

University of Applied Sciences

# SEAR<sup>C</sup>H–Tag EEXCESS SECH–Browser

## Programmier Leitfaden Spezifikation, Konstruktion

Prof. Dr. Peter Stöhr      Gottfried von Recum  
Alexander Pöhlmann      Lothar Mödl      Burak Erol  
Brian Mairhörmann      Andreas Netsch      Philipp Winterholler  
                                 Andreas Ziemer

Version 0.4.1

# Inhaltsverzeichnis

<b>I. Einleitung</b>	<b>2</b>
<b>1. Übersicht</b>	<b>3</b>
1.1. Abschnitt . . . . .	3
1.1.1. Teilabschnitt . . . . .	3
1.1.1.1. Unterteilabschnitt . . . . .	3
<b>2. User Interface</b>	<b>4</b>
2.1. Abschnitt . . . . .	4
2.1.1. Teilabschnitt . . . . .	4
2.1.1.1. Unterteilabschnitt . . . . .	4
<b>3. Programmlogik</b>	<b>5</b>
3.1. Abschnitt . . . . .	5
3.1.1. Teilabschnitt . . . . .	5
3.1.1.1. Unterteilabschnitt . . . . .	5
<b>II. Spezifikation</b>	<b>6</b>
<b>4. Browser</b>	<b>7</b>
4.1. Menüführung . . . . .	7
4.2. Lesezeichenverwaltung . . . . .	8
4.3. SEARCH-Tagverwaltung . . . . .	8
4.4. Einstellungen . . . . .	9
<b>III. Konstruktion</b>	<b>12</b>
<b>5. Übersicht</b>	<b>13</b>
<b>6. User Interface</b>	<b>14</b>
6.1. ViewController . . . . .	14
6.2. Delegate . . . . .	16
6.2.1. AppDelegate . . . . .	16
6.2.2. WebViewDelegate . . . . .	16
6.2.3. readHead.js . . . . .	16
6.2.4. FavTableDelegate . . . . .	16

6.2.5.	SechTableDelegate	16
6.3.	DataSource	17
6.3.1.	FavTableDataSource	17
6.3.2.	SechTableDataSource	17
6.4.	Components	18
6.5.	Persistence	19
6.5.1.	FavoritesModel	19
6.6.	WebContent	20
<b>7.</b>	<b>Programmlogik</b>	<b>21</b>
7.1.	TaskCtrl	21
7.2.	SEARCHExtraction	22
7.2.1.	SearchManager	22
7.2.2.	RegexForSEARCH	22
7.3.	SEARCHModels	23
7.4.	QueryCreation	24
7.4.1.	QueryCreationCtrl	24
7.5.	SearchQueryys	25
7.6.	QueryResolution	26
7.6.1.	JSONData	26
7.6.2.	QueryBuild	27
7.6.2.1.	AbstractBuilder	28
7.6.2.2.	AbstractJSONBuilder	28
7.6.2.3.	AbstractURLBuilder	28
7.6.2.4.	EexcessJSONBuilder	28
7.6.2.5.	FarooURLBuilder	28
7.6.2.6.	DuckDuckGoURLBuilder	28
7.6.2.7.	EEXCESSOrigin	28
7.6.3.	QuerySend	29
7.6.3.1.	AbstractConnectionCtrl	29
7.6.3.2.	JsonConnectionCtrl	29
7.6.3.3.	URLConnectionCtrl	29
7.6.4.	ResponseParse	30
7.6.4.1.	FarooResponseParser	30
7.6.4.2.	DuckDuckGoResponseParser	30
7.6.4.3.	EexcessResponseParser	30
7.6.4.4.	AbstractResponseParser	30
7.7.	SearchResults	31
7.8.	Ranking	32
7.8.1.	Rules	32
7.8.2.	SearchRules	32
7.8.3.	RankingPersistence	32
7.8.3.1.	PersistenceController	32

7.8.3.2.	RankingDataObject . . . . .	32
7.8.3.3.	RankingDataObjectPersistency . . . . .	32
7.9.	SettingsModel . . . . .	33

# Teil I.

## Einleitung

# **1. Übersicht**

## **1.1. Abschnitt**

### **1.1.1. Teilabschnitt**

#### **1.1.1.1. Unterteilabschnitt**

#### **Paragraph**

##### **Unterparagraph**

1. Eine
2. kleine
3. Aufzählung

## **2. User Interface**

### **2.1. Abschnitt**

#### **2.1.1. Teilabschnitt**

##### **2.1.1.1. Unterteilabschnitt**

##### **Paragraph**

###### **Unterparagraph**

1. Eine
2. kleine
3. Aufzählung

## **3. Programmlogik**

### **3.1. Abschnitt**

#### **3.1.1. Teilabschnitt**

##### **3.1.1.1. Unterteilabschnitt**

#### **Paragraph**

##### **Unterparagraph**

1. Eine
2. kleine
3. Aufzählung



# Teil II.

## Spezifikation

## 4. Browser

Im Folgenden werden die Use-Case-Diagramme des SECH-Browsers übersichtlich vorgestellt. Diese veranschaulichen alle verschiedenen Funktionalitäten, die der Benutzer tätigen kann.

### 4.1. Menüführung

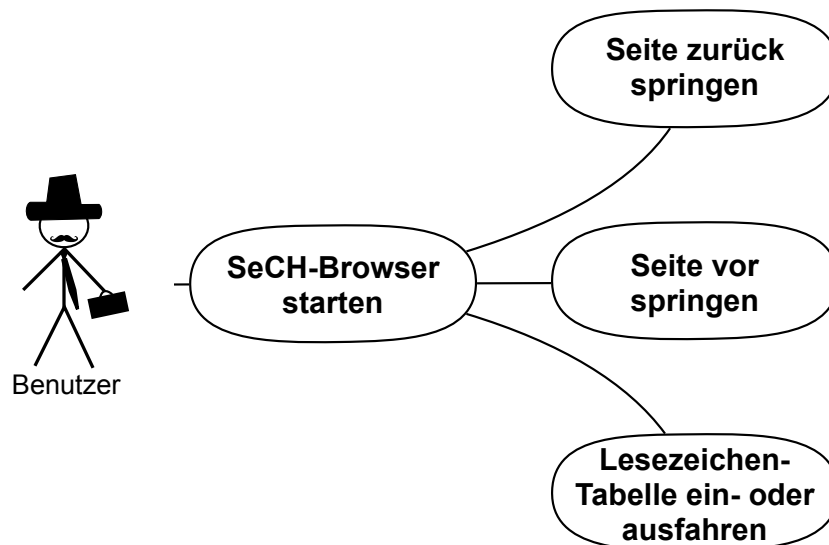


Abbildung 4.1.: Navigationselemente

Dieses Use-Case-Diagramm zeigt die wesentlichen Funktionen des Browsers an. Der Nutzer kann auf einer Homepage eine Seite zurück- und vorspringen. Weiterhin hat er die Möglichkeit die Lesezeichentabelle auszufahren und im Anschluss diese wieder einzufahren.

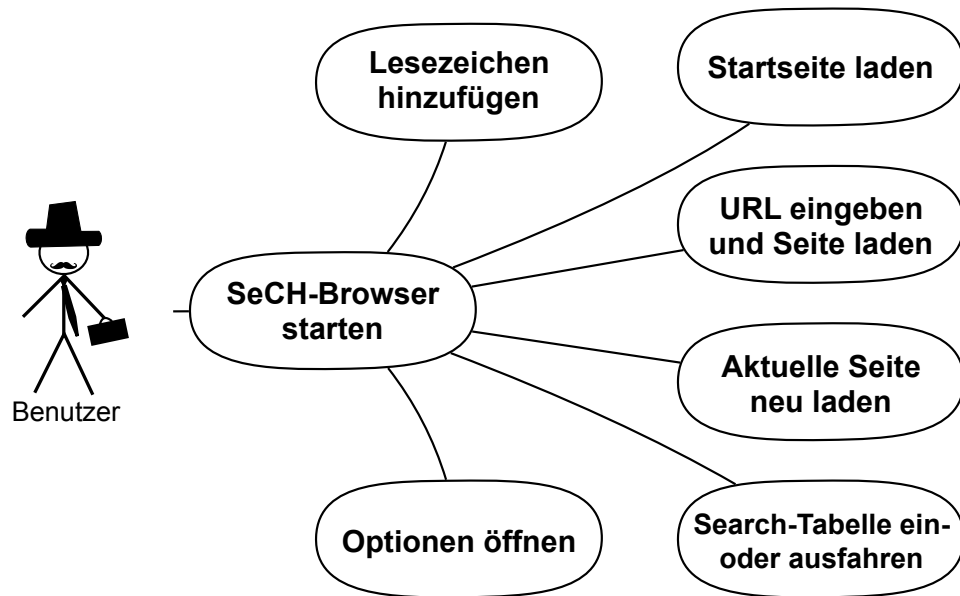


Abbildung 4.2.: Navigationselemente

In diesem Use-Case-Diagramm werden weitere Navigationselemente des Browsers vorgestellt. Der Nutzer kann eigene Lesezeichen hinzufügen, die von ihm definierte Startseite laden, eine URL in die Addressbar eingeben und diese dann laden, die aktuelle Seite neu laden, die  $\text{SEAR}^{\text{CH}}$ -Tabelle aus- und einfahren und das Fenster für Optionen öffnen.

## 4.2. Lesezeichenverwaltung

Ist die Lesezeichentabelle ausgefahren, so kann der Benutzer ein persönlich angelegtes Lesezeichen auswählen und es wird die von ihm gewünschte Seite geladen. Das Löschen und das Editieren ausgewählter Lesezeichen ist ebenfalls in der Tabelle verwirklichtbar.

## 4.3. $\text{SEAR}^{\text{CH}}$ -Tagverwaltung

Der Nutzer kann durch zwei verschiedene Wege Suchergebnisse für das gewünschte  $\text{SEAR}^{\text{CH}}$ -Tag anzeigen lassen. Zunächst muss der Nutzer durch Eingabe und Bestätigung einer

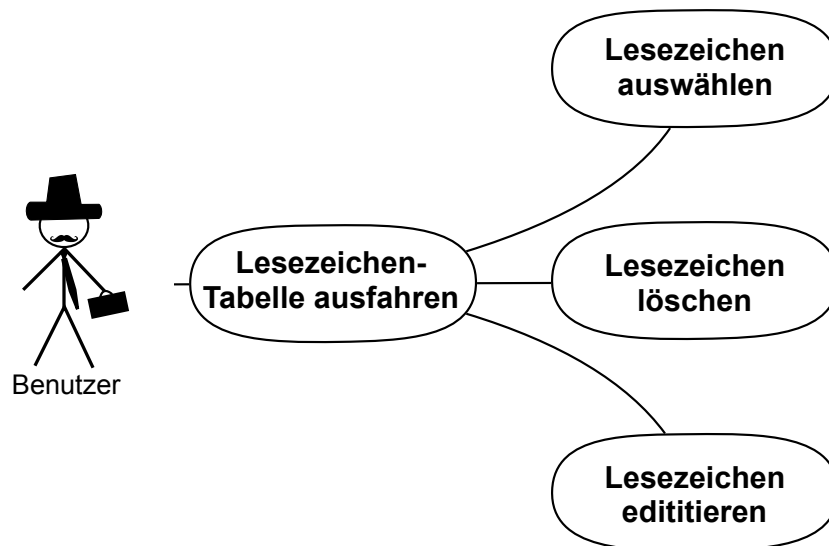


Abbildung 4.3.: Lesezeichenverwaltung

URL die Seite aufrufen. Die erste Variante ist das gewünschte  $\text{SEARCH}$ -Tag, welches hervorgehoben wird, anzuklicken. Die zweite Variante ist durch Ausfahren der  $\text{SEARCH}$ -Tabelle das gewünschte  $\text{SEARCH}$ -Tag auszuwählen. Beide Wege führen zum selben Ergebnis und zwar die Anzeige des ersten Suchergebnisses des jeweiligen  $\text{SEARCH}$ -Tags. Zusätzlich kann der Nutzer alle weitere Suchergebnisse zu einem  $\text{SEARCH}$ -Tag anzeigen lassen.

## 4.4. Einstellungen

Befindet sich der Nutzer in den Browsereinstellungen, so hat er die Möglichkeit bestimmte Suchmaschinen, welche für die Suchergebnisse der  $\text{SEARCH}$ -Tags verwendet werden, zu aktivieren oder diese zu deaktivieren. Im Weiteren kann der Benutzer sein persönliches Nutzerprofil verwalten und seine Startseite für die Applikation festlegen, welche direkt nach dem Öffnen der Anwendung als erste Seite geladen wird.

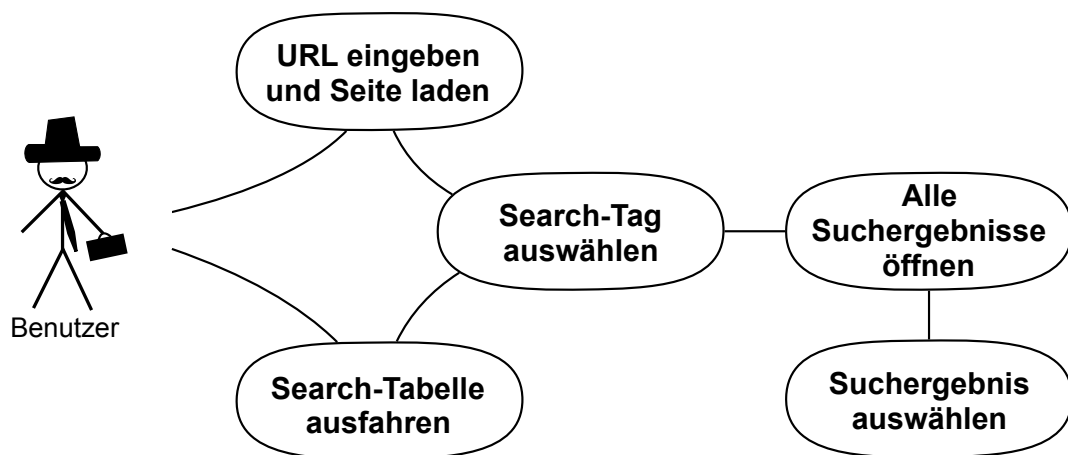


Abbildung 4.4.: SEARCH-Tagverwaltung

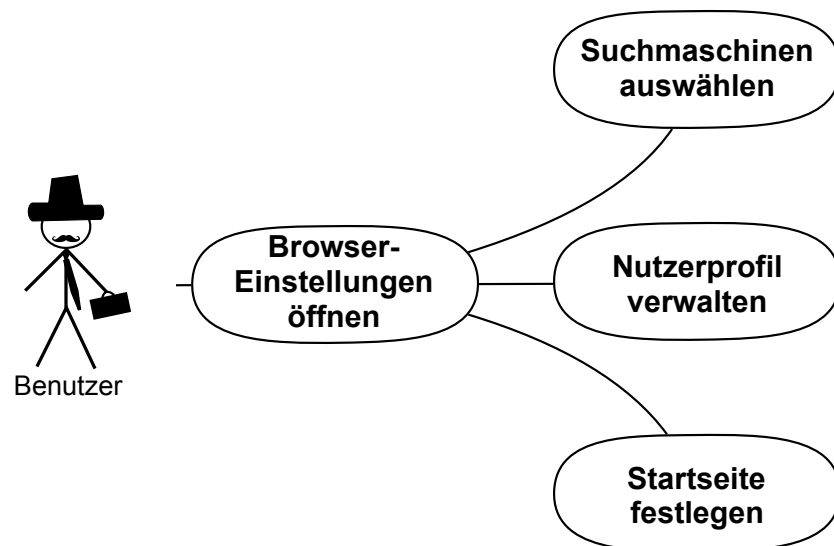


Abbildung 4.5.: Einstellungen

# Teil III.

## Konstruktion

## 5. Übersicht

Die Konstruktion gliedert sich in drei Abschnitte:

1. Übersicht
2. User Interface
3. Programmlogik



## 6. User Interface

Im Folgenden wird ein Überblick über die Klassenabhängigkeiten des UIs gegeben, der Verlauf zur Anzeige eines SEARCH-Links erklärt, sowie die Packagebedeutungen innerhalb des UIs erläutert.

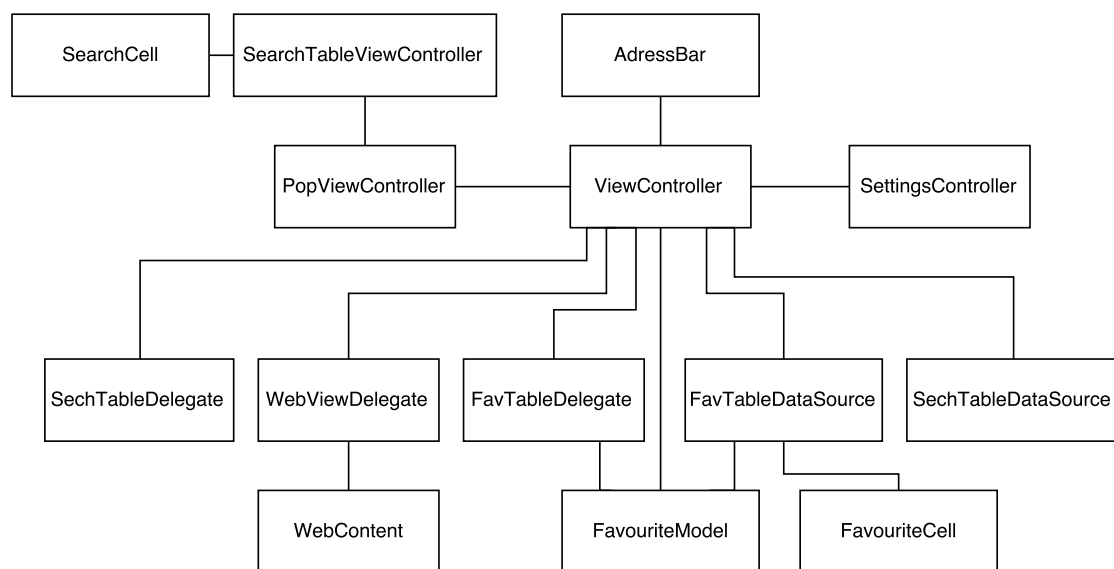


Abbildung 6.1.: Klassenabhängigkeiten im UI

Die obige Darstellung zeigt die Verbindungen der Klassen des User Interfaces. Hierbei steht der ViewController im Mittelpunkt, da von ihm die meiste Funktionalität ausgeht.

Obige Grafik zeigt den Programmablauf beim Klick auf einen Searchtag. Die Position des Klicks wird über JavaScript ausgelesen und an den WkWebView weitergeleitet. Dieser erzeugt ausgehend von der übertragenen Position einen PopView und zeigt diesen an.

## 6.1. ViewController

Der **ViewController** dient zur Steuerung der Funktionen welche direkt mit dem Interface zusammen gehören. Dazu gehören **IBActions** welche von Buttons oder anderen Interaktionsflächen angestoßen werden können, sowie Funktionen welche direkt mit diesen zusammenhängen, beziehungsweise aus diesen resultieren. Der **ViewController** ist die zweite Schicht nach dem UI. Im Folgenden werden die einzelnen **ViewController**

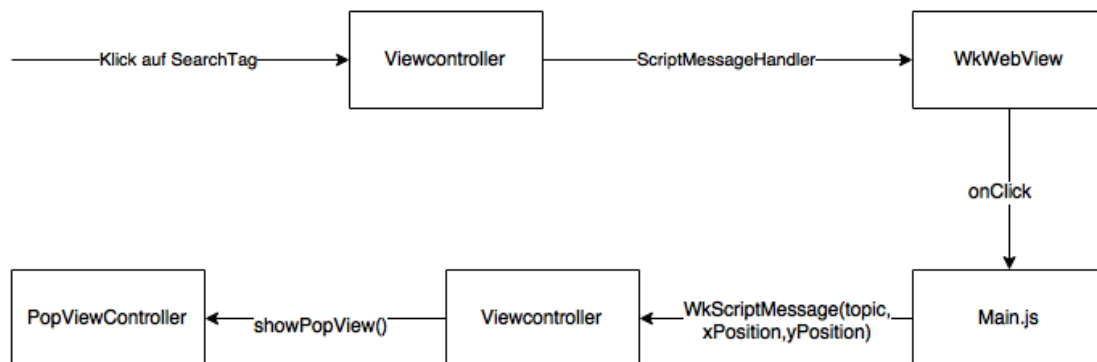


Abbildung 6.2.: Searchtag Anzeige

mit ihren jeweiligen Funktionen beschrieben. Der **Viewcontroller** ist der Hauptviewcontroller für die normale Browseransicht mit der Adresszeile und dem **WkWebView**. Er verwaltet die Kopfzeile mit Adresszeile sowie den Buttons für Vorwärts, Zurück, Lesezeichen, Lesezeichen hinzufügen, Reload, **SEAR<sup>C</sup>H**-Tabelle sowie die Settings. Die Datei **Main.js** welche aus dem **Viewcontroller** heraus aufgerufen wird, markiert alle gefundenen **SEAR<sup>C</sup>H**-Tags. Außerdem wird im Falle eines Klicks die Position sowie die id des **SEAR<sup>C</sup>H**-Tags übermittelt. Der **PopViewController** ist zuständig für die Verwaltung eines neuen **PopViews**, beim klicken auf einen **SEAR<sup>C</sup>H**-Tag. Er verwaltet die Anzeige des ausgewählten **SEAR<sup>C</sup>H**-Links in einem neuen **WkWebView** sowie den Button für die Auswahl der verschiedenen Suchergebnisse zu einem **SEAR<sup>C</sup>H**-Tag. Der **SEAR<sup>C</sup>H**-**TableViewCellController** verwaltet die einzelnen Suchergebnisse zu einem **SEAR<sup>C</sup>H**-Tag. Alle Ergebnisse werden sortiert in einer Tabelle mit Bild, Title und Link angezeigt. Der **Settingscontroller** ist für die Verwaltung der Browsereinstellungen zuständig.

## 6.2. Delegate

Die Delegates im SECH-Browser dienen zur Bereitstellung einer einheitlichen Struktur innerhalb des Programms. Von den einzelnen Delegate Klassen werden Funktionen zur Darstellung von Informationen oder interne Protokolle aufgerufen. Im `WebViewDelegate` werden die Funktionen zur Validierung der Website mit Hilfe der JavaScript Dateien und die Extraktion der SEAR<sup>C</sup>H-Tags angestoßen.

Die `readHead.js` enthält den Javascript Code zur Auslesen des Headbereiches einer HTML-Seite. In der Klasse `FavTableDelegate` werden die Lesezeichen bereitgestellt. Der Benutzer hat die Möglichkeit ein ausgewähltes Lesezeichen auszuwählen und es sich im Browser anzeigen zulassen. Zusätzlich enthält die Klasse die Funktionen, um Lesezeichen bearbeiten und löschen zu können. Der `SechTableDelegate` enthält die Routine zum Darstellen von SEAR<sup>C</sup>H-Links aus der SECH-Tabelle.

### 6.2.1. AppDelegate

### 6.2.2. WebViewDelegate

### 6.2.3. readHead.js

### 6.2.4. FavTableDelegate

### 6.2.5. SechTableDelegate

## 6.3. DataSource

Die DataSources im SECH-Browser dienen zur Bereitstellung der Daten in den Tabellen. Von den einzelnen Klassen werden jeweils Funktionen zum befüllen der einzelnen Zellen aufgerufen. In der Klasse **FavTableDataSource** werden zunächst alle gespeicherten Lesezeichen geladen, anschließend werden die einzelnen Zellen der Tabelle mit den Lesezeichen befüllt. Im **SechTableDataSource** werden alle SEARCH-Tags die auf der Seite gefunden wurden in den Zellen der SECH-Tabelle dargestellt. Zusätzliche befinden sich noch Funktionen zum Erweitern der SEARCH-Tags in der Klasse.

### 6.3.1. FavTableDataSource

### 6.3.2. SechTableDataSource

## 6.4. Components

In den Components werden die spezifischen Zellen für die Lesezeichentabelle und SEARCH-Tagtabelle modelliert, die in den jeweiligen Delegates verwendet werden. Weiterhin ist in Components die Validierung für die Addressbar definiert, welche die Eingaben in der Suchleiste überprüft und verwaltet.

## 6.5. Persistence

Im Persistence-Package befinden sich Models die für eine persistente Speicherung gedacht sind. Das **FavouritesModel** stellt einerseits das nötige Model, andererseits die Funktion des Speicherns für Lesezeichen zur Verfügung.

### 6.5.1. FavoritesModel

## 6.6. WebContent

Der **WebContent** stellt die Schnittstelle zwischen dem UI und der **SearchExtraction** dar. Er beinhaltet HTML-Code, in Form eines Strings, sowie die URL, von welcher der HTML-Code stammt. Der HTML-Code ist in den Head und den Body der Website aufgeteilt. Aus dem **WebContent** wird in der **SearchExtraction** ein **SearchModel** erzeugt.

## 7. Programmlogik

### 7.1. TaskCtrl

Der `TaskCtrl` ist einer der wichtigsten Elemente bei der Kommunikation zwischen der Oberfläche und dem Abrufen der Daten aus dem Internet. Er bestimmt mit Hilfe der vom Nutzer getroffenen Einstellungen an welche Suchmaschine die Anfrage geschickt wird. Die Einstellungen werden durch das `Settings.bundle` definiert und können durch die Klasse `SettingsManager` abgerufen werden.

Der `TaskCtrl` wird von der Klasse `SearchResults` initialisiert und aufgerufen. Zuerst wird unter Verwendung der Klasse `QueryCreationCtrl` ein allgemeingültiges Format für die Suchanfragen erstellt. Dieses allgemeingültige Format wird anschließend von den jeweiligen `QueryBuilder`n (`EEXCESSJSONBuilder`, `DuckDuckGoURLBuilder` und `FarooURLBuilder`) in das für die jeweilige Suchmaschine passende Format umgewandelt. Die Suchanfragen werden asynchron durch die entsprechenden `ConnectionCtrls` (`JSONConnectionCtrl` und `URLConnectionCtrl`) versendet. Bei erfolgreicher Suche und nach erfolgreichem Parsen der Ergebnisse werden diese mit Aufruf der Methode `setRecommendation(status:String, message:String)` an den Aufrufer, die Klasse `SearchResults`, zurückgegeben. Diese speichert die Suchergebnisse in einem Cache zwischen. Der `TaskCtrl` wird nur aufgerufen, falls zu dem jeweiligen `SEARCH`-Tag noch keine Ergebnisse im Cache vorhanden sind.



## 7.2. SEARCHExtraction

Die SearchExtraction nimmt ein **WebContent**-Objekt entgegen und verarbeitet es zu einem **SearchModels**-Objekt. Hierbei wird der HTML-Code, der im **WebContent** enthalten ist, auf Search-Tags abgesucht. Diese Search-Tags werden dann zu **SearchModel**-Objekten zusammengebaut und in einem **SearchModels**-Objekt gesammelt.

Für den Ablauf der Analyse und Erzeugung ist der **SearchManager** zuständig. Die Methoden für die Analyse des HTML-Codes stellt die **RegexForSEARCH** zur Verfügung.

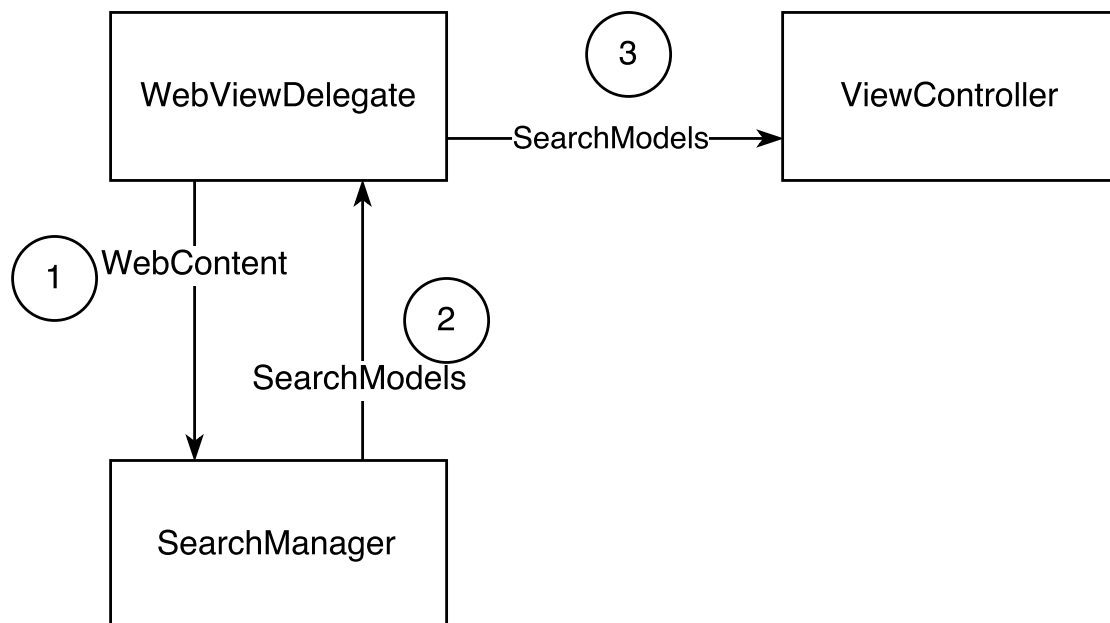


Abbildung 7.1.: Umwandeln eines **WebContent**-Objekts in ein **SearchModels**-Objekt

Die obige Darstellung beschreibt den Ablauf und die Übergabestellen der Schnittstellen **WebContent** und **SearchModels**. Der **WebContent**, erzeugt vom **WebViewDelegate**, wird in Schritt 1 dem **SearchManager** übergeben. Der **SearchManager** erzeugt aus dem **WebContent** die **SearchModels** und gibt diese in Schritt 2 an den **WebViewDelegate** zurück. Der **WebViewDelegate** übergibt dann das **SearchModels**-Objekt an den **ViewController**, wo es zur Verfügung für die anderen Packages steht.

### 7.2.1. SearchManager

### 7.2.2. RegexForSEARCH

## 7.3. SEARCHModels

Das **SearchModels** stellt die Schnittstelle zwischen der **SearchExtraction** und dem **TaskController** dar. Von der **SearchExtraction** erzeugt, beinhaltet ein **SearchModels**-Objekt eine Sammlung von **SearchModel**-Objekten erzeugt aus einer Website. In einem **SearchModel**-Objekt werden Search-Tag-Links mit der zugehörigen Search-Tag-Section/-Head und Filtern gespeichert.

Zur globalen Identifikation wird in einem **SearchModel** die Url der Website von der es kommt gesetzt. Um ein **SearchModel** innerhalb einer Seite zu finden wird der Index gesetzt.

## 7.4. QueryCreation

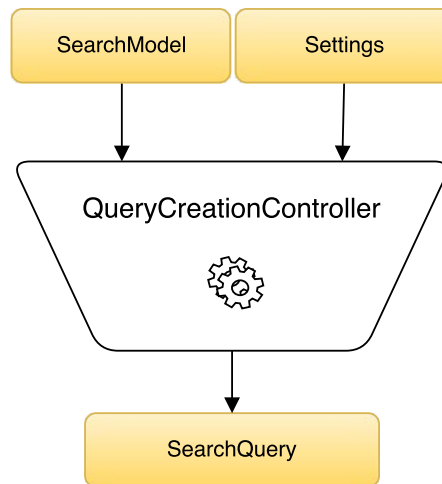


Abbildung 7.2.: Aufbau des Moduls QueryCreation

QueryCreation generiert aus den Suchwörtern (SearchModel) und den Geräte- und App-Einstellungen ein allgemeines Anfrageformat. Dieses Anfrageformat wird in QueryResolution zu einem für jede Suchmaschine spezifischen Suchformat umgewandelt.

### 7.4.1. QueryCreationCtrl

## 7.5. SearchQueryys

## 7.6. QueryResolution

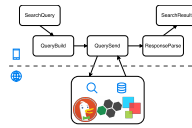


Abbildung 7.3.: Übersicht des Moduls QueryResolution

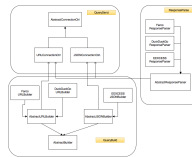


Abbildung 7.4.: Aufbau des Moduls QueryResolution

### 7.6.1. JSONData

### 7.6.2. QueryBuild

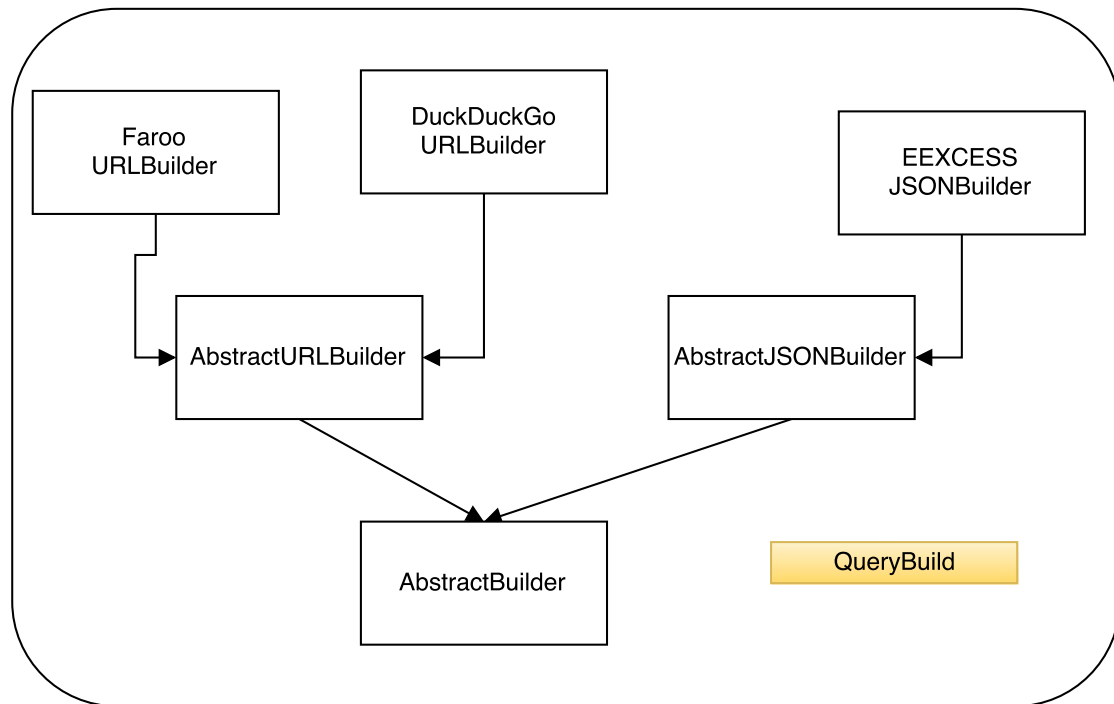


Abbildung 7.5.: Aufbau des Moduls QueryBuild

Das Modul QueryBuild liefert den größten Teil der Intelligenz im Modul QueryResolution. Die Aufgaben des Moduls sind es, die ConnectionController in QuerySend zu konfigurieren, die Query in das suchmaschinenspezifische Anfrageformat umzuwandeln und dem ConnectionController den Parser für die Antwort zu geben.

Es gibt für jede Suchmaschine eine Implementation der Spezifikation AbstractBuilder, welche durch die Spezifikationen URLAbstractBuilder und JSONAbstractBuilder erweitert wird.

**7.6.2.1. AbstractBuilder**

**7.6.2.2. AbstractJSONBuilder**

**7.6.2.3. AbstractURLBuilder**

**7.6.2.4. EexcessJSONBuilder**

**7.6.2.5. FarooURLBuilder**

**7.6.2.6. DuckDuckGoURLBuilder**

**7.6.2.7. EEXCESSOrigin**

### 7.6.3. QuerySend

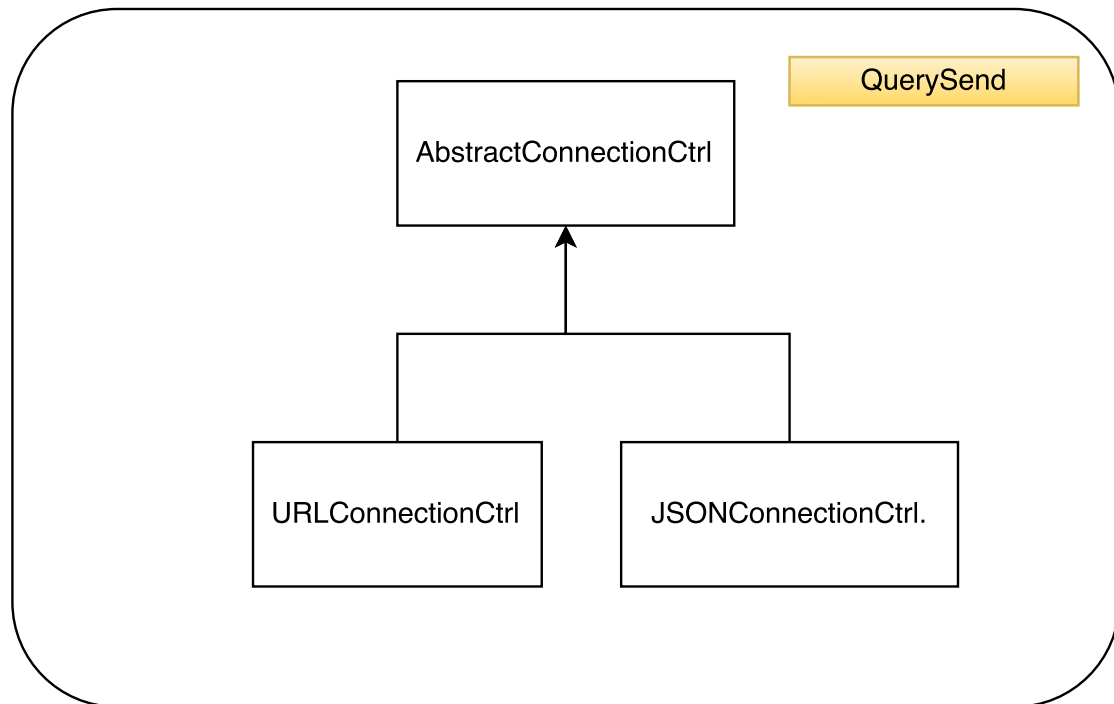


Abbildung 7.6.: Aufbau des Moduls QuerySend

Im Modul QuerySend befinden sich die ConnectionController, welche für das Versenden der Anfrage zuständig sind. Die ConnectionController werden mithilfe von QueryBuild konfiguriert und erhalten den Parser (ResponseParse).

Bevor die Antwort an den TaskController weitergereicht wird, wird diese noch durch den Parser von NSData zu SearchResult umgewandelt.

#### 7.6.3.1. AbstractConnectionCtrl

#### 7.6.3.2. JsonConnectionCtrl

#### 7.6.3.3. URLConnectionCtrl



## **7.6.4. ResponseParse**

### **7.6.4.1. FarooResponseParser**

### **7.6.4.2. DuckDuckGoResponseParser**

### **7.6.4.3. EexcessResponseParser**

### **7.6.4.4. AbstractResponseParser**

## 7.7. SearchResults

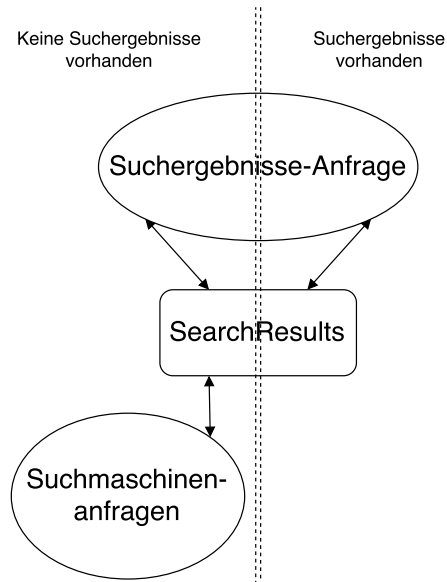


Abbildung 7.7.: Ablaufmöglichkeiten im SearchResults

SEARCHResults ist ein Container, welcher für ein Suchwort die Suchergebnisse speichert. Der Container ist in der Lage, bei fehlenden Suchergebnissen Suchanfragen abzuschicken und informiert mit dem `observe`, ob neue Suchergebnisse von der Suchmaschine gekommen sind.

## 7.8. Ranking

### 7.8.1. Rules

### 7.8.2. SearchRules

### 7.8.3. RankingPersistence

#### 7.8.3.1. PersistenceController

#### 7.8.3.2. RankingDataObject

#### 7.8.3.3. RankingDataObjectPersistency

Das Ranking dient der Sortierung der Suchergebnisse nach den Interessen des Nutzers. Das Sortieren der Zusatzinformationen wird durchgeführt, bevor die Ergebnisse dem Nutzer präsentiert werden. D.h. es wird in der Klasse „SearchResults“ initialisiert und gestartet, nachdem diese aus dem Internet heruntergeladen wurden.

Die Funktionalität des Rankings wird mit Hilfe von verschiedenen Regeln realisiert. Aktuell existieren drei Regeln: „Language“, „Mendeley“ und „MediaType“. Jede Regel besitzt einen Faktor für die Gewichtung. Dieser legt fest, wie wichtig eine Regel ist. Für jeden Datensatz werden die o.g. Regeln mit den entsprechenden Eingangsparametern erzeugt. Im weiteren Fortschritt des Projekts soll das Ranking dahingehend erweitert werden, dass nicht für jeden Datensatz alle Regeln erzeugt werden, sondern nur diejenigen, die für den jeweiligen Datensatz notwendig sind. So wird z.B. für einen Datensatz, der nicht von der Suchmaschine Mendeley kommt, auch nicht die Regel „Mendeley“ erzeugt.

Die erstellten Regeln pro Datensatz werden in einem Hilfsarray zwischengespeichert (SearchRules). Der Hauptteil des Rankings beginnt erst mit der Berechnung. Trifft eine Regel zu, d.h. der erwartete Wert der jeweiligen Regel entspricht dem tatsächlichen Wert des Suchergebnisses, so wird eine 1 von der Regel zurückgeliefert. Trifft eine Regel nicht zu, so wird eine 0 zurückgeliefert. Die Rückgabewerte werden mit Hilfe des Enums „RuleMatch“ realisiert. Dieser Wert wird mit der Gewichtung der jeweiligen Regel multipliziert. Im Anschluss werden alle Ergebnisse aller Regeln, die zu einem Datensatz gehören, aufsummiert und durch die Anzahl der verwendeten Regeln dividiert. Dieser errechnete Wert gibt prozentual wieder, in wie weit das Suchergebnis für den Nutzer interessant sein könnte. Nach dem Endergebnis werden alle Suchergebnisse zugehörig zu einem Search-Link der Größe nach sortiert und im Anschluss wieder dem Aufrufer zurückgegeben, der sie dann dem Nutzer präsentiert.

Die Klassen „PersistenceController“, „RankingDataObject“ und „RankingDataObjectPersistency“ werden bei dem aktuellen Stand des Projektes noch nicht verwendet. Im weiteren Verlauf werden die Klassen zum Speichern von nutzerspezifischen Informationen verwendet, um so die Suchergebnisse noch besser auf die Interessen des Nutzers zuschneiden zu können.

## 7.9. SettingsModel

Im SettingsModel werden die verschiedenen Einstellungen für den Browser gespeichert. Der Nutzer kann diese über das Einstellungsmenü des Geräts ändern.

**Einstellungen** Die Einstellungen sind in drei verschiedene Kategorien eingeteilt.

- Suchmaschinen
- Nutzerprofil
- Browsereinstellungen

**Suchmaschinen** Hier kann der Nutzer die verschiedenen Suchmaschinen einstellen, die für die SearchTags verwendet werden sollen. Dabei kann entweder jede Suchmaschine einzeln gewählt werden, oder den Empfehlungen des Autors gefolgt werden. Zur Auswahl stehen:

- DuckDuckGo
- Eexcess
- Faroo

Sobald eine Suchmaschine gewählt wurde, werden die Empfehlungen des Autors ignoriert.

**Nutzerprofil** Hier können die Nutzerdaten angegeben werden. Davon wird bisher nur die Sprach verwendet. Einstellungsmöglichkeiten:

- Name
- Alter
- Stadt
- Land
- Sprache

**Startseite** Hier kann die Startseite des Browser festgelegt werden. Die hier gesetzte Seite wird noch nicht im Browser als Startseite übernommen.

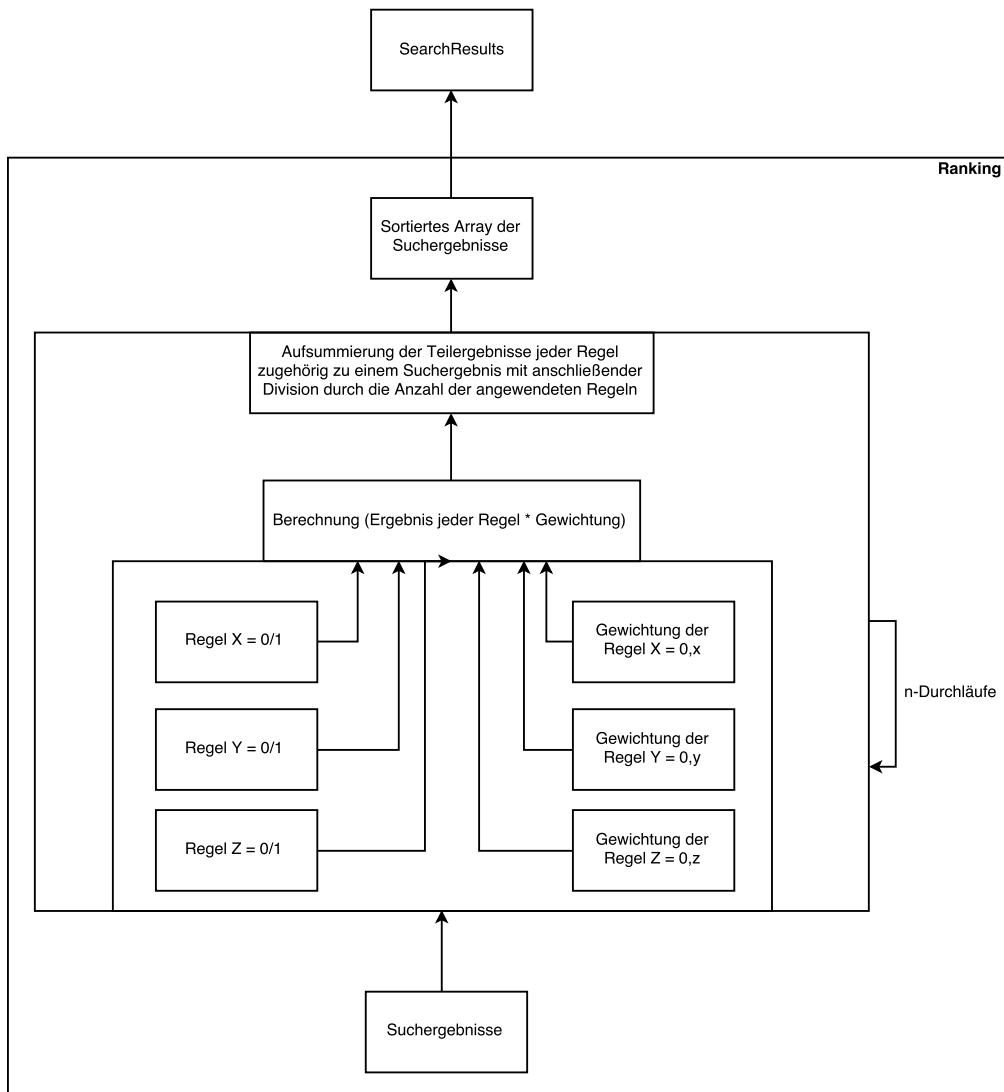


Abbildung 7.8.: Ablauf des Rankings

## Abbildungsverzeichnis

4.1. Navigationselemente . . . . .	7
4.2. Navigationselemente . . . . .	8
4.3. Lesezeichenverwaltung . . . . .	9
4.4. SEARCH-Tagverwaltung . . . . .	10
4.5. Einstellungen . . . . .	11
6.1. Klassenabhängigkeiten im UI . . . . .	14
6.2. Searchtag Anzeige . . . . .	15
7.1. Umwandeln eines WebContent-Objekts in ein SearchModels-Objekt . . . .	22
7.2. Aufbau des Moduls QueryCreation . . . . .	24
7.3. Übersicht des Moduls QueryResolution . . . . .	26
7.4. Aufbau des Moduls QueryResolution . . . . .	26
7.5. Aufbau des Moduls QueryBuild . . . . .	27
7.6. Aufbau des Moduls QuerySend . . . . .	29
7.7. Ablaufmöglichkeiten im SearchResults . . . . .	31
7.8. Ablauf des Rankings . . . . .	34

## Tabellenverzeichnis