Hochschule Konstanz
Technik, Wirtschaft und Gestaltung

Hochschule Konstanz
Winter Semester 2022/2023

Dokumentation:
Team Projekt AIN PROSPER-DB: Systematische Personalsuche

Projektmitglieder:

Herr Konstantin Zabaznov,
Herr Muecahit Degirmenci,
Herr Stefan Grad,
Frau Selma Sahuric

Inhaltsverzeichnis

1. Vorstellung des Projekts

Bis voraussichtlich 2026 sollen 21 professorale Stellen an der HTWG besetzt werden.[1] Die größte Herausforderung für dieses Anliegen ist die geeignete Person in der großen Konkurrenz zu finden. An Fachhochschulen wird neben der didaktischen und wissenschaftlichen Qualifizierung, ausreichend Berufserfahrung verlangt – welche häufig fehlt.

Da für viele mit einer Professur der Weg der Wirtschaft interessanter ist als die akademische Laufbahn, ist es entscheidend für die HTWG, ein attraktiver Arbeitgeber zu werden.
Mit knapp 13.700 Professorinnen liegt der Frauenanteil bundesweit bei circa einem Drittel, der Frauenanteil an der HTWG-Professorenschaft liegt momentan bei 15%.

An dieser Problemstellung setzt das PROSPER-Projekt ein. Die HTWG Konstanz verfolgt drei Ziele zur Sicherung der Gewinnung professoralen Personals:

Durch Maßnahmen zur Erreichung der Berufungsvoraussetzungen wird das qualifizierte Fachkräfteangebot kurz- und langfristig erhöht.

Durch operative und strategische Rekrutierungsunterstützung wird das Bewerberaufkommen erweitert. Berufungskommissionen erhalten Unterstützung bei der Entscheidungsfindung und können damit Verfahren schneller abschließen.

Zur Erleichterung der Entscheidungsfindung von Bewerberinnen und Bewerbern und als Anreiz zum Verbleib in der Professur strebt die Hochschule gute Arbeitsbedingungen zur Vereinbarkeit von Beruf und Familie sowie zur Flexibilisierung der professoralen Aufgaben an. Damit werden erneute Vakanzen vermindert. Besondere Herausforderungen der Hochschule sind ein Schwerpunkt in MINT-Fächern, der Konkurrenz mit der Industrie um einen begrenzten Bewerberpool bewirkt, sowie die Nähe zum Hochlohnland Schweiz.

Im Projekt PROSPER verbessert die HTWG ihre Suchprozesse für die Berufung auf Professuren auf drei Ebenen. Erstens werden mehr Personen qualifiziert, um die Berufungsbedingungen zu erfüllen. Zweitens werden mehr berufungsfähige Personen angesprochen, die sich bislang nicht auf Professuren bewerben ("Aktive Rekrutierung"). Drittens werden Neuberufenen gute Einstiegsbedingungen und mittelfristige Entwicklungsmöglichkeiten aufgezeigt.

Die PROSPER-Aktivität SPRINT fokussiert sich auf die zweite Ebene, also die Ansprache auf sozialen Netzwerken (Xing, LinkedIn, Researchgate u.ä.) und filtert auf verbeamtungsfähige Altersgruppen. Das erhoffte Resultat ist eine Liste von Personen für die "Kaltakquise" auf absehbare Vakanzen der nächsten Jahre.

Eine in den meisten Gebieten harte (unabdingbare) Berufungsvoraussetzung ist die Promotion. Für eine systematische Ansprache sollen alle Promovierten in für die HTWG relevanten Fachgebieten (z.B. Informatik, Maschinenbau, aber nicht z.B. Medizin) ermittelt werden.

Die so ermittelten Identitäten sollen auf einer eigens kreierten Plattform identifiziert und in wenigen Schritten gefiltert werden, welche eine geeignete Besatzung wären.

Das Studentische Team des Bacherlorstudiengangs Angewandte Informatik (WS 2022/23) befasste sich als erste Gruppe mit der Recherche als auch der Umsetzung und dem Aufbau für die eigens HTWG geeignete Plattform.

[1] vlg. (HTWG-Konstanz, ohne Datum)

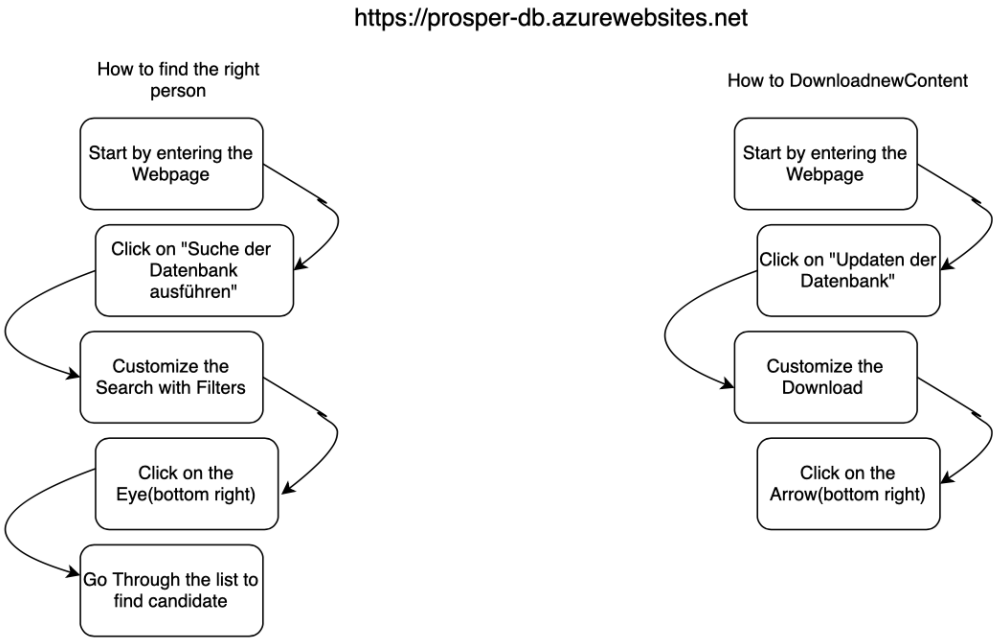
2. Anwenderdokumentation zur Plattform

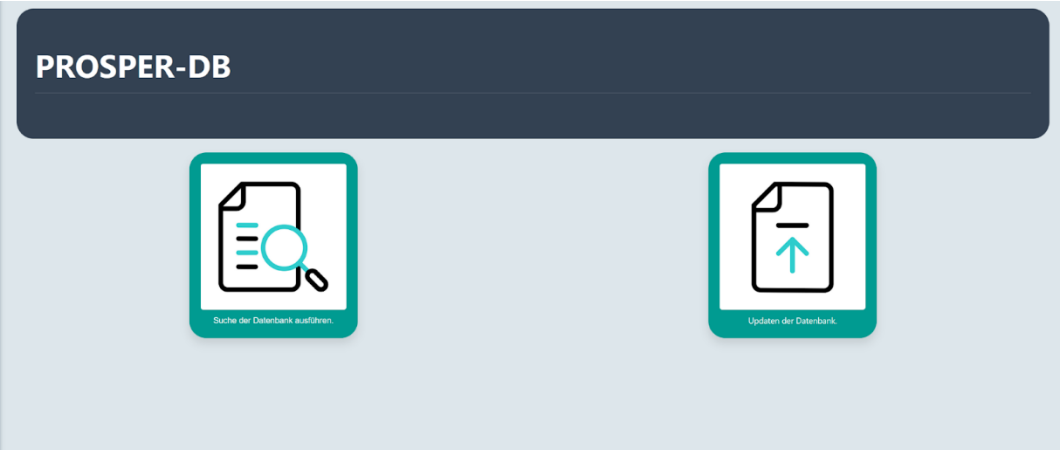
Das unten abgebildete Flussdiagramm zeigt die wichtigsten Schritte, die zur Nutzung des Produkts erforderlich sind. Es ist ein visuelles Tool, das dem Leser einen schnellen Überblick über den Ablauf des Prozesses gibt. Es kann auch helfen, die Zusammenhänge zwischen den einzelnen Schritten besser zu verstehen und die Abhängigkeiten zwischen den Schritten zu erkennen.

Die Schritte, die im Flussdiagramm dargestellt sind, sind grob zusammengefasst, um die Übersichtlichkeit zu erhöhen. Sie sollen dem Leser einen groben Eindruck darüber vermitteln, was in jedem Schritt passiert, ohne in zu viele Details zu gehen.

In der Anwenderdokumentation werden die Schritte dann detaillierter beschrieben und erklärt, um dem Leser ein tieferes Verständnis zu vermitteln.

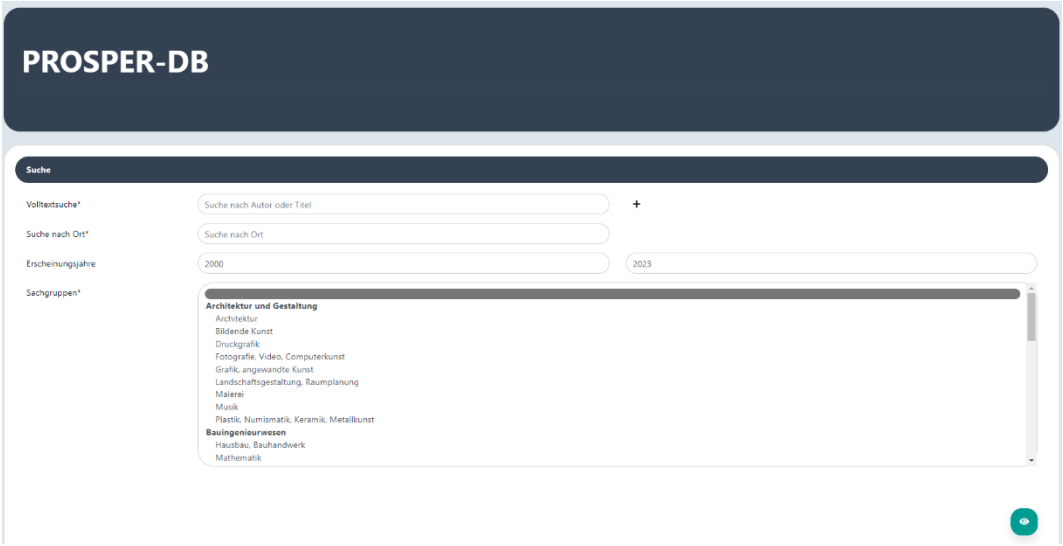
Öffnen Sie zunächst Ihren Webbrowser und geben Sie die folgende URL https://prosper-db.azurewebsites.net/ ein, auf der Sie dann auf der Prosper-DB Seite landen. Diese URL dient als Beispiel und kann je nach Bereitstellungsort variieren. Fragen sie für die genau URL den zuständigen Entwickler oder Administator.





Dort haben sie nun zwei Möglichkeiten.

Die erste wäre, dass sie in der Datenbank nach möglichen Professorin*innen suchen können. Dazu müssen Sie lediglich auf das Zeichen mit der Lupe (Suche der Datenbank ausführen) klicken. Haben Sie dies getan, werden Sie auch schon auf die nächste Seite weiter geleiten.



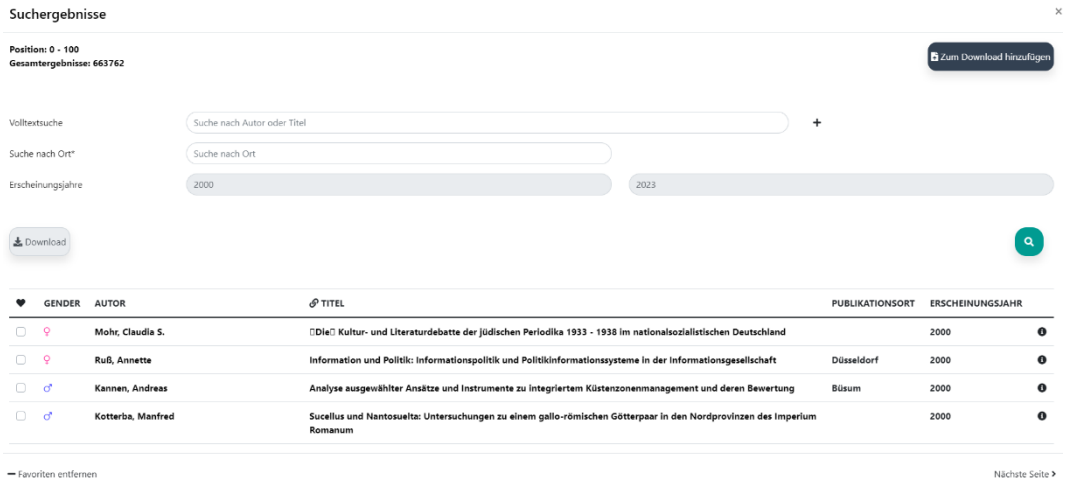
Dort sehen sie nun ein Fenster mit Filtermöglichkeiten. Diese wären:

- Die Volltextsuche ermöglicht es dem Benutzer, nach bestimmten Schlüsselwörtern in der Datenbank zu suchen. Mit dem Pluszeichen, das rechts daneben ist, kann der Benutzer weitere Schlüsselwörter hinzufügen, um die Suche weiter zu verfeinern.
- Die Suche nach dem Ort ermöglicht es den Publikationsort der jeweiligen Publikation auf eine bestimmte Hochschule einzuschränken.
- Das Erscheinungsjahr ist ein weiteres wertvolles Werkzeug zur Einschränkung der Suche. Hier kann der Benutzer das Jahr angeben, in dem die Suche durchgeführt werden soll. Dadurch werden nur Dissertationen angezeigt, die in diesem Jahr veröffentlicht wurden.
- Die Sachgruppen dienen dazu, die Suche weiter einzugrenzen, indem sie die Dissertation nur in bestimmten Fachgebieten durchführen. Der Benutzer kann aus einer Liste von Sachgruppen auswählen, die am besten zu seinem gewünschten Thema passen und die Suche wird nur in diesen Sachgruppen durchgeführt. Dadurch wird die Suche auf die relevanten Ergebnisse beschränkt und der Benutzer erhält eine gefilterte Ausgabe.

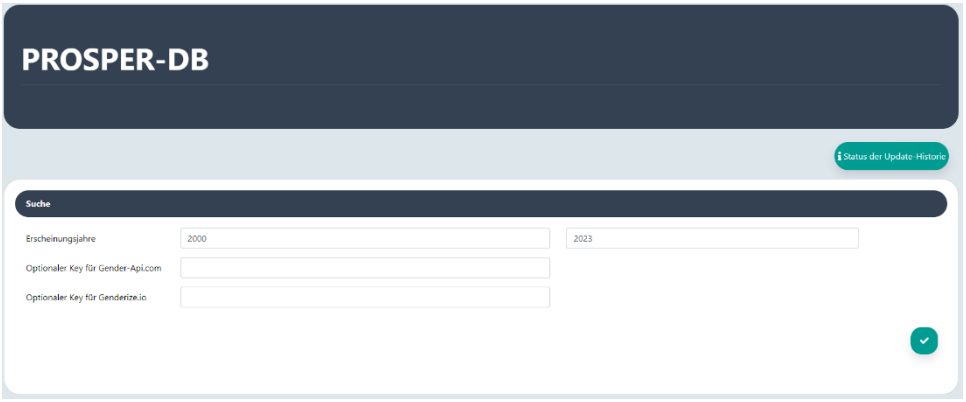
Insgesamt bieten die Volltextsuche, der Publikationsort, das Erscheinungsjahr und die Sachgruppen dem Benutzer eine Reihe von Werkzeugen, um die Suche nach Informationen in der Anwenderdokumentation zu vereinfachen und zu verfeinern und ermöglichen ihm so eine schnellere und präzisere Recherche.

Haben Sie ihre Filter genauer definiert und angeben, somit müssen sie auch nur noch auf das Auge unten rechts aufm Bild klicken, um die Suche zu starten.

Die Suchergebnisse sind jetzt zu sehen. Hier haben sie jetzt folgende Optionen:



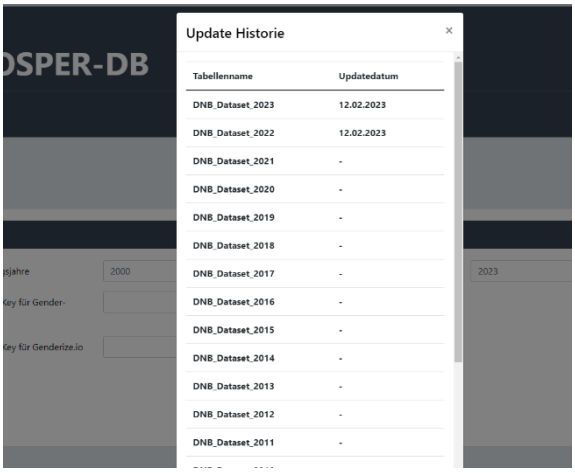
- Sie bietet dem Benutzer auch die Möglichkeit, die Suchergebnisse nach verschiedenen Kriterien zu sortieren. Der Benutzer kann nach Alphabet, Gender, Titel, Erscheinungsjahr und Favoriten sortieren. Mit einem Klick auf das gewünschte Kriterium werden die Ergebnisse aufsteigend sortiert und mit einem weiteren Klick absteigend.
- Klicken sie auf den Namen der Autor*innen, so landen sie auf dem LinkedIn Suchfenster. Dort kann man nun gezielter nach der Person suchen.
- Wollen sie jemandem favorisieren, so klicken sie lediglich auf das rechteckige Fenster unter dem Herzsymbol.
- Oben rechts können Sie die Daten der aktuellen Seite zum Download hinzufügen und Links durch Anklicken des Download-Buttons herunterladen. Es könnte ein Pop-Up-Fenster, das akzeptiert werden muss, erscheinen.
- Klicken Sie auf den Titel und landen Sie direkt auf der Seite, wo die Dissertation hochgeladen wurde.
- Bewegen Sie Ihren Mauszeiger auf das Ausrufezeichensymbol, dadurch erfahren Sie, in welchen Sachgruppen die Dissertation geschrieben wurde.
- Oben links wird die Anzahl der Suchergebnisse angezeigt und an welcher Position der Benutzer sich aktuell befindet. Auch hier hat der Benutzer die Möglichkeit, die Volltextsuche zu erweitern, um die Suchergebnisse zu verbessern.
- Sind sie fertig mit der Suche, so müssen sie nur auf das X oben rechts klicken.



Die zweite Möglichkeit wäre, die Datenbank zu aktualisieren. Um die Datenbank aktualisieren zu können, navigieren Sie zur "`Updaten der Datenbank`", falls verfügbar.

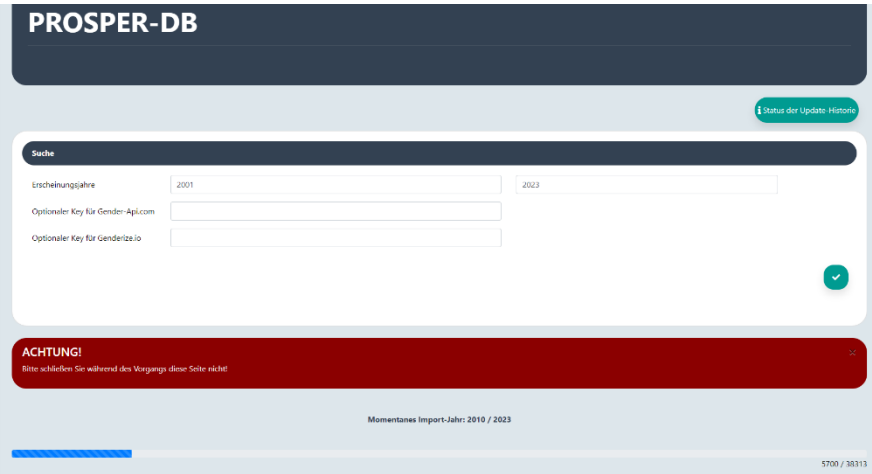
Hier haben sie jetzt folgende Optionen:

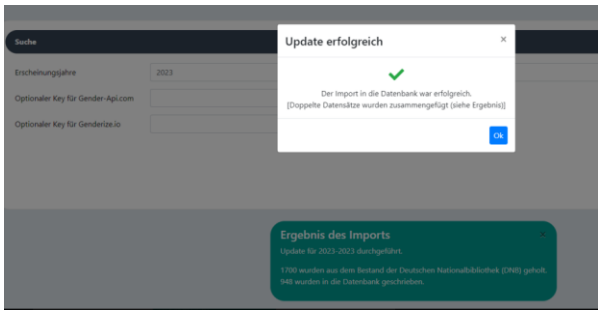
- Status der Update-Historie: Es erscheint eine weiteres Fenster, welches anzeigt, wann die Datenbanktabelle eines bestimmten Jahres das letzte Mal aktualisiert wurde.

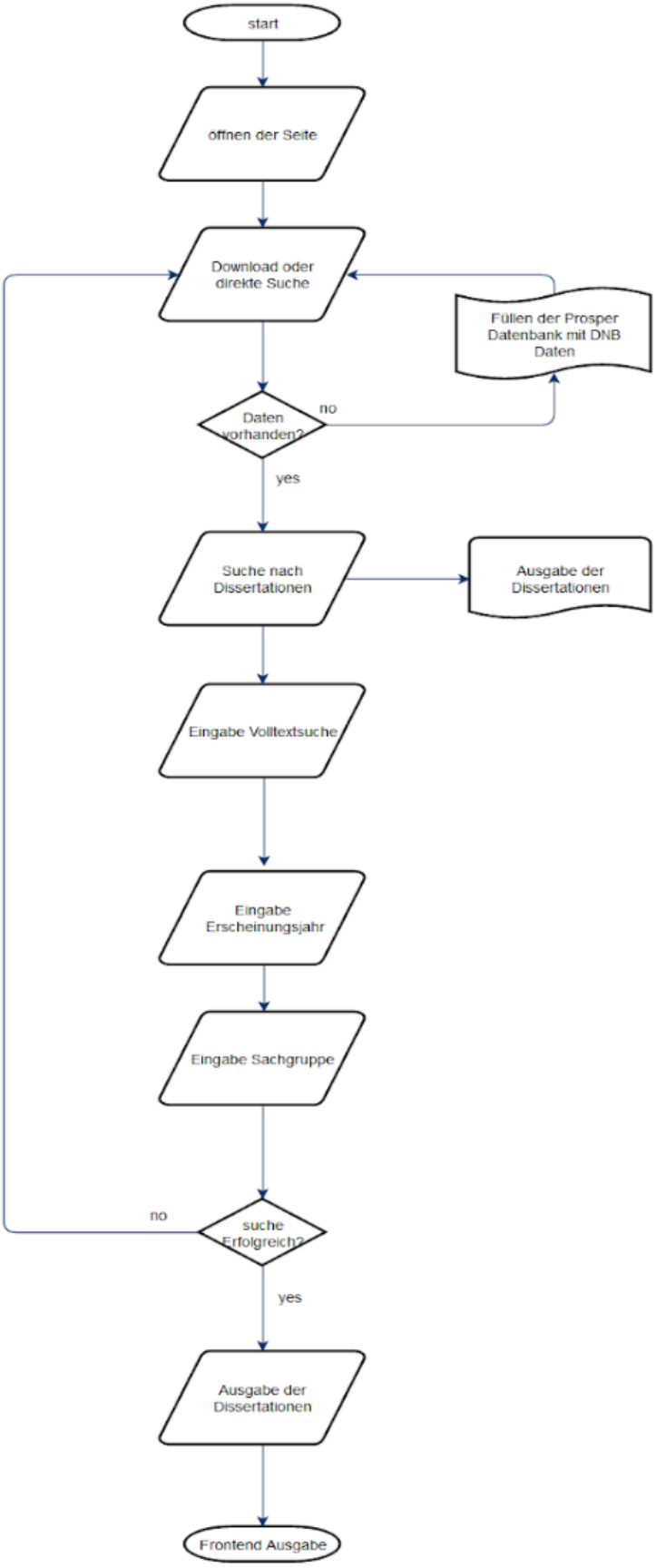


- Erscheinungsjahr: Das Erscheinungsjahr ist ein wichtiger Faktor bei der Suche nach Dissertationen. Es gibt an, in welchem Zeitraum die Daten aktualisiert werden sollen. Der Benutzer kann das Erscheinungsjahr eingeben, um die Suche auf bestimmte Jahre zu beschränken und so nur die relevanten Ergebnisse zu erhalten.
- Optional Key für Gender-API.com: Die Anwenderdokumentation bietet auch die Möglichkeit, optional einen Key für Gender-API.com und Genderize.io einzugeben. Diese Dienste ermöglichen es, das Gender von Autorinnen zu bestimmen. Wenn neue Namen aufgetaucht sind, die noch kein Gender haben, kann der Benutzer diesen Namen in den optionalen Key eingeben und das Gender wird automatisch bestimmt. Dies ermöglicht es dem Benutzer, die Suche nach Autorinnen gezielt nach ihrem Gender einzugrenzen.

Durch verschiedene Popups und Infoboxen wird der Nutzer durch den Aktualisierungsprozess geführt, um hierbei Fehler zu vermeiden:


PROSPER-DB
Status der Update-Historie
Suche
Erscheinungsjahre
2001
2023
Optionaler Key für Gender-Api.com
Optionaler Key für Genderize.io
ACHTUNG!
Bitte schließen Sie während des Vorgangs diese Seite nicht!
Momentanes Import-Jahr: 2010 / 2023
5700 / 38313


Suche
Erscheinungsjahre
2023
Optionaler Key für Gender-Api.com
Optionaler Key für Genderize.io
Update erfolgreich
Der Import in die Datenbank war erfolgreich.
[Doppelte Datensätze wurden zusammengefügt (siehe Ergebnis)]
Ok
Ergebnis des Imports
Update für 2023-2023 durchgeführt.
1700 wurden aus dem Bestand der Deutschen Nationalbibliothek (DNB) geholt.
948 wurden in die Datenbank geschrieben.


start
öffnen der Seite
Download oder direkte Suche
Füllen der Prosper Datenbank mit DNB Daten
Daten vorhanden?
no
yes
Suche nach Dissertationen
Ausgabe der Dissertationen
Eingabe Volltextsuche
Eingabe Erscheinungsjahr
Eingabe Sachgruppe
suche Erfolgreich?
no
yes
Ausgabe der Dissertationen
Frontend Ausgabe

3. Entwicklerdokumentation

Diese Entwicklerdokumentation richtet sich vor allem an Entwickler, sowie an Personen eines fachspezifischen Publikums, welche die Aufgaben der Weiterentwicklung und Wartung der Web-Anwendung „PROSPER-DB“, inklusive aller zugehörigen Komponenten haben.

Verwendete Technologien

.NET-Framework & C#

Die zugrundeliegende Programmiersprache, welche für das Front-End, sowie das Back-End verwendet wird ist C#. C# ist eine der beliebtesten Programmiersprachen, mit einem guten Support, einer hohen Communitydichte und unterliegt dem objektorientierten Konzept und wird kontinuierlich weiterentwickelt.

Das .NET-Framework wurde unter anderem an C# angepasst und wird als zugrundeliegendes Framework verwendet. Die Anwendung läuft momentan an .NET-Framwork Version 6.0, welche einen Langzeitsupport hat.

Blazor Server

Blazor ist ein WebFramework, welches für die Erstellung von Komponenten für Webseiten (Razor-Komponenten) zuständig ist. Blazor Server wird komplett auf dem Server ausgeführt, arbeitet also mit Thin-Clients. Blazor wird verwendet um das Front-End aufzubauen, sowie die Datenbankabfrage mittels API bereitzustellen.

Microsoft SQL Server

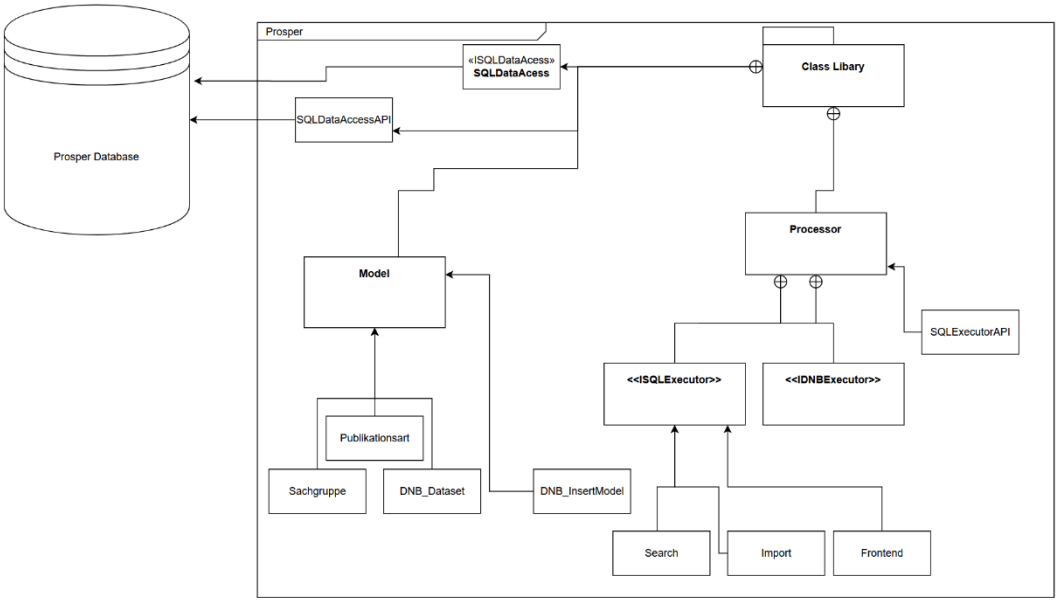
Microsoft SQL Server ist ein relationales Datenbankmanagementsystem, welches für die persistenten Speicherung von Daten zuständig ist.

Microsoft Azure

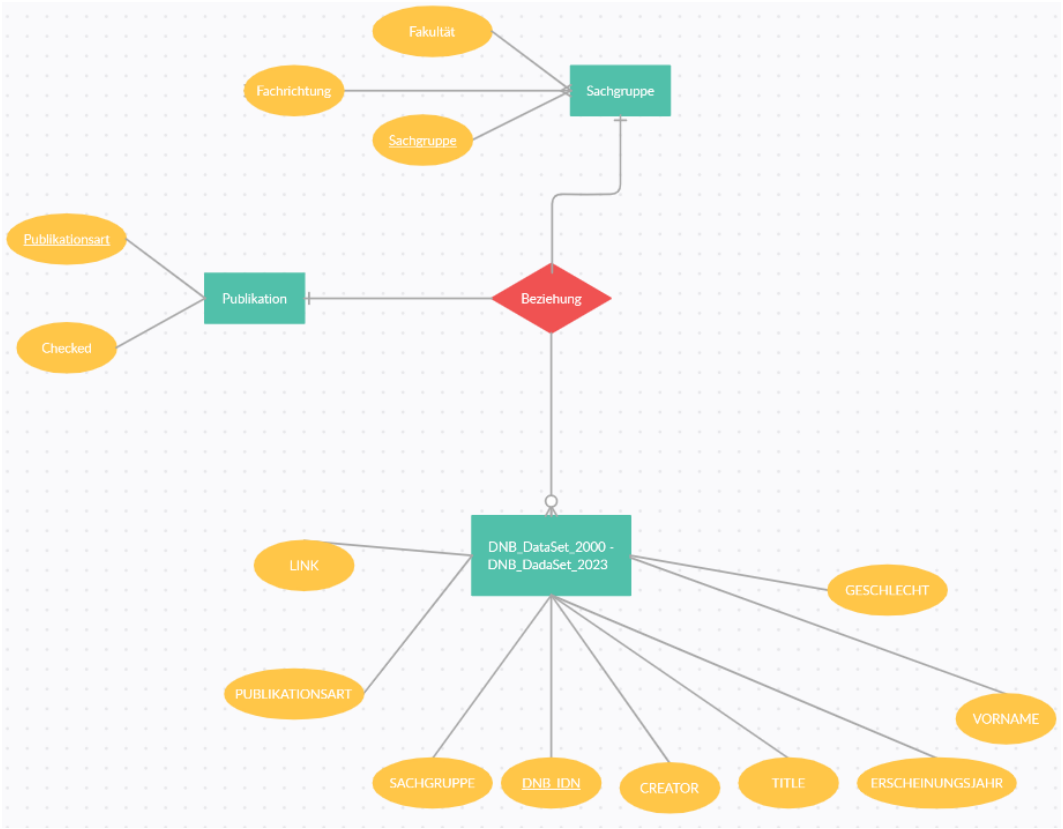
Bei Azure handelt es sich um eine Cloud-Plattform, die verschiedene Dienste, wie z.B. das Hosting von WebApps oder des SQL Servers. Somit kann sichergestellt werden, dass Komponenten von außen erreichbar sind und gleichzeitig die Administration und Wartung hauseigener Server entfällt. Dadurch können Arbeitsaufwand und Kosten reduziert werden.

Architektur

Grober Aufbau der Architektur



ER-Diagramm



Grundprinzip

Das Frontend der Web-Anwendung enthält die Bereiche „`Index`“, „`Search`“, „`Import`“. Das Frontend spricht über eine Dependency Injection Komponenten der ClassLibrary an, welche entweder einen SQL-Befehl an die Datenbank stellen oder die SRU-Schnittstelle der DNB abfragen, um die zurückgelieferten Daten anschließend in die Datenbank zu laden.

Index-Page

Die Index-Page ist die Startseite und zeigt dem User die Auswahl zwischen „`Search`“ und „`Import`“ an.

Import-Page

Die Import-Page bietet die Zusatzfunktion für das Updaten der Datenbank an. Hierbei werden die Jahre für die Aktualisierung abgefragt, sowie die optionalen Gender-API-Keys um das Geschlecht nicht nur über bereits vorkommende Vornamen in der Datenbank abzufragen, sondern auch über spezielle APIs, sobald die Datenbank kein Match liefert. Bei dem Import werden für jedes Jahr, welches innerhalb der angegebenen Zeitspanne liegt über das Backend Daten von der DNB über die sogenannte SRU-API abgefragt, welche von der DNB zur Verfügung gestellt wird. Die empfangenen Daten werden anschließend nach z.B. Creator, Title, Link etc. gefiltert und in eine „`DNBInsert_Model`“-Model geladen. Es wird vor dem Hinzufügen in die Datenbank überprüft ob der Vorname des Autors bereits zusammen mit dem Geschlecht in der Datenbank angelegt wurde. Sollte dieses nicht der Fall sein, wird versucht den Namen über die Gender-APIs zuzuordnen. Sollte kein Key angegeben worden sein, wird zunächst das Kostenlose Guthaben aufgebraucht. Sollte eine Art der Zuordnung erfolgreich sein (DB-Abfrage, API1, API2), wird werden die darauffolgenden Zuordnungsarten ignoriert umso die Last des Systems zu reduzieren und kosten einzusparen. Um zudem das benötigte Guthaben zu reduzieren, wird versucht andere Datensätze mit dem selben Vornamen mittels der über die API gefundene Zuordnung zu finden und zu aktualisieren.

Search-Page

Die Search-Page führt nach Eingabe und Bestätigung verschiedener Daten wie verschiedener Suchbegriffe, Jahreszahlen und/oder Fachrichtungen eine Suche innerhalb der Datenbank durch. Die Suche wird mittels einfachem Load-Operation für die übergebenen Parameter ausgeführt und jedes einzelne „`DNB_DataSet`“-Model in der „`data`“-Liste gespeichert. Hierbei sollte bei der Entwicklung und Warten darauf geachtet werden, dass die Sachgruppe „entpackt“ werden müssen, da es für verschiedene Fakultäten und Fachrichtung dieselbe Sachgruppenzuordnung geben kann (siehe Beispielsausgabe – Fakultätentabelle). Diese Darstellung wird in der ClassLibrary in dem „`Sachgruppe`“-Model nachgebildet.

	Sachgruppe	Fakultät	Fachrichtung
1	004	Informatik	Informatik
2	330	Informatik	Wirtschaft
3	610	Informatik	Medizin/Gesundheit
4	700	Architektur und Gestaltung	Bildende Kunst
5	710	Architektur und Gestaltung	Landschaftsgestaltung, Raumplanung
6	720	Architektur und Gestaltung	Architektur
7	730	Architektur und Gestaltung	Plastik, Numismatik, Keramik, Metallkunst
8	740	Architektur und Gestaltung	Grafik, angewandte Kunst
9	750	Architektur und Gestaltung	Malerei
10	760	Architektur und Gestaltung	Druckgrafik
11	770	Architektur und Gestaltung	Fotografie, Video, Computerkunst
12	004	Elektro- und Informationstechnik	Informatik

Die Daten werden aus der jeweiligen Datenbanktabelle geladen. Der Name dieser Tabellen setzen sich aus dem Präfix „`DNB_Dataset_`“ und dem jeweiligen Jahr, welches innerhalb des in der Suchmaske angegebenen Bereichs liegt. Das Modal welches sich anschließend öffnet, zeigt bei einer erfolgreichen Suche aufgrund der ansonsten entstehenden Performancegründen die durch das Rendering der HTML+CSS Elemente entstehen, nur 100 Sucherergebnisse, welches in der „`dataShow`“-Liste gespeichert wird. Das vor-/ und zurücklisten erfolgt über die Button „`Nächste Seite`“ und „`Vorherige Seite`“. Hierbei sollte vor allem beim Download der .xlxs-File geachtet werden, da das hierbei verwendete Package (Siehe Abschnitt: Verwendete NuGet-Packages) im Moment (Stand 25.01.2023) maximal 150 Datensätze pro File aufnehmen kann. Aus diesem Grund wird der „Zum Download hinzufügen“-Button verwendet, um pro Klick die jeweilige Seite als .xlxs-File vorzubereiten. Dabei können auch mehrere Files für verschiedene Seiten erzeugt werden. Durch den Klick auf den „Download“-Button werden die Files in einem .zip-File gebunden und zum Download angeboten. Die Dateien werden anschließend gelöscht um den Speicher des Servers nicht füllen. Die Favoriten-Anzeige wird in der Datenbank abgebildet. Beim Betätigen der Checkbox, wird für die jeweilige DNB_IDN die Änderung im dem „`Checked`“-Feld vorgenommen.

Erweiterte Mechanismen-Beschreibung

Der Sourcecode wurde kommentiert und es wurde versucht die Anwendung möglichst schmal und einfach zu halten. Anbei werden aber einige interessante Methodenaufrufe und der Zusammenspiel einzelner Komponente genauer erklärt.

Holen der Daten aus der SRU-DNB-Schnittstelle

Über den DNBExecutor-Service wird die DNB-SRU-Schnittstelle[2] angesprochen. Die Methode „`requestDNB`“ nimmt die Parameter entgegen und stellt einen GET-Request an die Schnittstelle. Das Format der Anfrage kann über die Dokumentation der Deutschen Nationalbibliothek über die SRU-Schnittstelle nachgeschlagen werden. Die zurückgeleiteten Daten werden weiter verarbeitet. Hierbei ist zu beachten, dass die Anfrage maximal 100 Datensätze zurückliefert (`maximumRecords`). Die maximumRecords, sowie die Position der des ersten Datensatzes wird über die von der Update-Page aufgerufenen Startmethode „`processingRequest`“ koordiniert.

```
private XDocument requestDNB(int position, int year, string publikationsart, int maxRecords)
{
    while (true)
    {
        string query;
        string url;

        string yearFilter = "jhr=(" + year + ")";
        string catalogFilter = "catalog=dnb.hss";
        query = publikationsart == "Dissertation" ? catalogFilter + " and " + yearFilter + " " +
            "and hss=diss*" : catalogFilter + " and " + yearFilter + " and hss=habil*";

        url = @"https://services.dnb.de/sru/dnb?version=1.1&operation=searchRetrieve&query=" + query + "" +
            "&startRecord=" + position + "&recordSchema=MARC21-xml&maximumRecords=" + maxRecords;

        var request = WebRequest.Create(url);
        request.Method = "GET";

        try
        {
            var webResponse = request.GetResponse();
            var webStream = webResponse.GetResponseStream();
            var reader = new StreamReader(webStream);
            var data = reader.ReadToEnd();

            return XDocument.Load(url);
        }
        catch
        {
            Console.Write("ERROR: TimeOut");
        }
    }
}
```

Innerhalb der verschiedenen „`Get`“-Methoden der Klasse werden die Daten gefiltert. Dabei wird zunächst ausgewählt ob man sich im „`datafield`“ oder „`controlfield`“ der XML-Datei bewegen möchte. Anschließend wird nach dem „`tag`“ und „`code`“ gefiltert. Dieser kann über die MARC 21 „Feldbeschreibung Titeldaten“[3] nachgeschlagen werden.

```
private string getIDN_FromRecords(XElement record)
{
    try
    {
        return record.Descendants(XNamespace.Get("http://www.loc.gov/MARC21/slim") +
            "controlfield").First(child => child.Attribute("tag").Value == "001").Value;
    }
    catch
    {
        return "";
    }
}
```

```
private string getCreator_FromRecords(XElement record)
{
    try
    {
        return record.Descendants(XNamespace.Get("http://www.loc.gov/MARC21/slim") +
            "datafield").ToList().FindAll(child => child.Attribute("tag").Value == "100").
            Elements().First(child => child.Attribute("code").Value == "a").Value;
    }
    catch
    {
        try
        {
            return record.Descendants(XNamespace.Get("http://www.loc.gov/MARC21/slim") +
                "datafield").ToList().FindAll(child => child.Attribute("tag").Value == "110").
                Elements().First(child => child.Attribute("code").Value == "a").Value;
        }
        catch
        {
            try
            {
                return record.Descendants(XNamespace.Get("http://www.loc.gov/MARC21/slim") +
                    "datafield").ToList().FindAll(child => child.Attribute("tag").Value == "700").
                    Elements().First(child => child.Attribute("code").Value == "a").Value;
            }
            catch
            {
                return "";
            }
        }
    }
}
```

[2] vlg. (Deutsche Nationalbibliothek, 2019b)
[3] vgl. (Deutsche Nationalbibliothek, 2019a)

Einfügen der Daten in der Datenbank

Die gefilterten Daten aus der SRU-Schnittstelle werden vom DNBExecutor-Service an den SQLExecutor-Service weitergeleitet. Die „`InsertDNBData`“-Methode versucht zunächst aus den Autordaten den Vornamen abzuleiten. Um diesen Vornamen anschließend ein Geschlecht zuzuordnen. Hierbei wird zunächst versucht aus den bereits zugeordneten Vornamen innerhalb der Datenbank Rückschlüsse auf den das Geschlecht des momentan zu verarbeitenden Datensatz zu treffen. Hierfür werden durch die Methode „GetGender“ alle Tabellen mit dem Präfix „`DNB_Dataset_`“ durchsucht. Hierbei können innerhalb der Datenbank die Werte „`male`“, „`female`“, sowie „`undefined`“ für das Geschlecht vorkommen.

```
public string? GetGender(string vorname)
{
    string dbPrefix = "DNB_Dataset_";
    int minYear = GetMinJahr();
    int maxYear = GetMaxJahr();

    for (int year = minYear; year <= maxYear; year++)
    {
        string sql = "SELECT GESCHLECHT FROM " + dbPrefix + year + " WHERE VORNAME = '" + vorname + "' ";
        string? genderResult = _db.LoadData<string, dynamic>(sql, new { }).FirstOrDefault();

        if (genderResult != null)
        {
            return genderResult;
        }
    }
    return null;
}
```

Sollte durch die Datenbank keine Zuordnung des Vornamen möglich sein, wird durch die Methode „`GetGender`“ null zurückgeliefert und es wird zunächst versucht durch die „Genderize“-API das Geschlecht zuzuordnen, ansonsten wird eine Zuordnung durch die „Gender-API“-API versucht. Sollte ein Key eingetragen sein um sein Guthaben bei den APIs aufzuladen wird dieser verwendet, ansonsten wird das kostenlose Guthaben von jeweils 100 Credits verwendet.

```
string? gender = GetGender(vorname);

if (gender == null)
{
    try
    {
        string httpResponse = genderizeKey != null ? await httpClient.GetStringAsync(@"https://api.genderize.io/?name=" +
            vorname + "&apikey=" + genderizeKey) : await httpClient.GetStringAsync(@"https://api.genderize.io/?name=" + vorname);

        dynamic json = JsonConvert.DeserializeObject(httpResponse)!;

        gender = json["gender"];
    }
    catch
    {
        gender = null;
    }
}
if (gender == null)
{
    try
    {
        string httpResponse = genderizeKey != null ? await httpClient.GetStringAsync(@"https://gender-api.com/get?name=" +
            vorname + "&key=" + genderapiKey) : await httpClient.GetStringAsync(@"https://gender-api.com/get?name=" + vorname);

        dynamic json = JsonConvert.DeserializeObject(httpResponse)!;

        gender = json["gender"];
    }
    catch
    {
        gender = null;
    }
}
```

Sollte hierbei auf irgendeine der Zuordnungsmethoden eine Zuordnung geklappt haben werden die anderen Zuordnungsmethoden nicht ausgeführt und die Daten werden direkt in die Datenbank geladen. Sollte die Zuordnung erfolglos gewesen sein, wird „`undefined`“ in der Datenbank abgespeichert.

Bereitstellung einer API

Die Anwendung stellt eine API bereit, die unter https://prosper-db.azurewebsites.net/api/Prosper erreichbar ist. https://prosper-db.azurewebsites.net/ ist hierbei eine Beispiels-URL, unter der sich die Prosper-Anwendung befinden könnte. Diese Beispiels URL wird in diesem Teilabschnitt zur besseren Erklärung verwendet.

Unter https://prosper-db.azurewebsites.net/api/Prosper ist die Syntax der API aufgelistet:

- `URL?releaseStart={int}&releaseEnd={int}`
- `URL?releaseStart={int}&releaseEnd={int}&sachgruppe={string}`
- `URL?releaseStart={int}&releaseEnd={int}&sachgruppe={string}&suchBegriff_1={string}`
- `URL?releaseStart={int}&releaseEnd={int}&sachgruppe={string}&suchBegriff_1={string}&suchBegriff_2={string}`
- `URL?releaseStart={int}&releaseEnd={int}&sachgruppe={string}&suchBegriff_1={string}&suchBegriff_2={string}&suchBegriff_3={string}`
- `URL/Fakultaeten`

Ein Beispiel für die Ausgabe der in der Datenbank eingepflegten Fakultäten-Tabelle wäre unter der folgenden Suchabfrage möglich Suchanfrage wäre also wie folgt aufgebaut:

- `https://prosper-db.azurewebsites.net/api/Prosper?Fakultaeten`

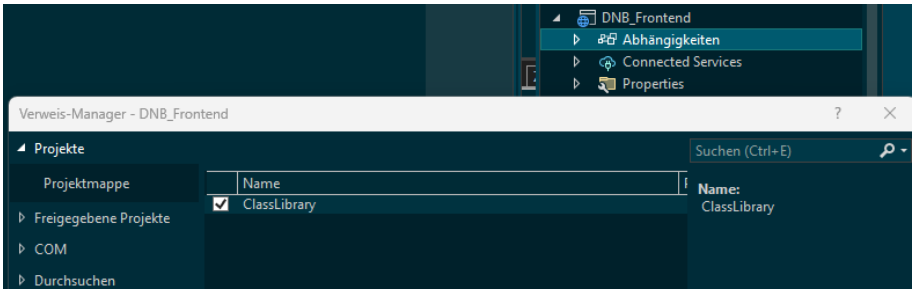
Die Ausgabe der zurückgelieferten Ergebnisse erfolgt im JSON-Format und ist unlimitiert. Die Anfrage an die Datenbank erfolgt über das selbe Schema wie innerhalb der Anwendung. Aus Security-Sicht wurde der SQLExecutor, der für die API verwendet wird jedoch so angepasst, dass dieser nur Load-Befehle an die Datenbank senden kann.

Besonderheiten

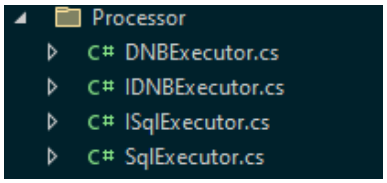
In diesem Abschnitt werden auf verschiedene Besonderheiten eingegangen um bestimmte Dinge, die gerne vergessen werden nachträglich anzusprechen und den Entwickler nicht mit einem Fragezeichen zurückzulassen.

Kommunikation Backend – Frontend

Damit das Frontend Komponente des Backends ansprechen kann (z.B. über `@using`), muss zunächst ein Projektverweis vorhanden sein. Hierfür geht man über den Rechtsklick auf „Abhängigkeiten“ innerhalb des Fronends und anschließend auf „Projektverweis hinzufügen“. Es öffnet sich der Verweis-Manager, indem unter Projekt→Projektmappe das jeweilige Projekt angehakt werden muss. Anschließend bestätigt man die Änderungen. Dies sollte lediglich bei der Erstellung eines neuen Projektes beachtet werden.



Die Kommunikation von Frontend und Backend, sowie zwischen einzelnen Komponenten des Frontends erfolgt über Dependency Injection. Hierfür sollte zunächst sollte für die jeweilige Klasse, die einen Service anbietet eine Schnittstelle implementiert werden.



Innerhalb des Frontends sollte unter der Klasse „`Programm.cs`“ der Service registriert werden und innerhalb der Blazor-Komponente (Page) der Service injected werden.

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddTransient<ISqlDataAccess, SqlDataAccess>();
builder.Services.AddTransient<ISqlExecutor, SqlExecutor>();
builder.Services.AddTransient<IDNBExecutor, DNBExecutor>();
```

```
@using ClassLibrary.Processor;
@using ClassLibrary.Model;
@using ClassLibrary;
@using GemBox.Spreadsheet;
@using System.IO;
@using System.IO.Compression;
@using Newtonsoft.Json;

@inject ISqlExecutor _db
@inject IJSRuntime JSRuntime
```

Connection Datenbank

Da die Anwendung über eine Kommunikation der Datenbank funktioniert, muss eine Verbindung zur Datenbank erfolgen. Dies funktioniert über einen sogenannten Connection-String, eine Zeichenfolge, mit deren Hilfe die Anwendung eine Verbindung zur Datenbank aufbauen kann. Diese kann unter der Date „`appsettings.json`“ eingetragen werden.

```
{
  "ConnectionStrings": {
    "Default": "CONNECTION-STRING HIER EINTRAGEN"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Verwendete NuGet-Packages

Dieser Abschnitt beschreibt wo die Packages verwendet werden und welchen Zweck diese erfüllen.

Pakete Frontend

- *Blazorise || Blazorise.Bootstrap || Blazorise.Icons.FontAwsome*
  - Diese Packages stellen die verwendete Komponenten aus Blazorise bereit. Über mögliche Komponenten, sowie Einsatzmöglichkeiten kann sich der Entwickler über die Blazorise-Dokumentation[4] informieren.
  - Die Packages werden als Service in der „`Program.cs`“ registriert und innerhalb der „`_Imports.razor`“ als `@using` hinzugefügt, um das Ansprechen der Komponente zu vereinfachen. Für eine genaue Anleitung sehen Sie sich den Quickstart Guide[5] von Blazorise an
- *Gembox.Spreadsheet*
  - Dieses Package wird verwendet um eine .xlxs-Datei anzulegen, diese zu formatieren und die Daten, welche in der Search-Page nach dem Laden angezeigt werden in die Datei zu laden. Das Limit ist in der kostenlosen Version auf 150 Zeilen in der Datei beschränkt.
  - Das Packages muss innerhalb der „`Programm.cs`“ registriert werden. Hierbei wird der Key angegeben. Sollte diese kostenlose Version verwendet werden, wird hierbei „`FREE-LIMITED-KEY`“ eingetragen

    ```
    SpreadsheetInfo.SetLicense("FREE-LIMITED-KEY");
    ```
  - Für weitere Information kann sich der Entwickler unter der GemBox-Dokumentation[6] informieren

Pakete Backend

- *Dapper || Dapper.Contrib*
  - Dapper ist ein SQL-Mapper, welcher durch den jeweiligen Dapper-Methodenaufruf, sowie den als String eingegeben SQL-Befehl einen Befehl an die verbundene Datenbank sendet und optional dabei Daten empfängt
- *System.Configuration.ConfigurationManager || Microsoft.Extensions.Configuration*
  - Nutzung für die das holen der Connectionstring, sowie das Verarbeiten dieses
- *System.Data.SqlClient*
  - Wird zusätzlich verwendet um eine Verbindung zur Datenbank aufzubauen und zusammen mit Dapper einen Befehl zu senden
- *Newtonsoft.Json*
  - Wird genutzt um mit JSON-Dateien umzugehen um diese so z.B. zu versenden oder zu empfangen

[4] vlg. (Blazorise documentation, ohne Datum)
[5] vlg. (Blazorise quick start guide, ohne Datum)
[6] vlg. (Fitzpatrick, 2017)