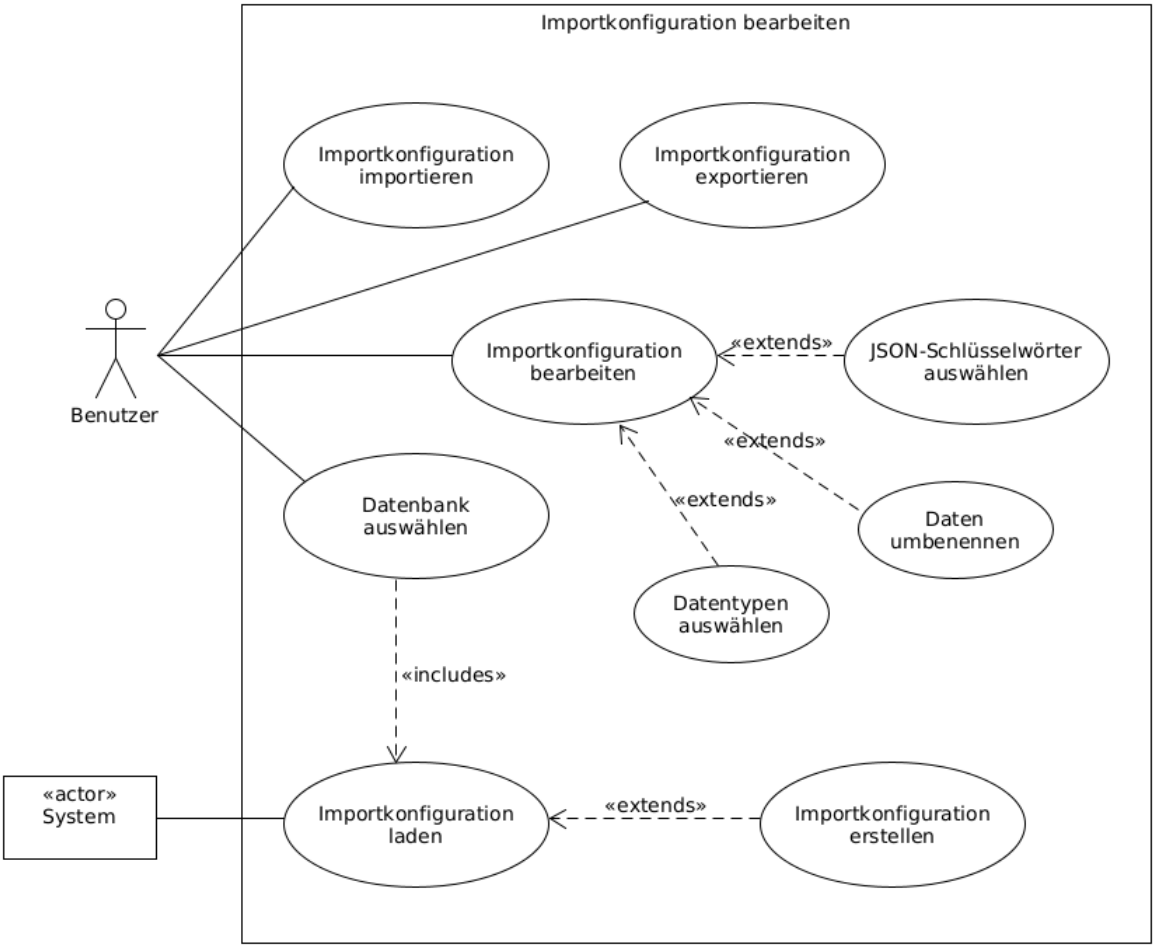
- Lokale Datei mit Drag & Drop reinziehen
- Lokale Datei über einen Datei-Pfad hochladen
- Dateiverweis als URL angeben
- Kopieren als Text

Datei laden:

(User Story ID 1)
Nach einer Bestätigung des Benutzers wird die JSON-Datei in das System geladen.

3.1.3 Importkonfiguration bearbeiten

Einstellungsmöglichkeiten für die Verarbeitung der JSON-Datei.



Importkonfiguration importieren:

(User Story ID 7)
Der Benutzer lädt eine lokal gespeicherte Importkonfigurationsdatei hoch.

Importkonfiguration exportieren:

(User Story ID 6)
Der Benutzer speichert die Importkonfigurationsdatei in der Datenbank.

Importkonfiguration bearbeiten:

(User Story ID 8)
Der Benutzer ändert eine Importkonfigurationsdatei und diese Änderung wird vom System als globale Importkonfigurationsdatei gespeichert.

- **JSON-Schlüsselwörter auswählen**

  (User Story ID 3)
  Der Benutzer kann auswählen, welche Daten der JSON-Datei in der Datenbank gespeichert werden.

- **Daten umbenennen**

  (User Story ID 4)
  Der Benutzer kann bestimmen, wie die Tabellen und Spalten in der Datenbank benannt werden.

- **Datentypen auswählen**

  (User Story ID 5)
  Der Benutzer kann für Daten aus der JSON-Datei Datentypen wählen, damit das Datenformat in der Datenbank übernommen wird.

Datenbank auswählen:

(User Story ID 9)
Der Benutzer wählt eine Datenbank für die Speicherung der JSON-Datei aus oder legt eine neue an.

Importkonfiguration laden:

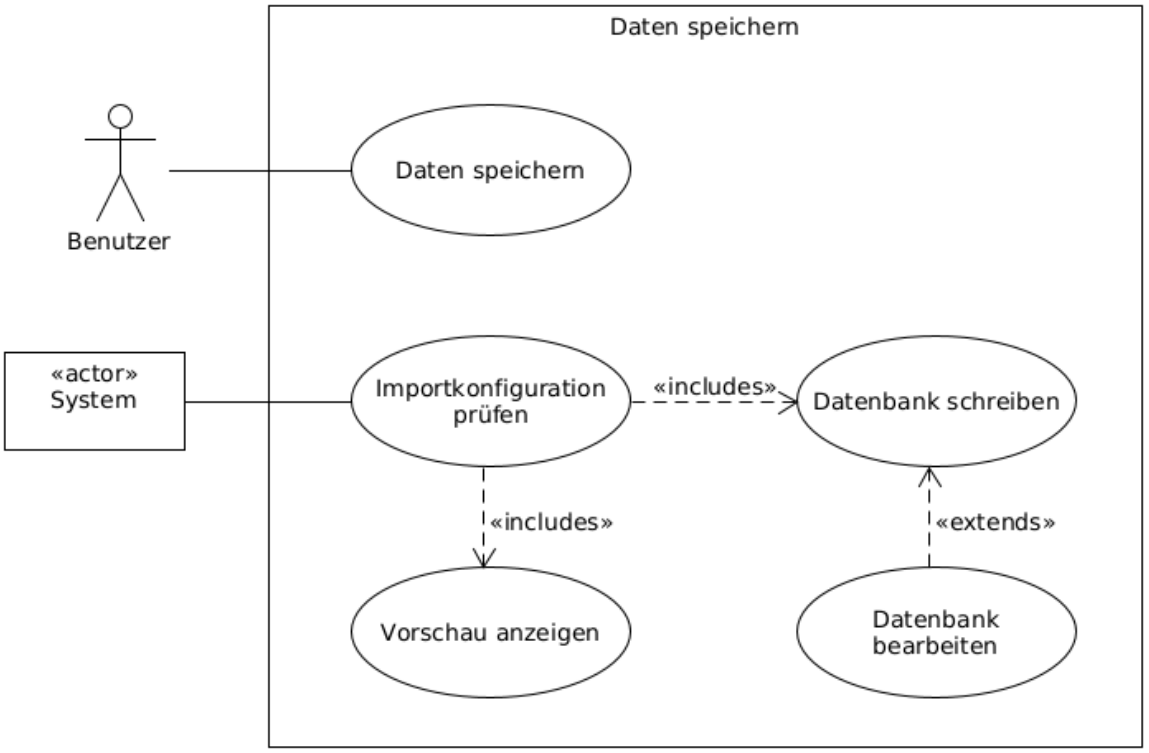
(User Story ID 10)
Nach der Auswahl der Datenbank durch den Benutzer lädt das System die zugehörige globale Import Konfigurationsdatei.

Importkonfiguration erstellen:

(User Story ID 10)
Wenn der Benutzer eine neue Datenbank anlegt, wird eine globale Importkonfigurationsdatei für diese Datenbank vom System erstellt.

3.1.4 Daten speichern

Sicherung der Daten aus der JSON-Datei in einer Datenbank.



Vorschau anzeigen:

(User Story ID 12)
Das System zeigt dem Benutzer, wie die Tabellen in der Datenbank nach dem Konfigurieren aussehen werden.

Daten speichern:

(User Story ID 11)
Der Benutzer bestätigt die Änderung der Datenbank.

Importkonfiguration prüfen:

Das System prüft, ob es beim Schreiben in die Datenbank Konflikte gibt.

Daten schreiben:

(User Story ID 11)

Das System benutzt die Importkonfigurationsdatei, um die Daten aus der JSON-Datei in die Datenbank zu schreiben.

Datenbank bearbeiten:

Das System kann mögliche Strukturänderungen, resultierend aus der Änderung der Importkonfigurationsdatei, auf die Datenbank übertragen.

Mögliche Änderungen:

- Tabellen hinzufügen und löschen
- Tabellen- und Spaltennamen ändern
- Datensätze ändern, löschen und hinzufügen
- Datentypen ändern

3.2 Funktionale Anforderungen im Detail

ID	In meiner Rolle als…	…möchte ich…	…, so dass…	Erfüllt, wenn…	Prio-rität
1	Benutzer (**Allgemein**)	eine JSON-Datei auswählen	ich sie hochladen kann	die Datei im System ist	Muss
2	Benutzer (**Allgemein**)	die JSON-Datei aus verschiedenen Quellen hochladen	ich nicht umwandeln muss	viele Eingabemethoden unterstützt sind	Soll
3	Benutzer (**Allgemein**)	auswählen welche Daten ich aus der JSON-Datei speichere	ich die Datensätze reduzieren kann	JSON-Schlüsselwörter ausgewählt werden können	Muss
4	Benutzer (**Allgemein**)	Tabellen und Spalten eigene Namen geben können	der Datenzugriff auf die Datenbank angepasst werden kann	JSON-Attribute umbenannt werden können	Soll
5	Benutzer (**Allgemein**)	Datentypen für die Daten wählen	die spätere Datenverarbeitung einfacher ist	Datentypen für die Datenbank setzen können	Soll
6	Benutzer (**Allgemein**)	eine Importkonfigurations datei erstellen	ich Einstellungen wiederverwenden oder an Dritte weiter geben kann	Importkonfigurations datei Export	Muss
7	Benutzer (**Allgemein**)	eine Importkonfigurations datei laden	ich gespeicherte Einstellungen wieder benutzen kann	Importkonfigurations datei Import	Muss
8	Benutzer (**Allgemein**)	JSON-Schema ändern können	bei Änderungen die Datenbank nicht neu aufsetzen muss	Änderung der Importkonfigurations datei kann auf die Datenbank angewendet werden	Soll
9	Benutzer (**Allgemein**)	Eine Datenbank auswählen können	weitere Daten hinzufügen kann	Datenbank kann ausgewählt werden	Muss
10	Benutzer (**Allgemein**)	Importkonfiguratione n global speichern können	andere Benutzer die selbe Importkonfiguration leicht anwenden können	globale Importkonfigurations datei benutzen	Muss
11	Benutzer (**Allgemein**)	die JSON-Daten in eine Datenbank speichern	ich die Daten wiederverwenden kann	Daten in Datenbank gespeichert	Muss
12	Benutzer (**Allgemein**)	Tabellenstruktur angezeigt bekommen	ich sehen kann, wie die Daten aufgeteilt werden	Tabellen-Vorschau wird angezeigt	Muss

3.3 Nicht-funktionale Anforderungen

Functional Suitability

Die funktionalen Anforderungen, die als “Muss” markiert sind, müssen erfüllt werden.

Performance Efficiency

Importieren der JSON-Datei und Config darf maximal 2 s dauern, bei sehr großen JSON-Dateien entsprechend länger. Die Dauer der Anforderung einer Ressource durch eine URL darf ebenfalls maximal 2 s dauern, bei sehr großen JSON-Dateien entsprechend länger.

Compatibility

Die Verwendung des Webservices soll auf möglichst vielen Betriebssystemen gewährleistet sein, da der Service durch die Verwendung eines Webbrowsers erreichbar ist. Die Betriebssysteme müssen aktuelle und gängige Browser wie Google Chrome Version 95, Firefox Version 94 oder Safari Version 15 unterstützen.

Usability

- Webanwendung mit einheitlichem Design
- Nutzung auf Endgeräten mit beliebiger Bildschirmauflösung
- Einfachere Benutzung durch intuitiv gestaltete Benutzeroberfläche
- Fehleingaben des Benutzers werden durch Abfragen unterbunden
- Warnhinweise bei fehlerhaften Daten und Nutzung

Reliability

Gewährleistung der Diensterreichbarkeit durch Aufbau einer redundanten Serverstruktur.

Security

- Passwörter werden verschlüsselt gespeichert (SHA-256) (Optional)
- Es müssen wöchentliche Backups der Daten vorgenommen werden

Maintainability

- Modularer Aufbau zur Minimierung von Anpassungen
- Verwendung von Standard-Bibliotheken

Portability

Der Dienst kann ohne Installation mithilfe eines Browsers verwendet werden.

3.4 Mockups

Übersicht mit Drag&Drop

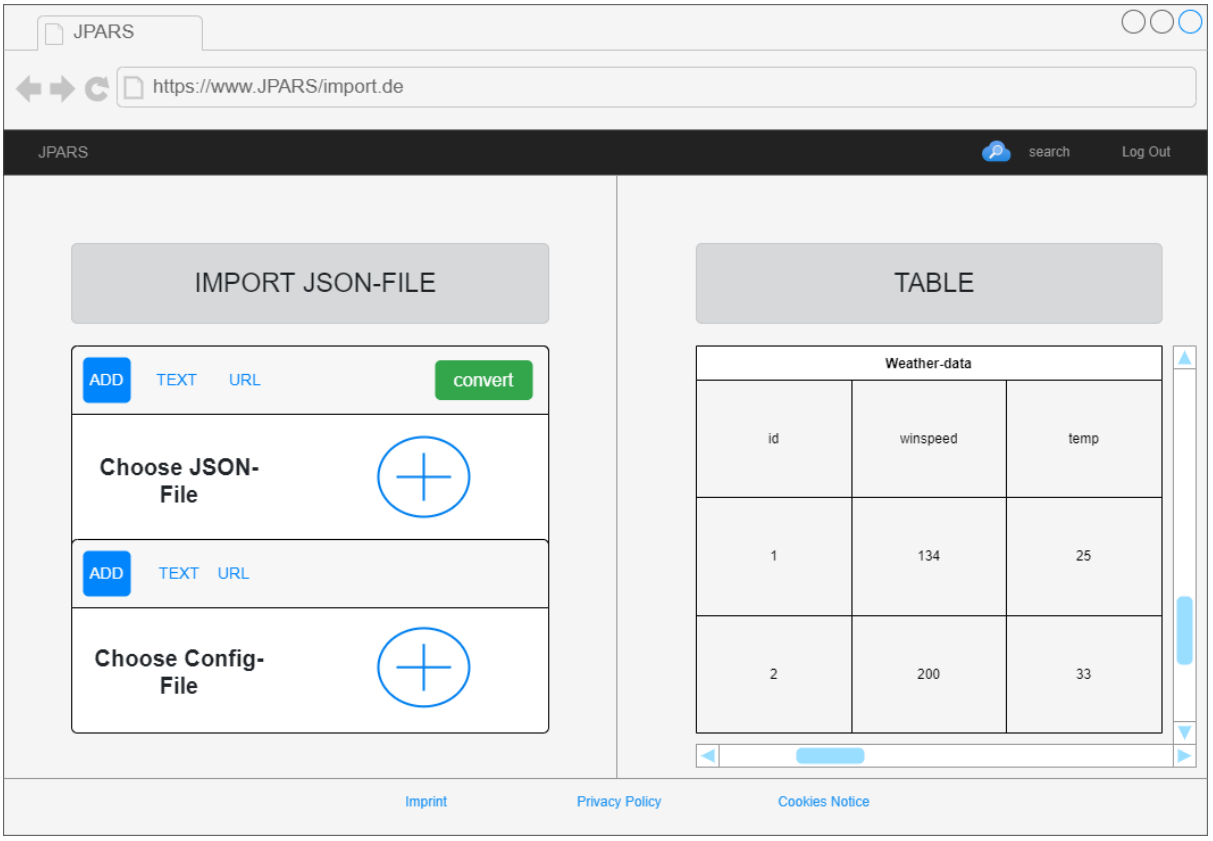


Abbildung 1: Die Übersicht über die Importfunktion mit Drag&Drop Funktion

Auf der Startseite (zu sehen in Abbildung 1) kann der Nutzer die JSON-Datei sowie die Config-Datei mit Drag & Drop importieren. Auf der rechten Seite hat der Nutzer die Vorschau von importierten Daten in einer Tabelle dargestellt. Die Suche- und Logout-Funktion sind optional, diese beiden Funktionen sind optional und werden nachher, wenn genug Zeit vorhanden ist, implementiert.

Import der Daten durch Text

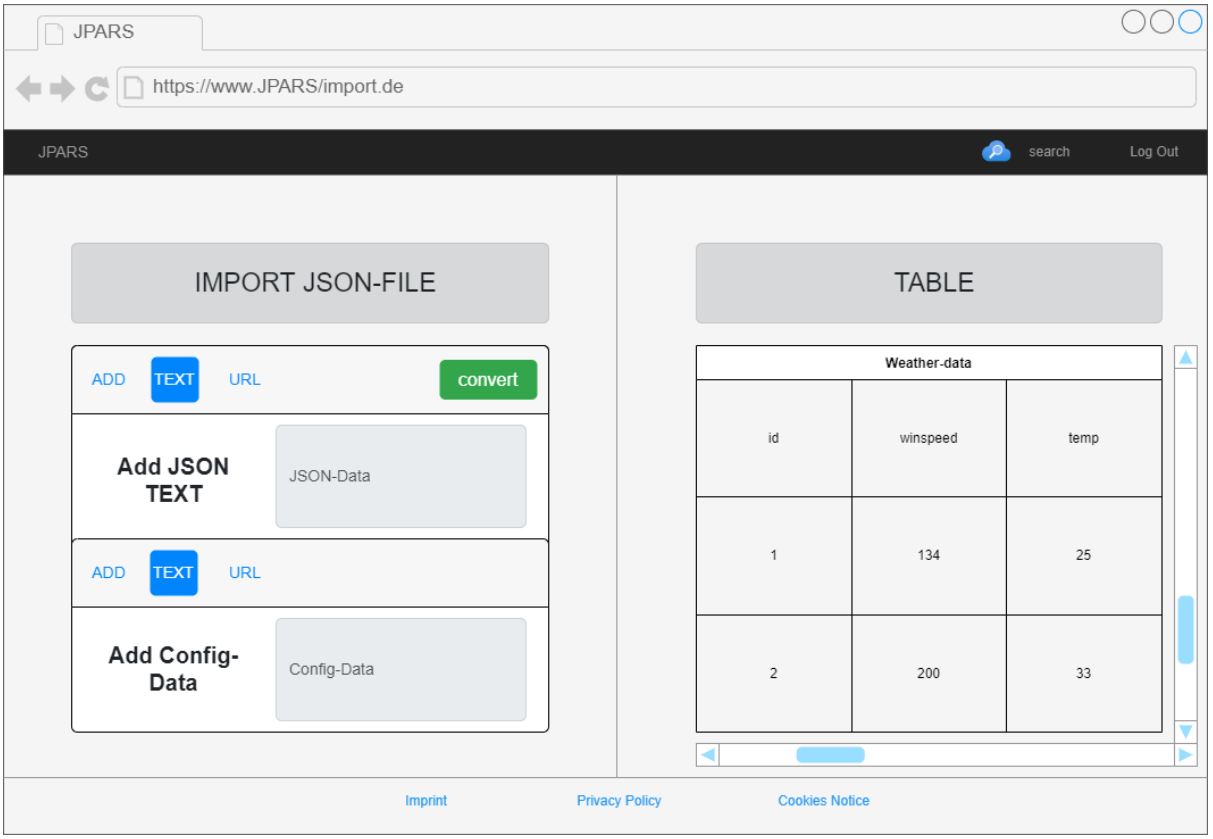


Abbildung 2: Die Übersicht über die Importfunktion mit Text Eingabe

Eine andere Möglichkeit, JSON-Daten und Config-Dateien hinzuzufügen, ist die Eingabe als Text in die Textbox, mittels der üblichen “Copy and Paste" Methode oder Daten direkt als Text in das Textfeld zu schreiben. Sowohl JSON-Daten als auch Config-Daten können vom Nutzer eingefügt und parallel bearbeitet werden.

Import der Daten durch URL

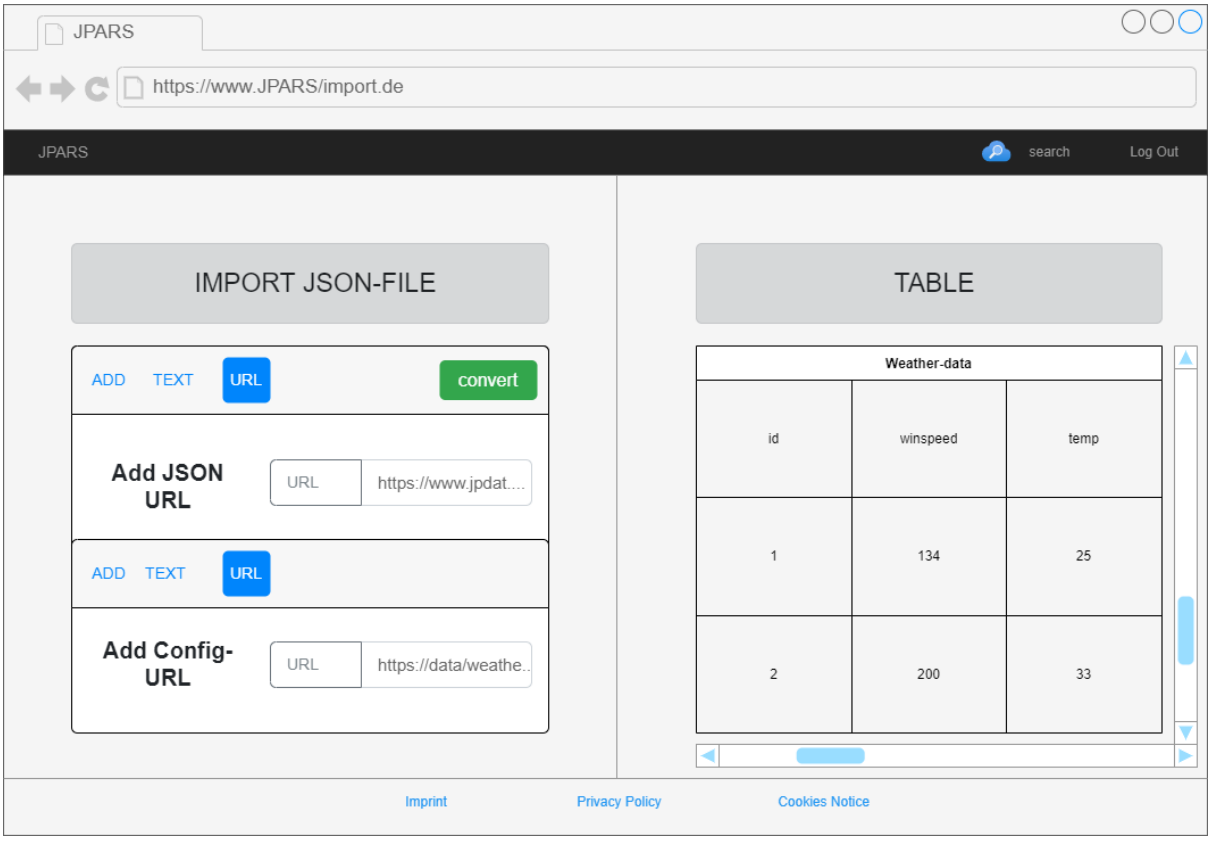


Abbildung 3: Die Übersicht über Importfunktion mit URL Eingabe

Abbildung 3 zeigt die Übersicht über die Importfunktion, die mit der Eingabe einer URL den Download einer JSON-Datei ermöglicht. Die JSON-Datei und die Config müssen nicht auf die gleiche Art hinzugefügt worden sein, damit der Datenimport funktioniert. Es ist möglich, die JSON-Datei aus einer URL herunterzuladen und die Config z.B. aus der Datenbank zu laden oder per Drag&Drop neu hinzuzufügen.

Konvertierung von JSON zu Tabellen

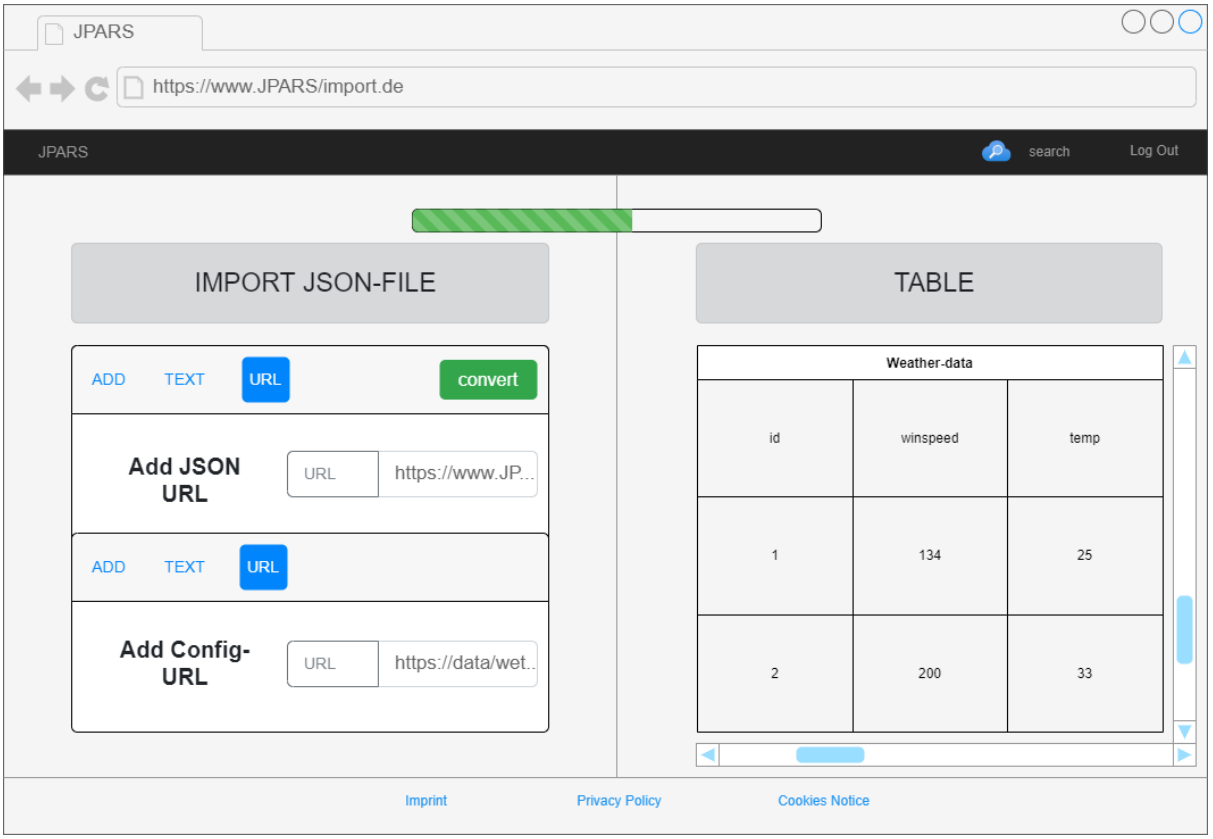


Abbildung 4: Die konvertierung von JSON in Tabellen

In Abbildung 4 wird die Verarbeitung von neuen JSON-Daten gezeigt. Die Verarbeitung sollte starten, nachdem der Nutzer die “convert” Taste drückt.

Auswählen einer Config-Datei

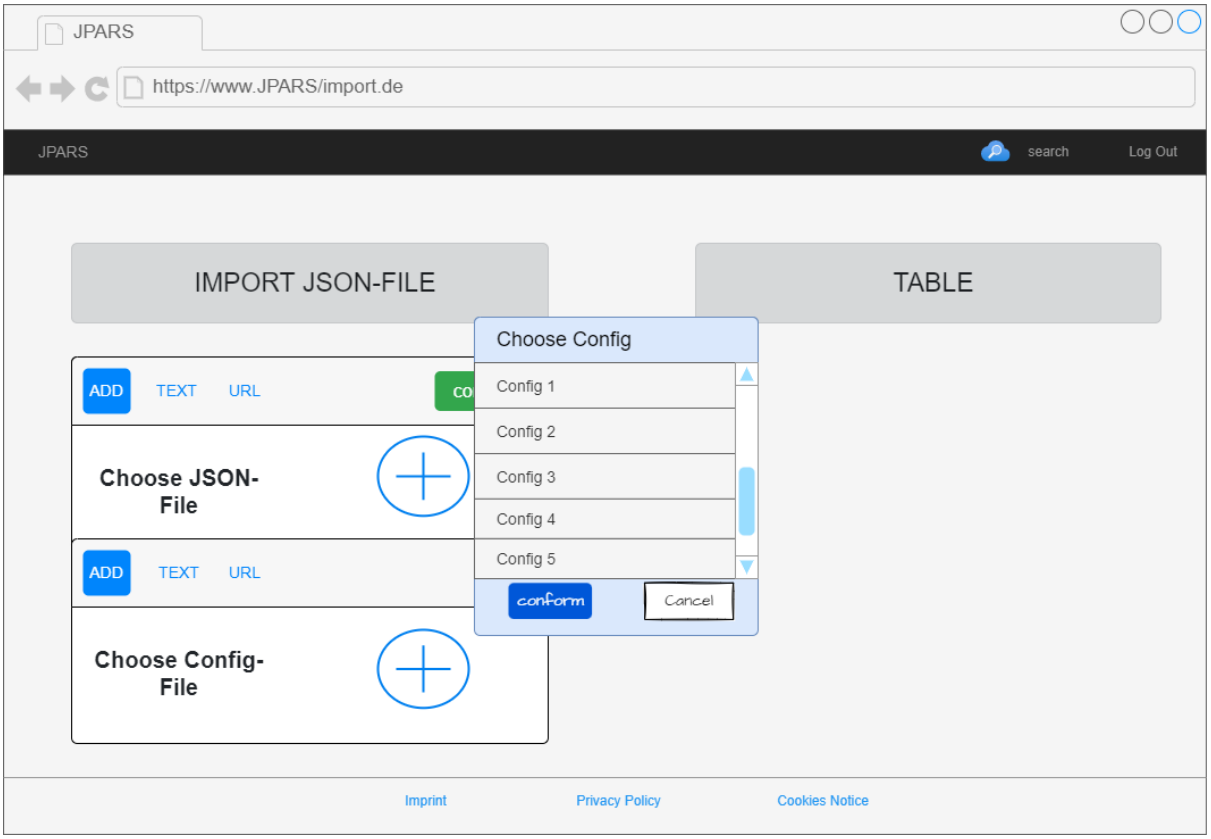


Abbildung 5: Beispiel zum Auswählen von bereits existierenden Config-Dateien

Abbildung 5 zeigt, wie der Nutzer aus den bereits existierenden Config-Dateien in der Datenbank eine Config-Datei auswählen kann. Diese Config-Datei wird dann für die importierte JSON-Datei verwendet.

Verwendung eines Filters

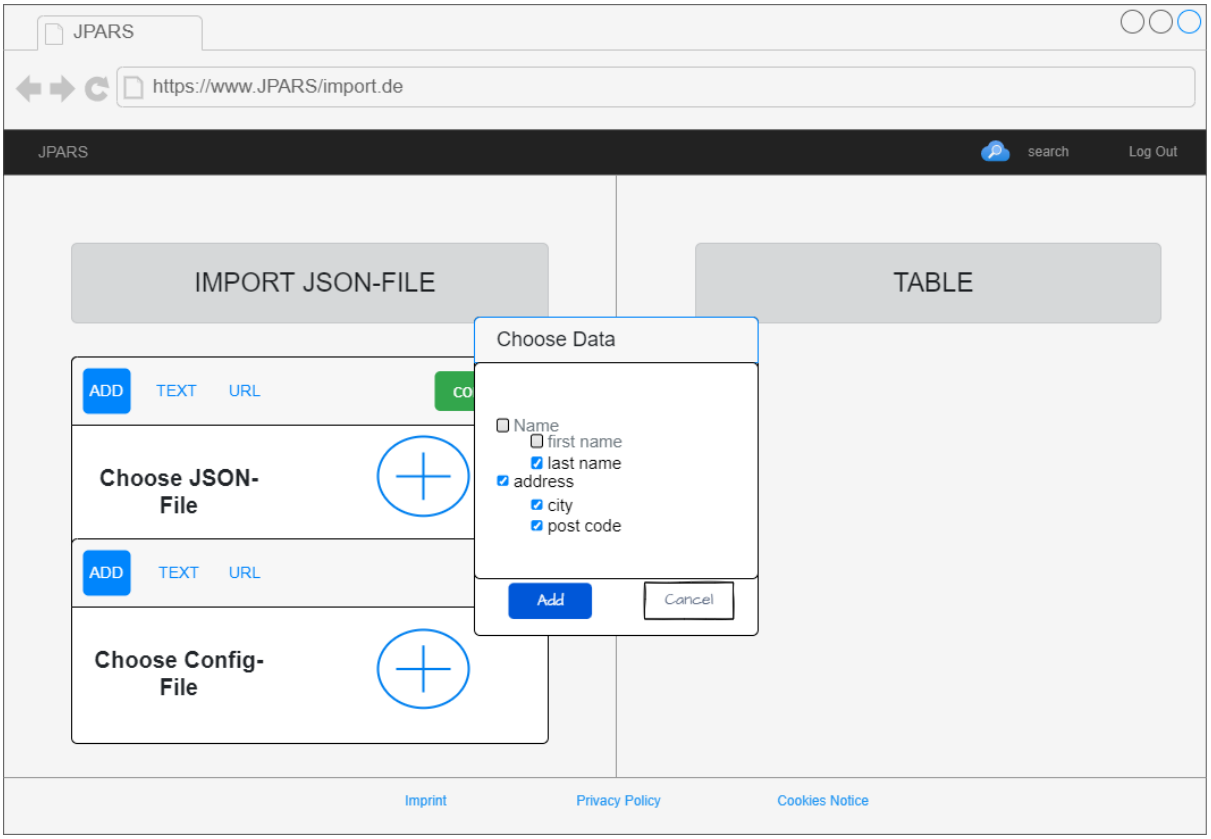


Abbildung 6: Beispiel zur Verwendung eines Filters

Wie in Abbildung 6 zu sehen ist, ermöglicht uns die Webapplikation nach der vollständigen Verarbeitung der JSON-Daten, die Daten aus den Dateien zu wählen, die später in einer Tabelle präsentiert werden.

Warnung bei fehlerhafter Datei

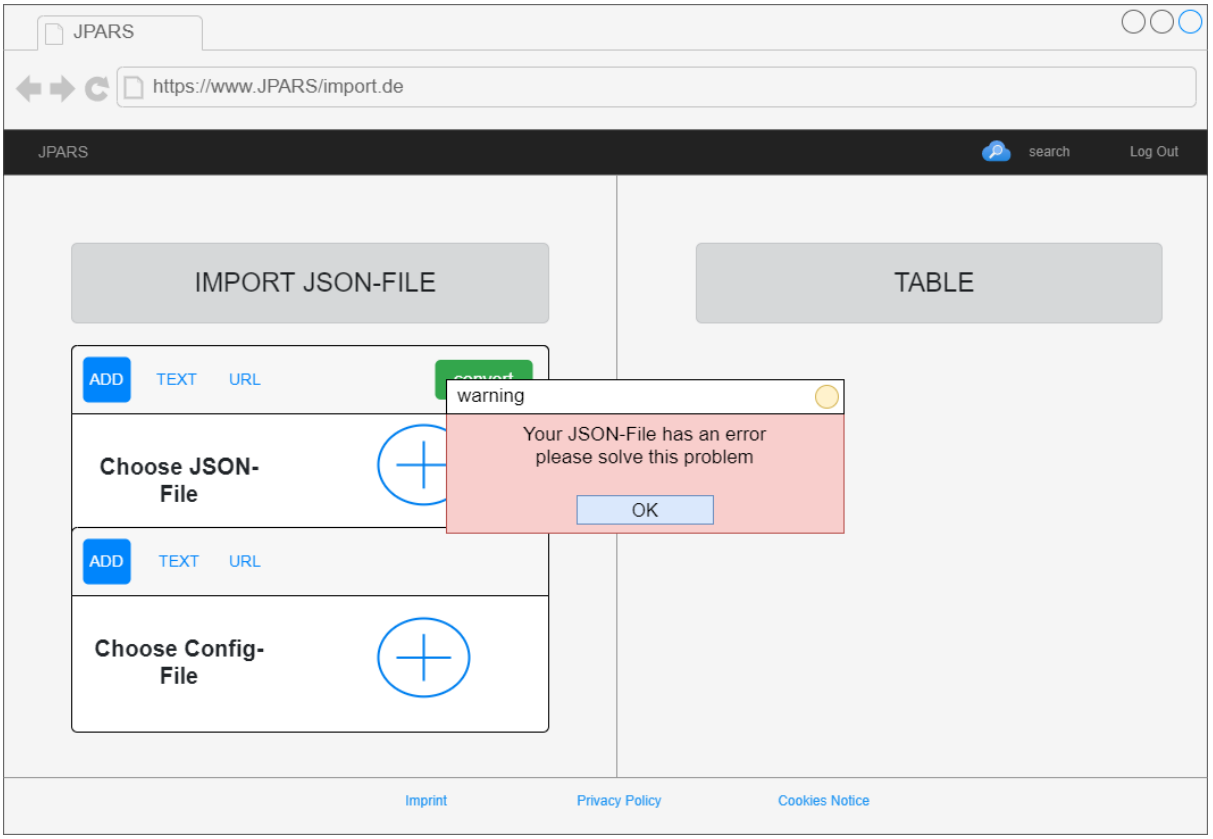


Abbildung 7: Die Übersicht im Fall einer fehlerhaften Datei

Im Fall, dass die Daten aus der JSON-Datei fehlerhaft sind, wird eine Fehlermeldung angezeigt, wie hier in Abbildung 7 dargestellt wurde.

Impressum

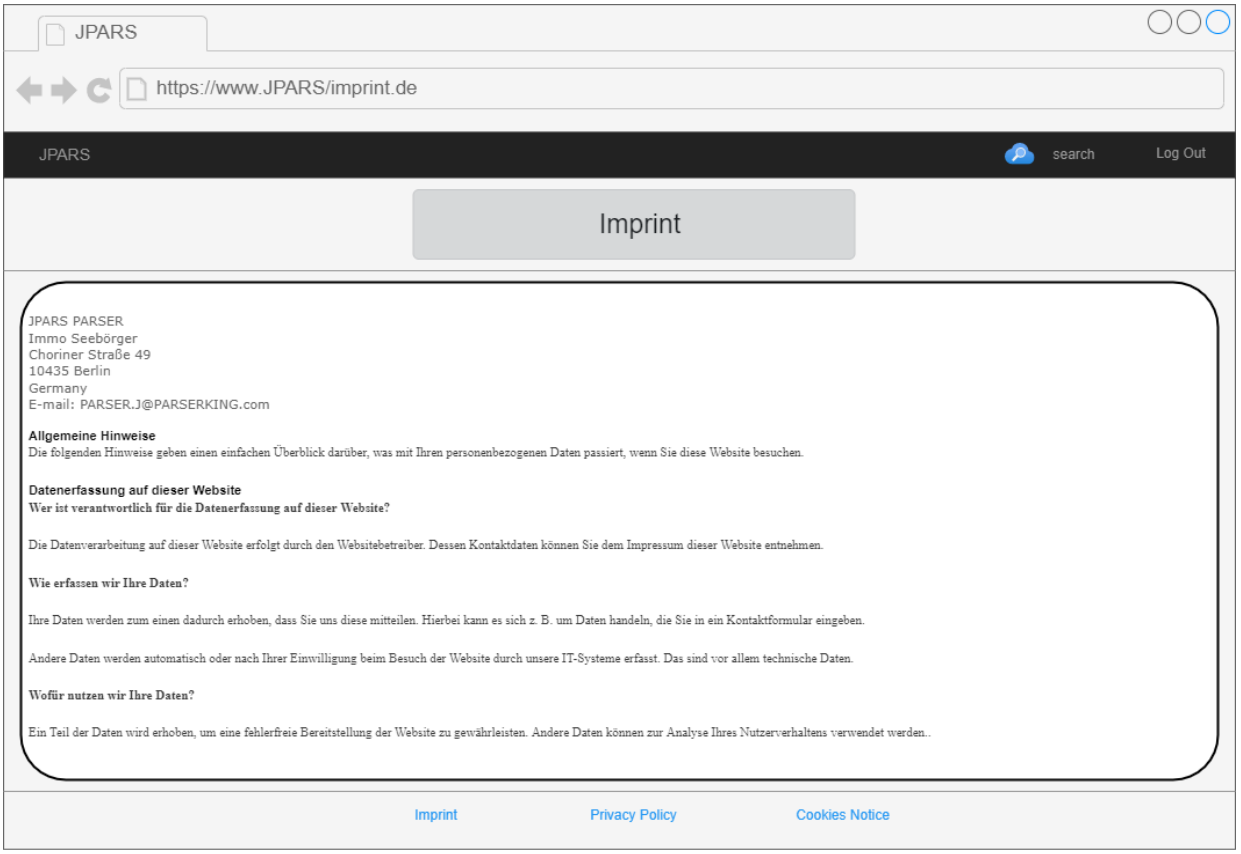
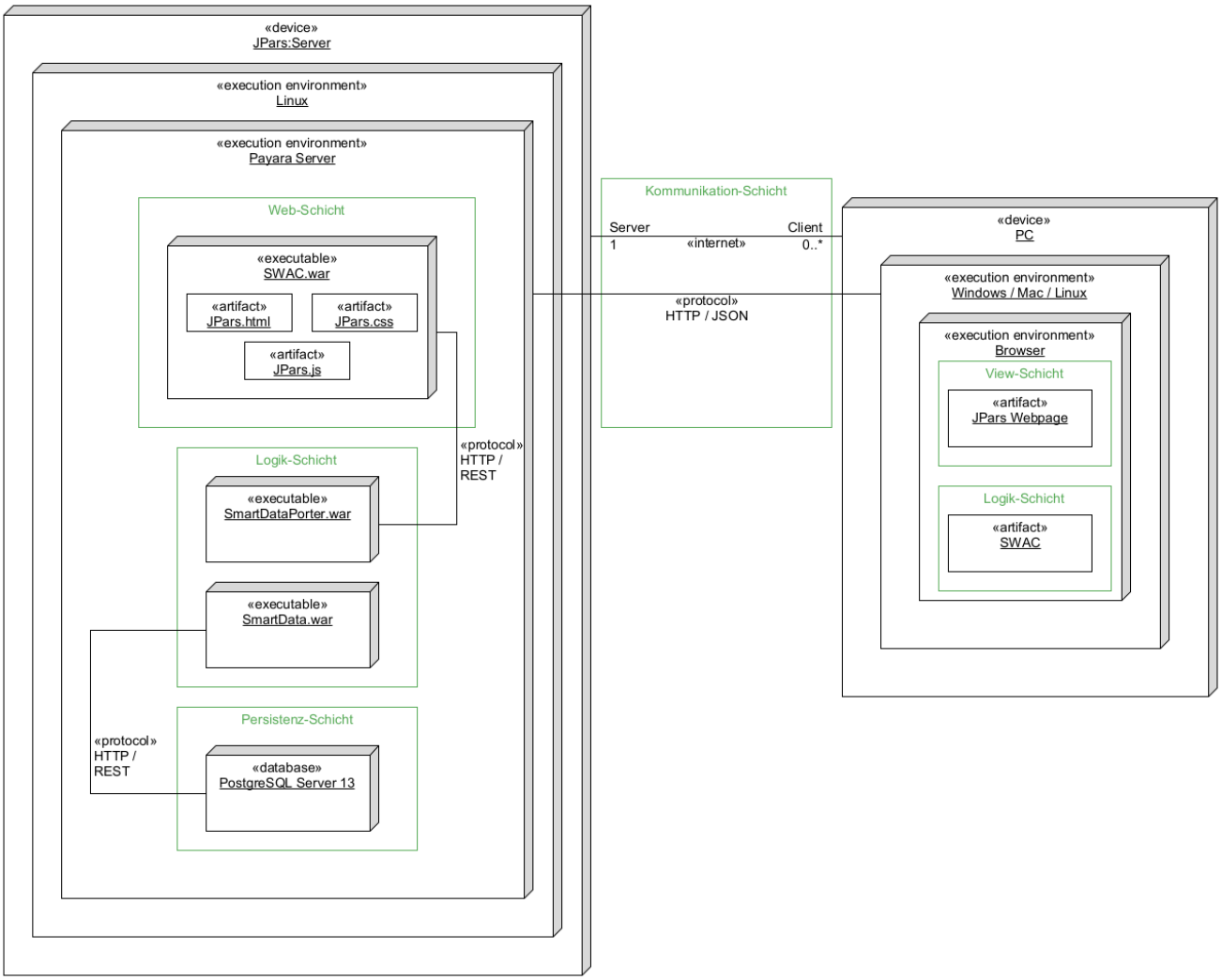


Abbildung 8: Impressum

Abbildung 8 zeigt das Impressum der Web Applikation.

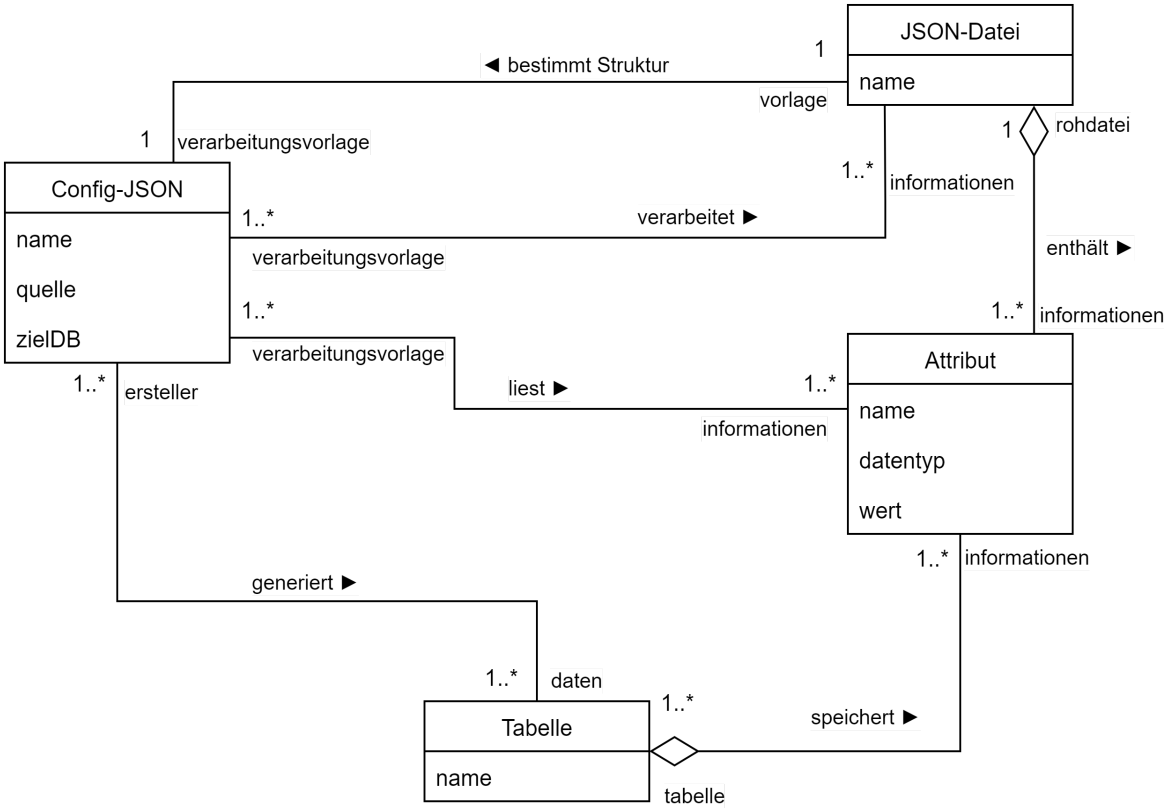
4 Technische Beschreibung

4.1 Deployment Diagramm



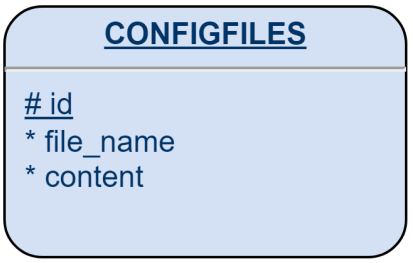
4.2 Datenmodell

Domain Model



Das Domain Model zeigt, welche Aufgaben eine JSON Datei und die Importkonfiguration haben und welche Daten diese mitbringen. Die JSON Datei kommt mit all ihren vollständigen Daten, wobei die Importkonfiguration eine Art Anleitung ist, wie die JSON Datei weiterverarbeitet in der Datenbank gespeichert werden soll.

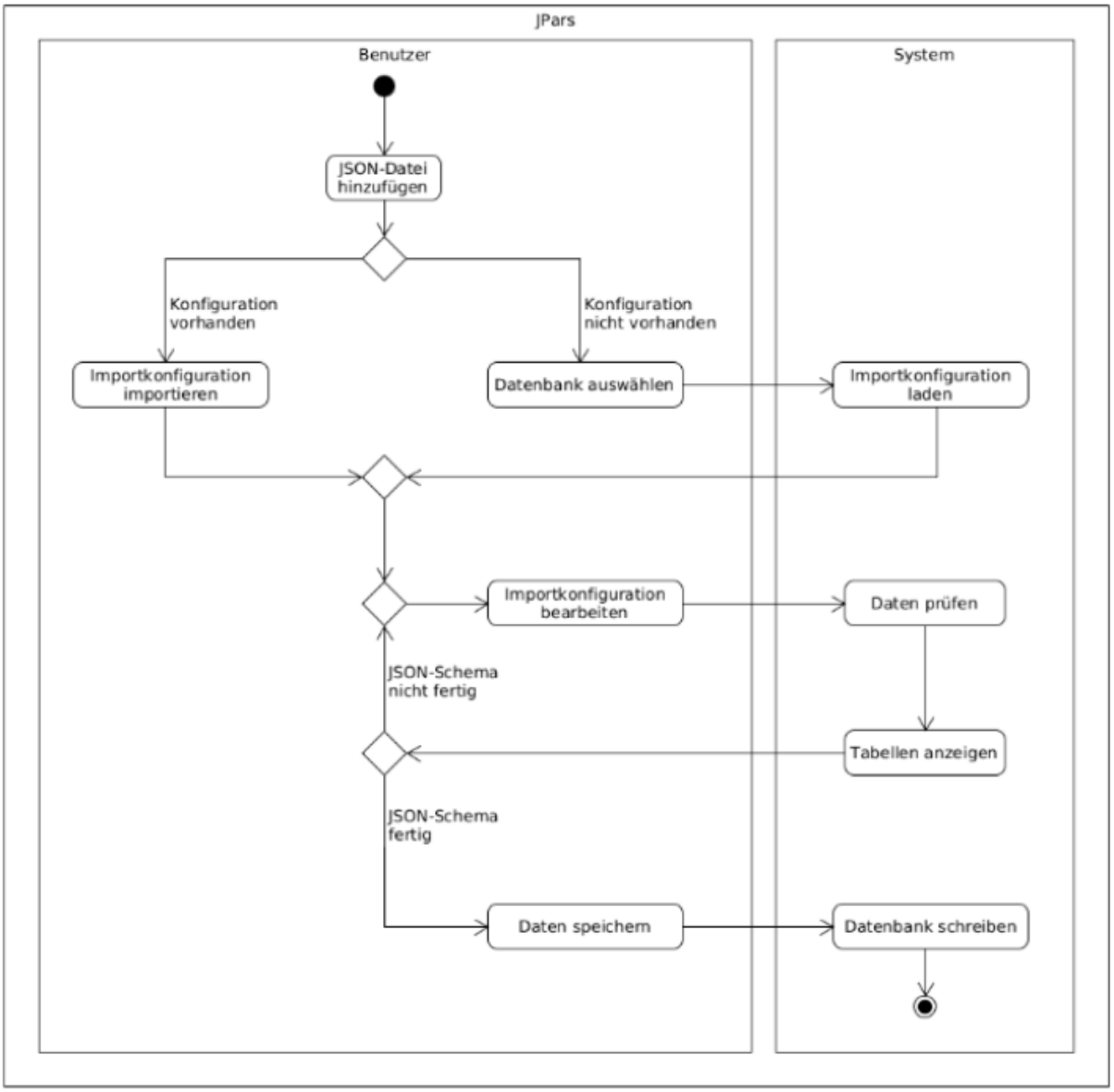
ER-Modell



Das ER-Modell zeigt, wie eine Importkonfiguration in der Datenbank gespeichert wird.

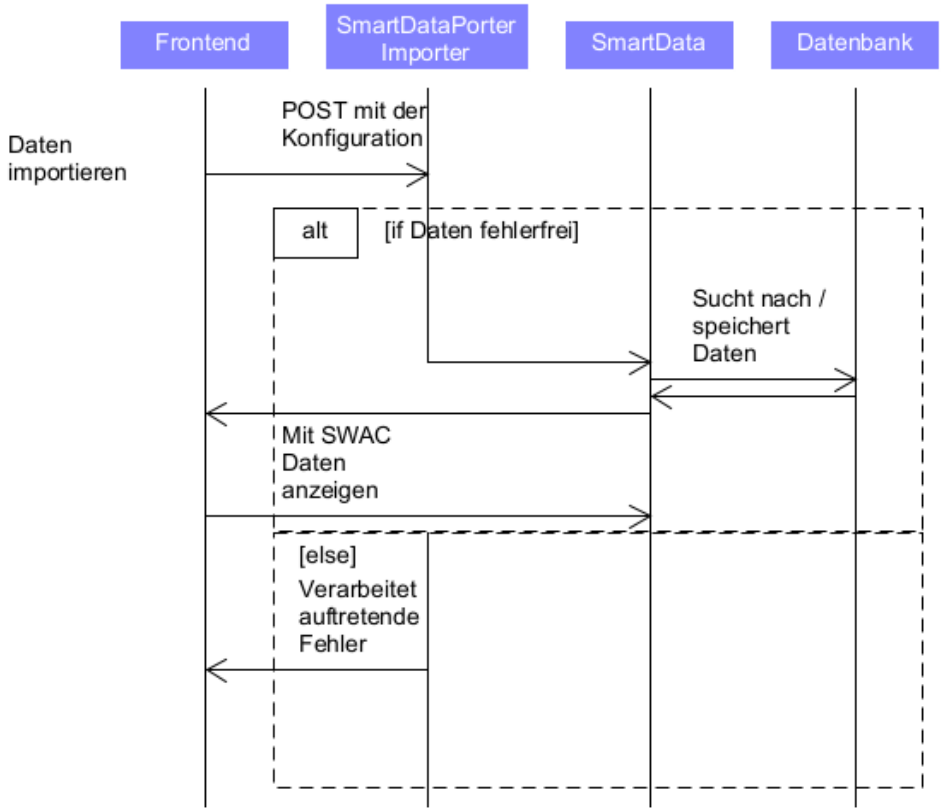
4.3 Abläufe

Flow-Diagramm

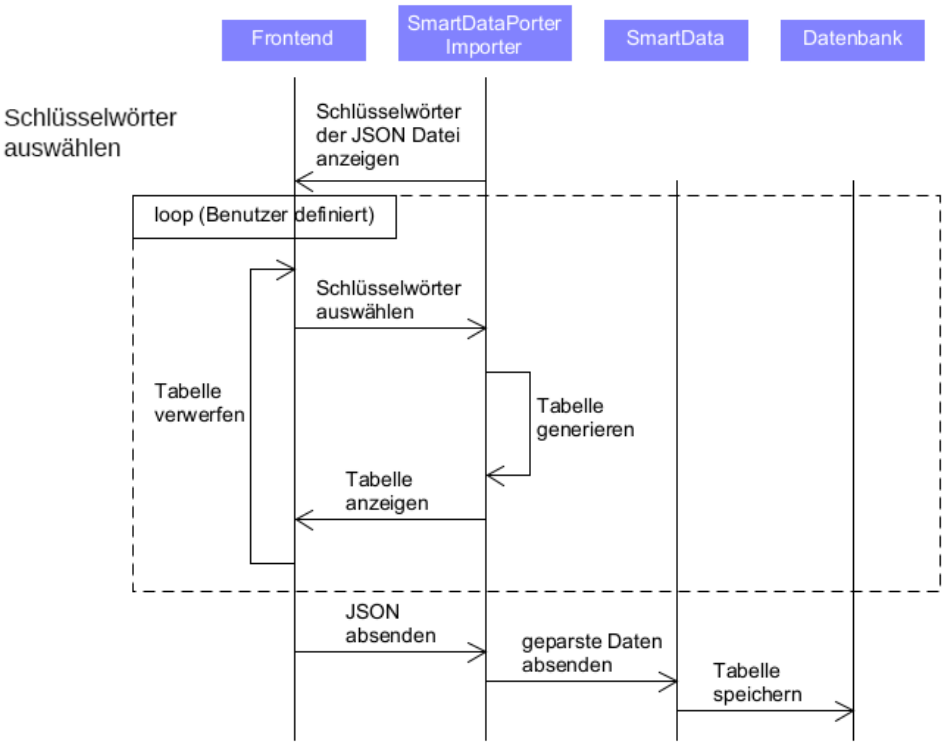


Vom Auswählen der JSON Datei bis zum Speichern der Daten in der Datenbank müssen einige Schritte abgearbeitet werden. Eine Importkonfiguration darf nicht fehlen, somit hat der Benutzer die Chance, eine Datei zu importieren oder eine ältere aus der Datenbank auszuwählen. Diese kann nach Wunsch vom Benutzer weiter angepasst werden.
Für den Benutzer ist es wichtig, die Importkonfiguration so anzupassen, dass nur die gewünschten Daten aus der JSON Datei weiterverarbeitet werden. Sobald die richtigen Schlüsselwörter ausgewählt wurden, kann der Importvorgang auch schon starten.

Sequenzdiagramm



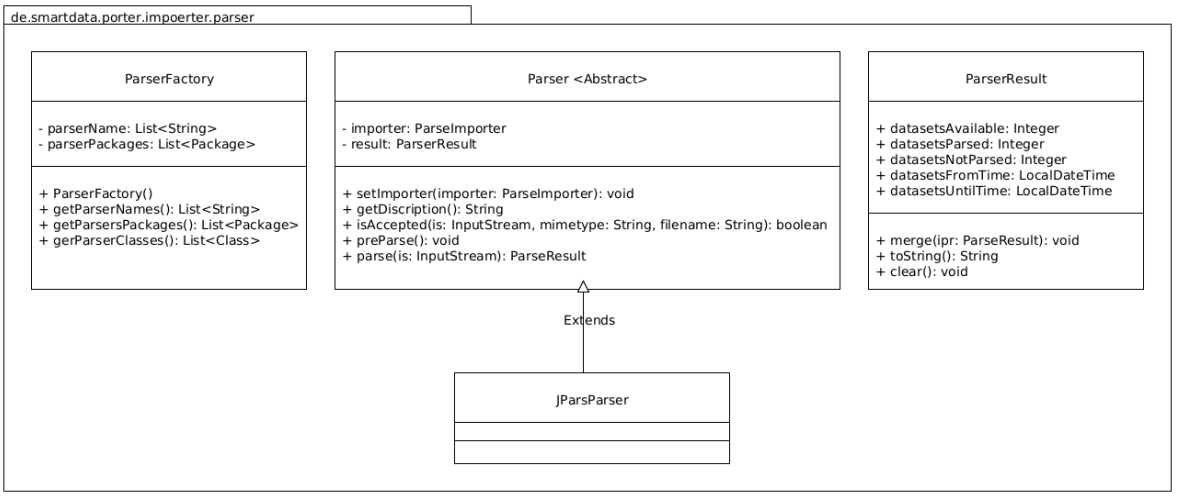
Der SmartDataPorter überprüft, ob die übergebene JSON Quelle Fehler enthält. Bei Fehlerfreien Quellen werden die Daten mithilfe des SmartDatas in die Datenbank gespeichert. Sobald aber eine Fehlerhafte JSON Quelle übergeben wurde, berichtet der Importer über diesen Fehler und bricht den Importvorgang ab.



Der Benutzer fügt eine JSON-Datei in das System ein. Dies kann aus unterschiedlichen Quellen erfolgen. Danach kann eine Importkonfigurationsdatei vom Benutzer hochgeladen werden. Ist keine Importkonfigurationsdatei vorhanden, kann der Benutzer eine Datenbank auswählen und die Datei wird vom System aus der Datenbank geladen. Bei einer neu erstellten Datenbank wird eine Importkonfiguration vom System erstellt. Die Importkonfigurationsdatei beinhaltet ein JSON-Schema, das aus der Struktur der JSON-Datei mit den Schlüsselwörtern und den Datentypen der Einträge besteht. Dieses kann vom Benutzer angepasst werden. Nach den Einstellungen wird vom System geprüft, ob die Daten in die Datenbank eingefügt werden können und eine Datenbank-Vorschau wird erstellt. Wenn die Einstellungen fertig konfiguriert sind, kann die Importkonfigurationsdatei exportiert werden. Danach wählt der Benutzer „Daten Speichern“ aus und das System schreibt die JSON-Daten in die ausgewählte Datenbank.

4.4 Entwurf

UML Klassendiagramm



5 Anleitung zur Nutzung von JPars

5.1 Die Startseite

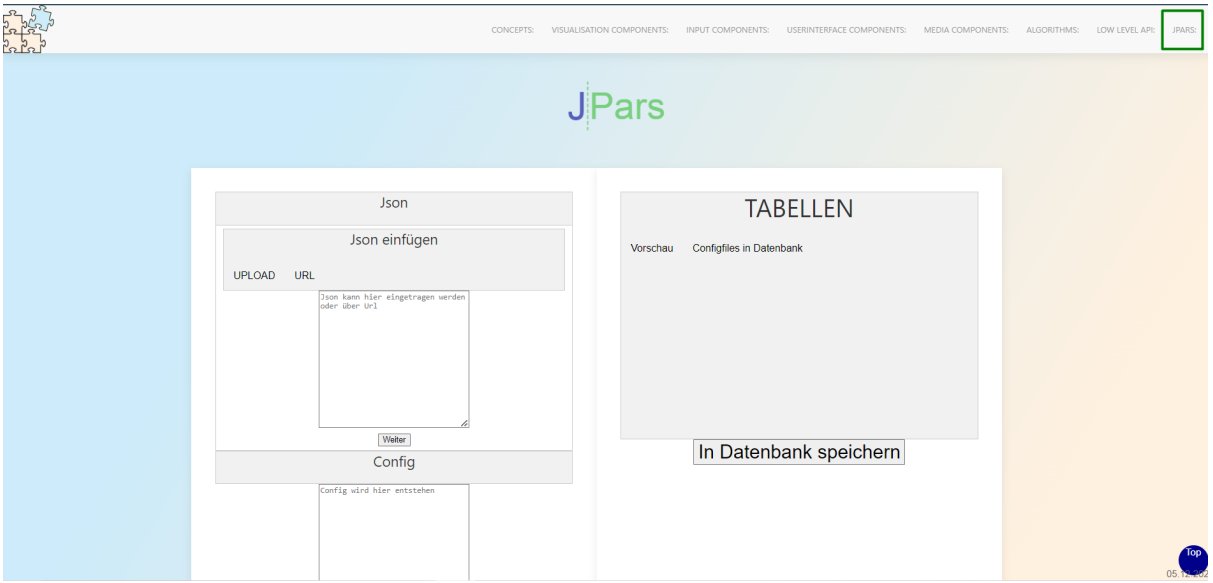


Abbildung 1: Die Startseite von JPars

Auf der linken Seite sind die jeweiligen Fenster für die JSON- und Config-Datei zu sehen. Auf der rechten Seite kann man sich die Ergebnisse anzeigen lassen oder ältere Konfigurationen aus der Datenbank laden.

5.2 JSON Einfügen

5.2.1 Upload von Json-Dateien

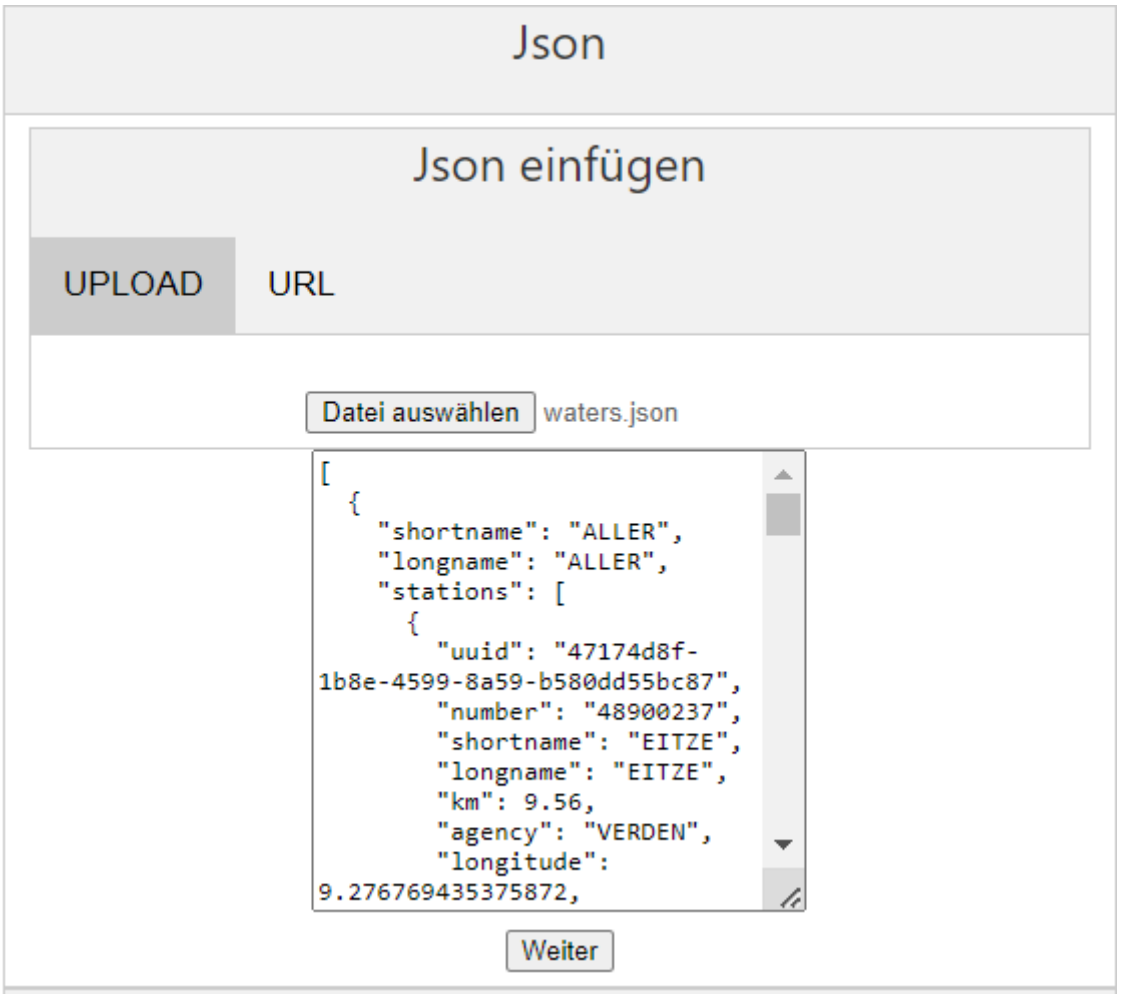


Abbildung 2: Das Einfügen von Json mit Upload

Um eine JSON Datei einzulesen, klickt man auf UPLOAD und zieht die gewünschte Datei in das kleine Fenster. Darunter wird dann die Datei ausgelesen angezeigt, diese kann man noch verändern. Um den Schritt abzuschließen, klickt man anschließend auf weiter.

5.2.2 Url von Json

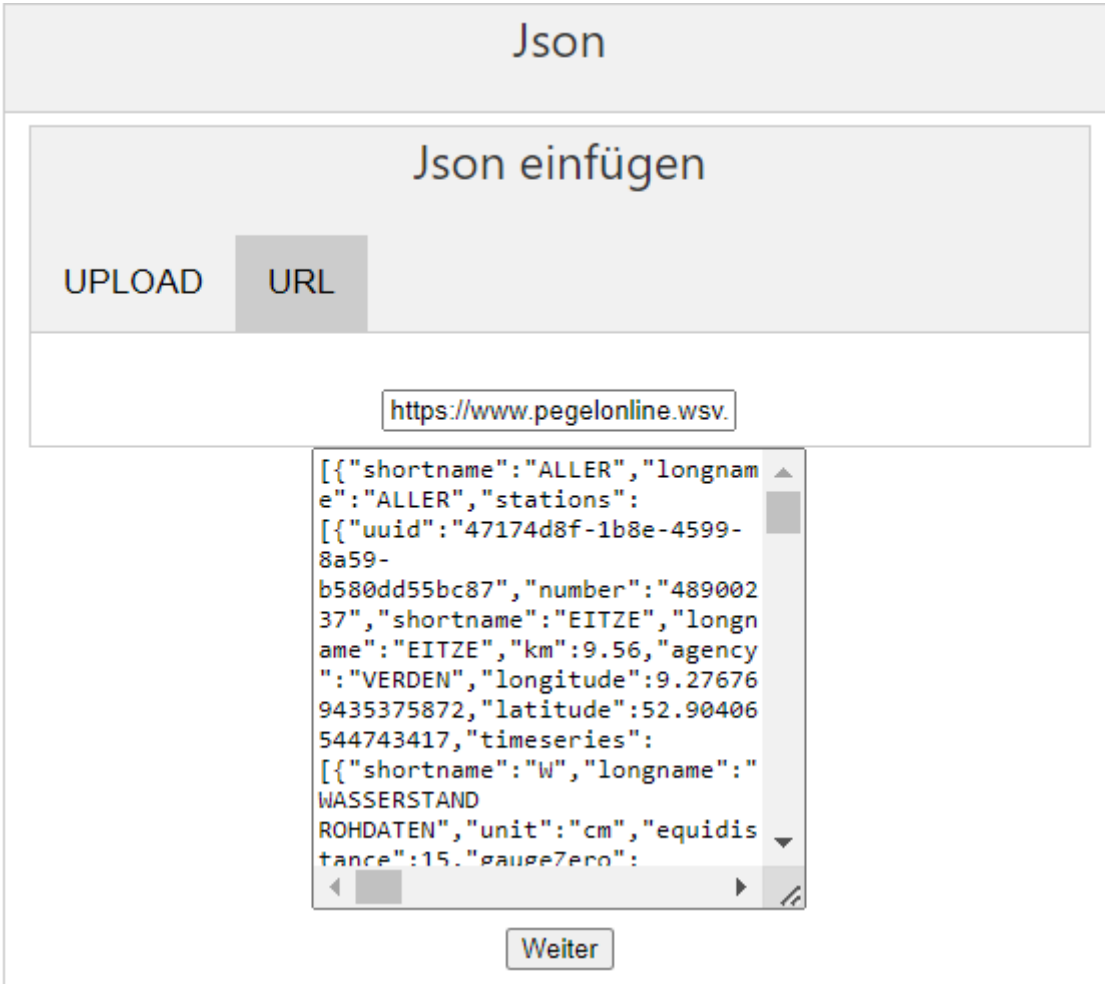


Abbildung 3: Das Einfügen von Json mit Url

Falls man die JSON per URL angeben möchte, klickt man zuerst auf URL und fügt in die kleine Spalte den gewünschten URL ein. Auch hier wird die JSON angezeigt und kann diese noch verändern. Als Text kann man die JSON auch angeben, dafür schreibt oder kopiert man in das Fenster in der Mitte einfach die JSON. Wenn man mit der JSON zufrieden ist, klickt man auf weiter.

5.2.3 Importkonfiguration bearbeiten

Abbildung 4: Erstellung von Konfiguration

Nun muss man mindestens eine Collection angeben, in der die Daten gespeichert werden sollen. Auch muss ein Name für die Importkonfiguration festgelegt werden, damit diese abgespeichert werden kann. Um beim Wiederverwenden einer Importkonfiguration nicht durcheinander zu kommen, ist es Sinnvoll, Namen zu verwenden, welche die Importkonfiguration beschreiben.

5.2.4 Importkonfiguration Ansicht

Abbildung 5: Ansicht von Konfiguration-Text

In dieser Ansicht, kann man oben die einzulesende JSON Datei sehen und unten die Importkonfiguration. Das obere Feld kann man bei diesem Schritt nicht mehr verändern. Aber das untere Feld, die Importkonfiguration, kann man noch beliebig anpassen, jedoch sollte man hier vorsichtig sein, denn fehlerhafte Änderungen können dazu führen, dass die Daten nicht mehr abgespeichert werden können.

5.2.3 Auswählen von Daten für Json-Tabellen

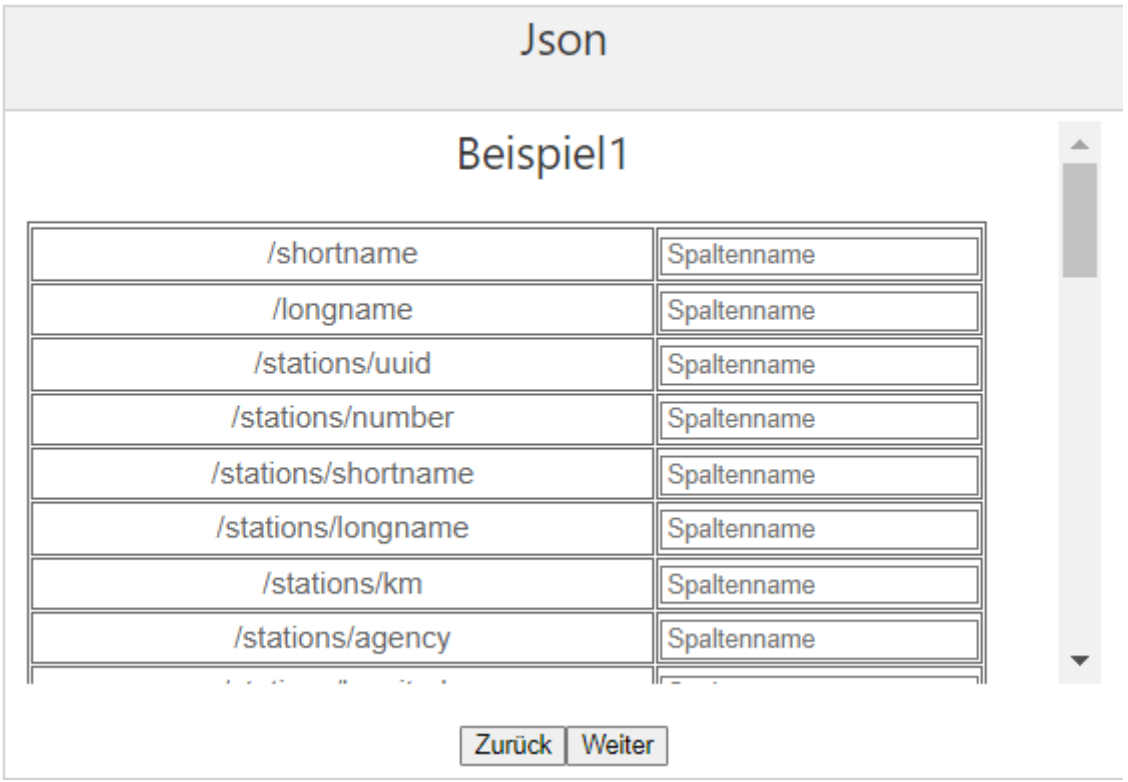


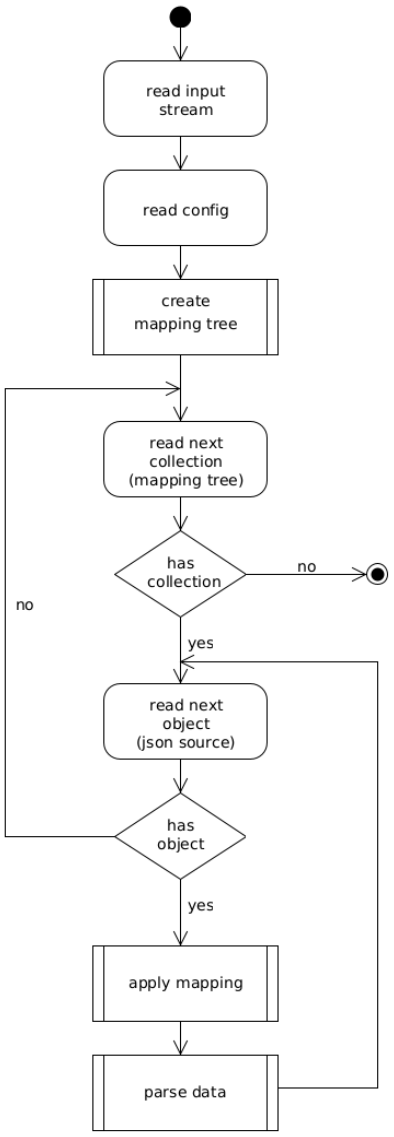
Abbildung 5: Erstellung von Importkonfiguration

Nun kann man die gewünschten Schlüsselwörter aus der JSON Datei auswählen und gleichzeitig angeben, in welche Spalten diese gespeichert werden sollen. Die nicht ausgefüllten Felder werden somit ignoriert und nicht in der Datenbank abgespeichert.

6 Parser Funktionsweise

Vereinfachte Übersicht der Funktionsweise des Parsers. Die JSON-Datei wird als Input Stream übergeben und der Parser erstellt daraus DataSets, die der Importer verarbeiten kann.

6.1 Parsing Prozess



Der Input-Stream wird gelesen und als JSON-Array für den weiteren Programmablauf vorbereitet. Im Folgenden als JSON-Quell-Datei bezeichnet.

Der Mapping-Bereich der Konfigurationsdatei wird ausgelesen. Dabei handelt es sich um ein JSON-Array mit dem Schlüsselwort „**json_mapping**“.

Der „**create mapping tree**“ Prozess wird gestartet. Für jede Collection wird ein eigener Mapping Tree erstellt. Als Ergebnis wird eine Sammlung zurückgegeben:

<Collection Name, Mapping Tree>

Diese Sammlung von Collections wird nacheinander in einer FOR-Schleife durchlaufen.

In jeder Collection-Schleife wird die gesamte JSON-Quell-Datei in einer FOR-Schleife durchlaufen.

Bei jedem ausgelesenen Objekt in dieser Schleife wird der „**apply mapping**“ Prozess angewandt. Dabei werden die benötigten Daten ausgelesen und in einen Data Tree geschrieben.

Mit dem fertigen Data Tree wird der „**parse data**“ Prozess gestartet. Es werden DataSets erstellt, die Werte hinzugefügt und an den Importer übergeben.

Sind alle Collections aus der Mapping-Sammlung durchlaufen, ist der „**parsing**“ Prozess abgeschlossen.

```java
    /**
     * Starts the parsing process.
     * Applies mapping and
     * parses datasets to the importer.
     *
     * @throws ImporterException
     */
    private void startParsing() throws ImporterException {

        // iterates through all collections in the mapping tree map
        for (Map.Entry<String, Tree> treeEntry : mappingTreeMap.entrySet()) {
            curCollection = treeEntry.getKey();
            Tree curMappingTree = treeEntry.getValue();

            // iterates through all objects from the json source data
            for (int jsonIndex = 0; jsonIndex < jsonDataArray.size(); jsonIndex++) {

                JsonObject curObject = jsonDataArray.getJsonObject(jsonIndex);

                // creates a new data tree for each object in the json data array
                dataTree = new Tree(null);

                applyMapping(curObject, curMappingTree, dataTree);
                readDataTree(dataTree);

            }
        }
    }
```

Die Methode startParsing() der Klasse JSONParser wird ausgeführt, wenn die Konfigurations- und JSON-Quell-Datei erfolgreich geladen und die mappingTreeMap erstellt ist.

Die Sammlung mappingTreeMap beinhaltet den Tabellennamen (Collection Name) und Mapping-Anweisungen für die Datenbank, in Form einer Baumstruktur (Mapping Tree).

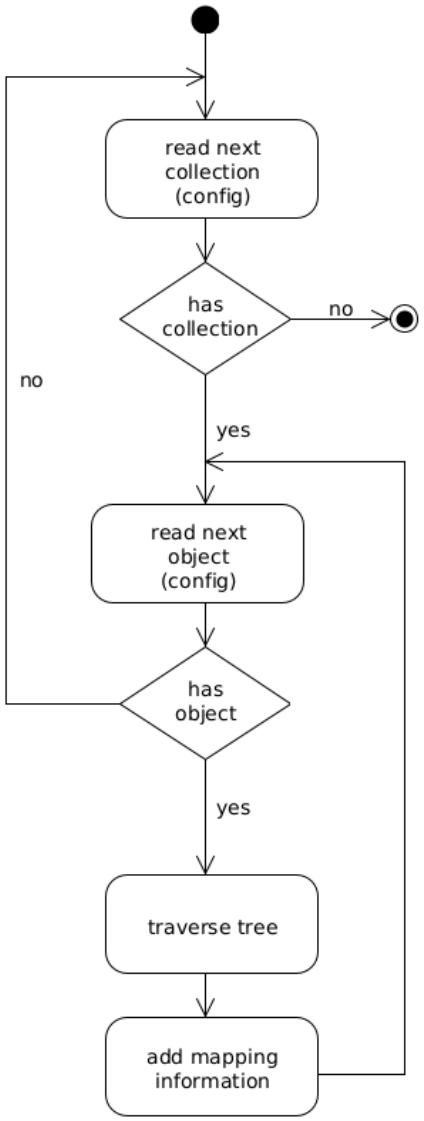
Die äußere Schleife iteriert durch alle Objekte der mappingTreeMap. Dabei werden der Name und der Mapping Tree für jede Collection geladen.

Die innere Schleife iteriert durch alle Objekte der JSON-Quell-Datei und erstellt für jedes dieser Objekte einen neuen Data Tree.

Der Mapping Tree wird mit der Methode applyMapping() auf das gegenwärtige Objekt der JSON-Quell-Datei angewendet. Diese werden zusammen mit dem Data Tree als Parameter übergeben.

Die Methode readDataTree() liest den Data Tree aus, erstellt DataSets und übergibt diese an den Importer.

6.2 Create Mapping Tree Prozess



Der „**create mapping tree**“ Prozess bekommt ein JSON-Array aus der Importkonfigurationsdatei übergeben, welcher Collection-Arrays enthält.

Ein Collection-Array hat einen Namen mit dem Schlüsselwort: „**collection**“ und ein Mapping-Daten-Array mit dem Schlüsselwort: „**mappingdata**“.

Die Collection-Arrays werden mit einer FOR-Schleife durchlaufen.

Jedes Objekt im Collection-Array hat einen Pfad mit dem Schlüsselwort: „**path**“, für die benötigten Daten aus der JSON-Quell-Datei, sowie die Zielspalten für die Datenbank mit dem Schlüsselwort: „**dbcolumn**“.

Diese werden nacheinander ausgelesen und in die Sammlung „**valuesMap**“ im Mapping Tree geschrieben: **<JSON-Schlüssel, Datenbankspalte>**

Dabei wird auf Basis der Pfade eine ähnliche Struktur wie in der JSON-Quell-Datei erstellt.

Wie bereits erwähnt, wird als Ergebnis eine Sammlung zurückgegeben: **<Collection Name, Mapping Tree>**.

Der Prozess ist abgeschlossen, wenn alle Collections des JSON-Arrays aus der Importkonfigurationsdatei durchlaufen sind.

```java
/**
 * reads mapping object values
 * adds child trees according to the path
 * adds <json key , column > as value to the mapping tree
 *
 * @param mappingObject: current mapping object
 * @param mappingTree: mapping tree for the current collection
 */
private void addObject(JsonObject mappingObject, Tree mappingTree) {
    String column = mappingObject.getString(OBJECT_COLUMN);
    String pathString = mappingObject.getString(OBJECT_PATH);

    // splitts the path string
    PathSplitter path = new PathSplitter(pathString);

    ArrayList<String> pathList = path.getPathStringList();
    String valueKey = path.getJsonKey();

    Tree child = mappingTree;

    // iterates the tree according to the path
    for (int index = 0; index < pathList.size(); index++) {
        child = child.addChild(pathList.get(index));
    }

    // adds < json key , database column > to the tree value
    child.addValue(valueKey, column);

}
```

Die Methode addObject() der Klasse MappingTreeBuilder wird auf jedes Objekt der Collection-Arrays angewendet. Dabei wird das gegenwärtige Mapping Objekt und Mapping Tree als Parameter übergeben.

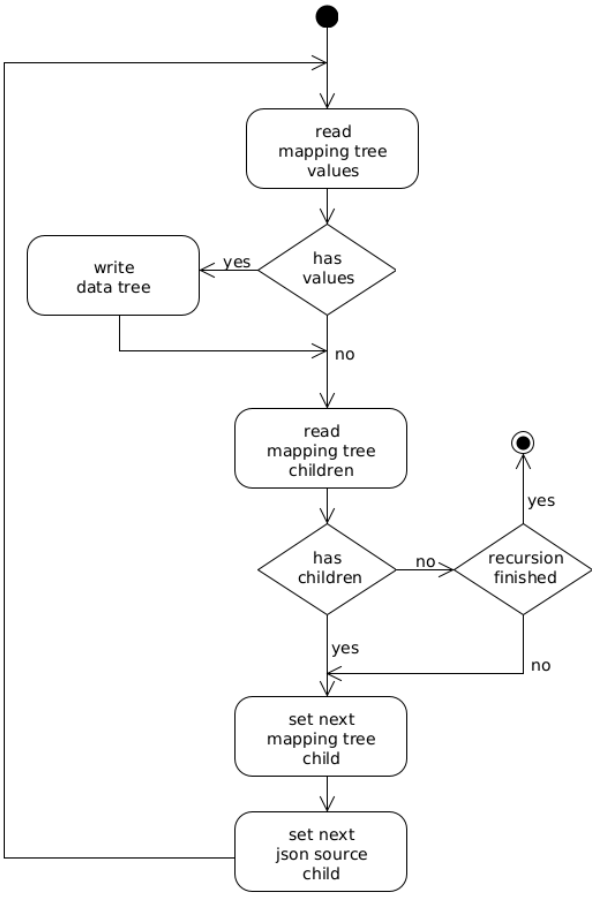
Im ersten Schritt werden der Pfad und die Zielspalte ausgelesen. Ein Pfad besteht aus einem String.Die Unterpfade werden mit dem Trennzeichen „/“ separiert.

Die Klasse PathSplitter trennt die Unterpfade in einzelne Strings und fügt diese in eine ArrayList (pathList) ein. Alle Zeichen nach dem letzten Trennzeichen werden als JSON-Schlüssel (valueKey) gespeichert.

Die pathList wird iteriert und die Methode addChild() der Klasse Tree aufgerufen. Dabei wird ein neues Kind mit dem Namen des Unterpfades in die Baumstruktur eingefügt, falls es noch nicht vorhanden ist.

Wenn der letzte Unterpfad erreicht ist, werden der JSON-Schlüssel und die Zielspalte in die „valuesMap“ des Mapping Tree geschrieben.

6.3 Apply Mapping Prozess



In diesem Prozess wird der Mapping Tree auf die JSON-Quell-Datei angewandt. Dabei werden die Ebenen beider Strukturen parallel, von oben nach unten, durchlaufen.

Im Mapping Tree wird geschaut, ob es in der aktuellen Ebene Werte gibt, die gespeichert werden müssen.

Mapping Tree „**valuesMap**“:
<JSON-Schlüssel, **Datenbankspalte>**

Die vorhandenen Daten werden aus der JSON-Quell-Datei ausgelesen und in den Data Tree geschrieben.

Data Tree „**valuesMap**“:
<Datenbankspalte, **JSON-Wert>**

Solange der Mapping Tree Kinder hat, wird die Methode wieder rekursiv aufgerufen. Die Objekte der tieferen Ebenen des Mapping Trees und der JSON-Quell-Datei werden als Parameter übergeben.

Sobald alle Ebenen des Mapping Trees durchlaufen sind, ist der Prozess abgeschlossen.

```java
    /**
     * recursive iteration through the json source data with the help from mapping tree
     *
     * @param curObject: current object in the json source data structure
     * @param mappingTree: current tree branch in the mapping tree
     * @param dataTree: current tree branch in the data tree
     */
    private void applyMapping(JsonObject curObject, Tree mappingTree, Tree dataTree) {
        // values in the mapping tree < json key , database column (String)>
        LinkedHashMap<String, Object> mappingValues = mappingTree.getValuesMap();
        LinkedHashMap<String, Tree> mappingChildrenMap = mappingTree.getChildrenMap();

        // add data if current mapping tree branch has values
        if (mappingValues != null) {
            writeDataTree(mappingValues, curObject, dataTree);
        }

        // if current tree branch has no children, reached one end of the mapping tree
        if (mappingChildrenMap == null) {
            return;
        }

        // iterates through all children in the current mapping tree branch
        for (Map.Entry<String, Tree> mappingChildrenEntry : mappingChildrenMap.entrySet()) {

            String jsonKey = mappingChildrenEntry.getKey();
            Tree childTree = mappingChildrenEntry.getValue();

            if (!curObject.containsKey(jsonKey)) {
                return;
            }

            Object nextObject = curObject.get(jsonKey);

            // current object is JsonArray
            if (nextObject instanceof JsonArray) {
                JsonArray subArray = (JsonArray) nextObject;
                // iterates through all objects in the array
                for (int arrayIndex = 0; arrayIndex < subArray.size(); arrayIndex++) {
                    // with multiple values adds a child to the data tree for correct data allocation
                    Tree tempTree = dataTree.addChild(String.valueOf(arrayIndex));
                    applyMapping(subArray.getJsonObject(arrayIndex), childTree, tempTree);
                }
                // current object is JsonObject
            } else {
                applyMapping((JsonObject) nextObject, childTree, dataTree);
            }

        }

    }
```

Bei der Methode applyMapping() der Klasse JSONParser werden die Mapping-Anweisungen auf die JSON-Quell-Datei angewendet und in den Data Tree geschrieben.

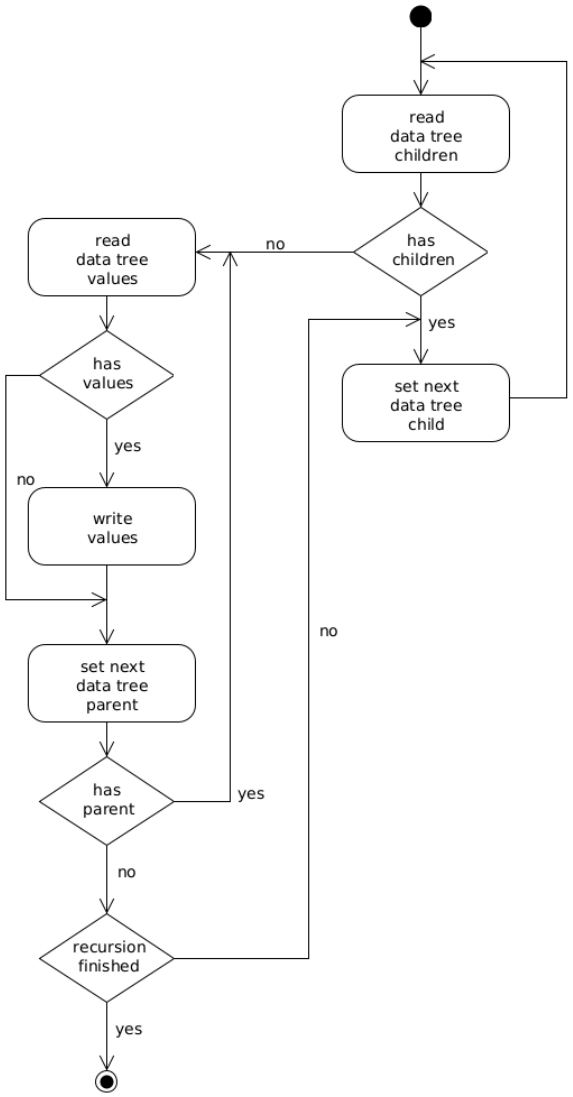
Die Parameter sind der Data Tree und das Objekt der JSON-Quell-Datei (curObject) und des Mapping Trees (mappingTree), der gegenwärtigen Ebene.

Mit den im mappingTree gespeicherten Werten (JSON-Schlüssel, Datenbankspalte) wird das curObject ausgelesen und die Werte in den Data Tree geschrieben.

Solange der mappingTree Kinderknoten hat, wird die Methode mit den Objekten der nächsten Ebene rekursiv aufgerufen.

Wenn curObject ein Array ist, werden dem Data Tree Kinderknoten hinzugefügt. Damit werden die Daten der Elternknoten allen Kindern zugewiesen.

6.4 Parse Data Prozess



Der Data Tree wird von oben nach unten rekursiv durchlaufen, bis zu den Elementen ohne Kinderknoten.

Diese Elemente werden zum Parsen übergeben.

Beim Parsen wird ein DataSet erstellt und der Data Tree wird von unten nach oben durchlaufen, bis ein Element keinen Elternknoten hat.

Dabei wird in jeder Ebene die „**valuesMap**“ ausgelesen und in das DataSet hinzugefügt: <**Datenbankspalte, JSON-Wert**>

Somit werden alle Daten der oberen Ebenen für jeden Wert eingefügt und gehen nicht verloren.

Beim obersten Elternknoten angekommen, wird der Collection Name hinzugefügt und anschließend das DataSet an den Importer übergeben.

Der Prozess ist abgeschlossen, wenn alle Knoten des Data Tree durchlaufen sind.

```java
    /**
     * iterates from bottom to top of the tree
     * creates datasets
     * parses data to the importer
     *
     * @param dataTreeBottom: last child from the data tree
     * @throws ImporterException
     */
    private void parseDataTree(Tree dataTreeBottom) throws ImporterException {

        JsonObjectBuilder jsonObject = Json.createObjectBuilder();
        Tree curTree = dataTreeBottom;
        boolean isDataEmpty = true;

        // goes from bottom to the top of the tree
        do {
            LinkedHashMap<String, Object> values = curTree.getValuesMap();
            if (values != null) {
                isDataEmpty = false;
                // iterates through all values in the current data tree branch
                for (Map.Entry<String, Object> databaseEntry : values.entrySet()) {
                    String column = databaseEntry.getKey();
                    JsonValue data = (JsonValue) databaseEntry.getValue();
                    jsonObject.add(column, data);
                }
            }

            curTree = curTree.getParent();
        } while (curTree != null);

        // check to not try to parse empty datasets
        if (!isDataEmpty) {
            // adds the target collection
            jsonObject.add("import.collection", curCollection);

            this.importer.addDataSet(jsonObject.build());
            result.datasetsParsed++;
        }

    }

}
```

Die parseDataTree() Methode der Klasse JSONParser bereitet einen DataSet vor und übergibt diesen an den Importer. Sie wird von der Methode readDataTree() aufgerufen, wenn diese einen Knoten ohne Kinder erreicht hat.

In Elternknoten sind immer gemeinsame Daten für alle Kinderknoten gespeichert, daher wird der als Parameter übergebene Kinderknoten in einer Schleife bis zum letzten Elternknoten durchlaufen. Dabei werden Daten aus jeder Ebene des Data Tree in ein DataSet eingefügt.

7 ConfigSaver Codeausschnitte

7.1 ConfigSaver.java

```java
/**
 * Saves a config into the database.
 *
 * This method looks for the keywords "file_name" and "content" and saves
 * the data into the table "configfiles".
 */
@Override
public ParserResult parse(InputStream is) throws ParserException {
    try {
        // Read config for "file_name"
        String fileName = this.importer.getConfig().getString("file_name");

        // Create JSON-Objects and read stream, stream contains "content"
        JsonReader reader = Json.createReader(is);
        String content = reader.readObject().toString();
        JsonObject jsonSet = createDataset(fileName, content);

        this.importer.addDataSet(jsonSet);
        result.datasetsAvailable = 1;
        result.datasetsParsed = 1;

        return result;
    } catch (Exception ex) {
        ParserException pe = new ParserException("Could not add values " +
                ex.getLocalizedMessage());
        pe.addSuppressed(ex);
        throw pe;
    }
}
```

Gezeigt ist die Methode parse() der Klasse ConfigSaver.java, welche die Klasse Parser erweitert. Die Aufgabe dieser Methode ist es, eine Configfile anzunehmen und diese in der Datenbank abzuspeichern.
Zuerst wird der String “file_name” aus der Importkonfiguration gelesen. Danach wird der Content der überlieferten Konfiguration aus dem InputStream gelesen. Diese beiden Strings werden mit der privaten Methode createDataset() in die Datenbank eingespeichert.

```
private JsonObject createDataset(String fileName, String content) {
    // Create dataset
    JsonObjectBuilder dataset = Json.createObjectBuilder();
    dataset.add("file_name", fileName);
    dataset.add("content", content);

    dataset.add("import.collection", "configfiles");

    return dataset.build();
}
```

Gezeigt ist die Methode createDataset(), welche den vorher gelesenen “file_name” und content annimmt und diese zu einem dataset baut, damit dieses in die Datenbank unter “configfiles” eingespeichert werden kann.

8 Frontend Codeausschnitte

8.1 Jpars.js

```
/**
 * Reads all keys that are in the JSON data file and filters out
 * multiple instances of one key, so keys are only unique.
 *
 * This function first reads the file from the textbox and then
 * starts to go recursively into the branches of the JSON file to find all keys.
 * If a key is an object or an array, the function recursively calls itself again
 * and goes deeper into the branch to look for keys.
 * The function only adds keys, that contain only premitive values.
 * No Arrays or Objects are added, except for Arrays, which only contain values.
 */
function Keyoutput() {
    uniqueKeys = [];
    collections = [];
    var data = document.getElementById("Jsontext").value;
    data = JSON.parse(data);
    jsontopars = data;
```

Gezeigt ist die Methode Keyoutput() der Datei Jpars.js, welche alle Schlüsselwörter aus der gegebenen JSON-Datei ausliest und darauf die doppelten Schlüsselwörter, unnötige Arrays und Objekte ausfiltert.
Zuerst werden hier Arrays angelegt, um die verschiedenen Daten abzuspeichern. Danach wird die gesamte JSON-Datei in die Variable data geparst und gespeichert.

```
// Find ALL keys, even multiple instances
const getNestedKeys = (data, resultKeys, path = "") => {

    if (!(data instanceof Array) && typeof data === 'object') {

        Object.keys(data).forEach(key => {
            if (!(data[key] instanceof Object)) {
                // Don't push Objects and Arrays (Arrays are Objects)
                resultKeys.push(path + key);
            } else if (data[key] instanceof Array) {
                var arr = data[key];
                if (!(arr[0] instanceof Object)) {
                    // Only push arrays that only contain values,
                    // no more nested attributes (Arrays or Objects)
                    resultKeys.push(path + key); // These arrays containing only
                                                 // values are pushed as a key
                }
            }
            var value = data[key];

            if (typeof value === 'object' && value != null) {
                getNestedKeys(value, resultKeys, path + key + "/");
            }

        });

    }
    else {
        Object.keys(data).forEach(key => {
            if (data instanceof Array) {
                getNestedKeys(data[key], resultKeys, path + "/");
            }
        });
    }

    return resultKeys;
};
var keys = getNestedKeys(data, []);
```

Danach wird die Methode getNestedKeys() definiert, welche nach der Definition direkt aufgerufen wird. Diese Methode wird rekursiv aufgerufen und geht immer tiefer in die JSON-Struktur und speichert dabei jedes einzelne Schlüsselwort ein, das dabei gelesen wird. Somit werden erst mal auch doppelte Schlüsselwörter ausgelesen. Ausnahme sind Arrays und Objekte, die nicht einzelne Werte enthalten, sondern weitere Variablen. Diese werden jetzt schon ausgefiltert.
Dabei geht die Methode so vor, dass erst geschaut wird, ob beim betrachteten Schlüsselwort ein Array oder Objekt vorliegt. Falls nicht, wird dieses Schlüsselwort in einem Array gespeichert. Falls es doch ein Array oder Objekt ist, wird die Methode rekursiv wieder aufgerufen und somit eine Ebene tiefer geschaut. Nur Schlüsselwörter mit primitiven Datentypen oder Arrays mit primitiven Datentypen werden letztendlich hinzugefügt.

```
    var keys = getNestedKeys(data, []);
    uniqueKeys = [...new Set(keys)]; // Filter out multiple instances of all keys,
                                     // only unique keys remain



    let pattern = /\/\//g;
    for (let i = 0; i < uniqueKeys.length; i++) {
        uniqueKeys[i] = "/" + uniqueKeys[i]; // Add front slashes
        uniqueKeys[i] = uniqueKeys[i].replace(pattern, "/"); // Remove double slashes
    }



    // User input, reads all collections the user puts in
    var collectionc = document.getElementById("collection").value;
    collectionc.split(",").forEach(function (item) {
        collections.push(item);
    });
    generatecheckbox(uniqueKeys, collections);


}
```

Im letzten Teil der Methode werden alle gespeicherten Schlüsselwörter verarbeitet.
Zuerst werden die doppelten Schlüsselwörter durch den kurzzeitigen Cast in ein Set ausgefiltert. Danach werden allen Schlüsselwörter ein Frontslash hinzugefügt und zusätzlich alle doppelt hintereinander auftretenden Slashes zu einem einfachen Slash gekürzt. Das wird gemacht, um eine einheitliche Struktur der Schlüsselwörter zu bekommen, welche auch in der weiteren Verarbeitung im Backend wichtig ist.
Zuletzt werden die Schlüsselwörter für die Ausgabe im Frontend in jede gelesene Collection eingespeichert, damit für jede Collection eigene Schlüsselwörter zur Speicherung ausgewählt werden können.

```
/**
 * generate a list of alle the Keys times the count of Collection
 */
function generatecheckbox(uniqueKeys, collections) {

    var stringfull = "<div class=\"scroller2\">";

    for (let j = 0; j < collections.length; j++) {
        var stringcol = "<h3>" + collections[j] + "</h3>" + "<table><tr>";

        stringfull = stringfull + stringcol;
        for (let i = 0; i < uniqueKeys.length; i++) {

            var stringkey = "<tr><td><label for=\" vehicle1 \">" + uniqueKeys[i]
            + "</label></td><td><input id=\"" + collections[j] + uniqueKeys[i]
            + "\" placeholder=\"Spaltenname\"><br></tr>";
            stringfull = stringfull + stringkey;
        }
        stringfull = stringfull + "</tr></table>";
    }
    stringfull = stringfull + "</div>";
    document.getElementById("demo").innerHTML = stringfull;
}
```

Hier werden die nun einzigartigen Schlüsselwörter in korrekter Struktur in die für die Anzeige im Frontend erstellten Listen eingebettet. Jede Collection wird durch eine eigene Liste angezeigt.

9 Fazit

9.1 Die Umsetzung

Die Gruppe empfand das Projekt als sehr gut durchführbar. Die Verwendung der REST-Schnittstelle „swagger“ sowie das Framework „SWAC“ nahm zwar etwas Einarbeitungszeit in Anspruch und stellte zunächst eine Herausforderung dar, da man aufkommende Probleme nicht googlen konnte, allerdings wurde der Gruppe mit schnellen und ausführlichen Antworten von Herrn Fehring gut geholfen. Auch die wöchentlichen Besprechungen konnten stetig Klarheit verschaffen.

9.2 Ausblick

Im Projekt wurde zunächst in der Implementierung nur die Anbindung an eine Datenbank (Lokal- und FH-Server) vorgesehen. Dies könnte sich in Zukunft ändern, sodass eine Anbindung an mehrere Datenbanken möglich ist. Zudem könnte in Zukunft eine Benutzerverwaltung eingefügt werden, um Einstellungen der Konfiguration-Dateien von JSON-Dateien selbst zu erstellen.