

# HTW P2P Volltextsuche

---

None

*Boris Caspary, Emma Calewaert, Jonathan Neidel, Joscha Seelig, Leon Enzenberger, Ryan Torzynski, Simon Breiter, Stefan  
Sadewasser*

*None*

# Table of contents

---

1. Home	3
1.1 Peer 2 Peer Fulltext search	3
1.2 Basic concepts	4
1.3 Subject	5
1.4 Work division	6
2. Design decisions	7
2.1 Design decisions	7
2.2 Architecture diagram	8
2.3 Sequence diagrams about handling speeches	9
2.4 Beispiel Partition By Keyword von zentraler Volltextsuche aus	10
3. Text Processing	12
3.1 Crawler	12
3.2 Textextraction 19	15
3.3 Textextraction 18	20
4. Fulltext-search	24
4.1 P2P-DHT	24
4.2 Fulltext-search	29
5. User Interface	34
5.1 Problemstellung	34
5.2 Lösungsansatz	35
5.3 Probleme	36
5.4 Schlussbetrachtung	37
5.5 Verwendung der Software	38
6. OPS	39
6.1 Description short	39
6.2 Description long	39
6.3 Setup	40
6.4 Challenges	42
7. Outlook	43
7.1 Outlook	43

# 1. Home

---

## 1.1 Peer 2 Peer Fulltext search

---

This is a project created for the Hochschule für Technik und Wirtschaft in Berlin. The participants are:

- [Boris Caspary](#)
- [Emma Calewaert](#)
- [Jonathan Neidel](#)
- [Joscha Seelig](#)
- [Leon Enzenberger](#)
- [Ryan Torzynski](#)
- [Simon Breiter](#)
- [Stefan Sadewasser](#)

The repository for the project can be found [here](#)

## 1.2 Basic concepts

---

In this project, we've combined two basic concepts into an application: the peer-2-peer network and the fulltext search. In order to explain more about our project, a broad definition of these two concepts must be established.

On one hand, we have the peer-2-peer network. This is a network type where, in contrast to the classic client-server model, each participating node has equal rights.

On the other hand, there's the full text search. This is a type of purely syntactical search, where the goal is to find given words in a large set of documents. Often this process is divided into two phases: The indexing phase - where documents are collected, prepared and indexed - and the search phase - where requests are received, corresponding documents are gathered, ranked and displayed to the user.

## 1.3 Subject

The subject for our fulltext search are the speeches of the *Bundestag*. This set of data was chosen due to its Open Data nature combined with the size of the dataset and the fact that these protocols are recorded live into an XML formatted document.

The XML format *should* mean, that these files are easier to parse, but this was not always the case. There's a clear distinction between the 19th legislative period and the 18th and earlier periods.

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet href="dbtplenarprotokoll.css" type="text/css" charset="UTF-8"?>
<!DOCTYPE dbtplenarprotokoll SYSTEM "dbtplenarprotokoll.dtd">
<dbtplenarprotokoll vertrieb="Bundesanzeiger Verlagsgesellschaft mbH, Postfach 1 0 05 34, 50445 Köln, Telefon (02 21) 97 66 83 40, Fax (02 21) 97 66 83 44,
www.betrifft-gesetze.de" herstellung="H. Heenemann GmbH Co. KG, Buch- und Offsetdruckerei, Bessemerstraße 83-91, 12103 Berlin, www.heenemann-druck.de" sitzung-
ort="Berlin" herausgeber="Deutscher Bundestag" issn="0722-7980" wahlperiode="19" sitzung-nr="237" sitzung-datum="25.06.2021" sitzung-start-urzeit="9:00"
sitzen-ende-urzeit="18:16" sitzung-naechste-datum="07.09.2021" start-seitennr="30883">
  <vorspann>
    <kopfdaten>
      <plenarprotokoll-nummer>Plenarprotokoll <wahlperiode>19</wahlperiode></sitzen-snr>237</sitzen-snr>
    </plenarprotokoll-nummer>
    <herausgeber>Deutscher Bundestag</herausgeber>
    <berichtart>Stenografischer Bericht</berichtart>
    <sitzungstitel>
      <sitzungs-nr>237</sitzen-snr>. Sitzung</sitzen-snr>
    </sitzungstitel>
    <veranstaltungsdaten>
      <ort>Berlin</ort>, <datum date="25.06.2021">Freitag, den 25. Juni 2021</datum>
    </veranstaltungsdaten>
  </kopfdaten>
  <inhaltsverzeichnis>
    <ivz-titel>Inhalt:</ivz-titel>
    <ivz-eintrag>
      <ivz-eintrag-inhalt>Absetzung des Zusatzpunktes 21</ivz-eintrag-inhalt>
      <a href="S30883" typ="druckseitennummer">
        <seite>30883</seite>
        <seitenbereich>A</seitenbereich>
      </a>
    </ivz-eintrag>
  </inhaltsverzeichnis>
</dbtplenarprotokoll>
```

This first excerpt is an example from the 19th period. The document follows the XML structure well and provides good groundwork to allow easy parsing.

```
<?xml version="1.0" encoding="UTF-8"?>
<DOKUMENT>
  <WAHLPERIODE>18</WAHLPERIODE>
  <DOKUMENTART>PLENARPROTOKOLL</DOKUMENTART>
  <NR>18/179</NR>
  <DATUM>23.06.2016</DATUM>
  <TITEL>Plenarprotokoll vom 23.06.2016</TITEL>
  <TEXT>Plenarprotokoll 18/179

Deutscher Bundestag
Stenografischer Bericht

179. Sitzung

Berlin, Donnerstag, den 23. Juni 2016

Inhalt:

Wahl der Abgeordneten Nina Warken als or-
dentliches Mitglied des Gemeinsamen Aus-
schusses . . . . . 17575 A

Wahl des Abgeordneten Steffen Bilger als
ordentliches Mitglied des Vermittlungsaus-
schusses . . . . . 17575 B

Erweiterung und Abwicklung der Tagesord-
nung . . . . . 17575 B

Absetzung der Tagesordnungspunkte 14, 15 b
und 25 . . . . . 17576 D

Begrüßung des Botschafters der Republik
Polen, Herrn Jerzy Jozef Marganski . . . . . 17613 C
```

This second excerpt is an example from the 18th period. Without the few XML tags at the start of the document, it would be impossible to tell that this is indeed an XML file and not a plain text file. This made these documents quite a lot more difficult to parse.

For more information about these problems and their solutions, please find the documentations about these respective components.

## 1.4 Work division

---

During our work on the project, our group naturally tended towards a division of the components into working groups. We had two major groups: The text processing group (crawler, extraction 18, extraction 19) and the fulltext-search group (P2P-DHT and fulltext-search). Aside from these groups, we also had OPS and the UI, which stood more separately on their own.

## 2. Design decisions

---

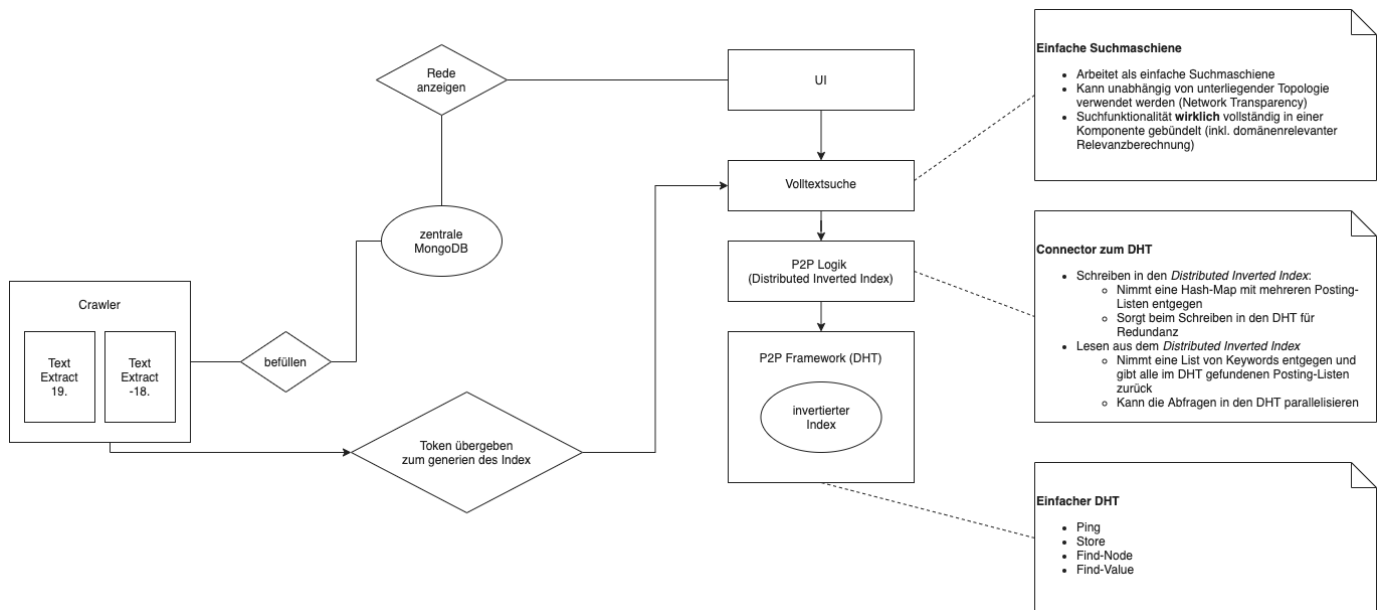
### 2.1 Design decisions

---

This section is dedicated to the major design decisions that were made within the duration of the project and the reasoning behind these decisions.

## 2.2 Architecture diagram

This diagram is an overview of the overarching architecture we used for our project.

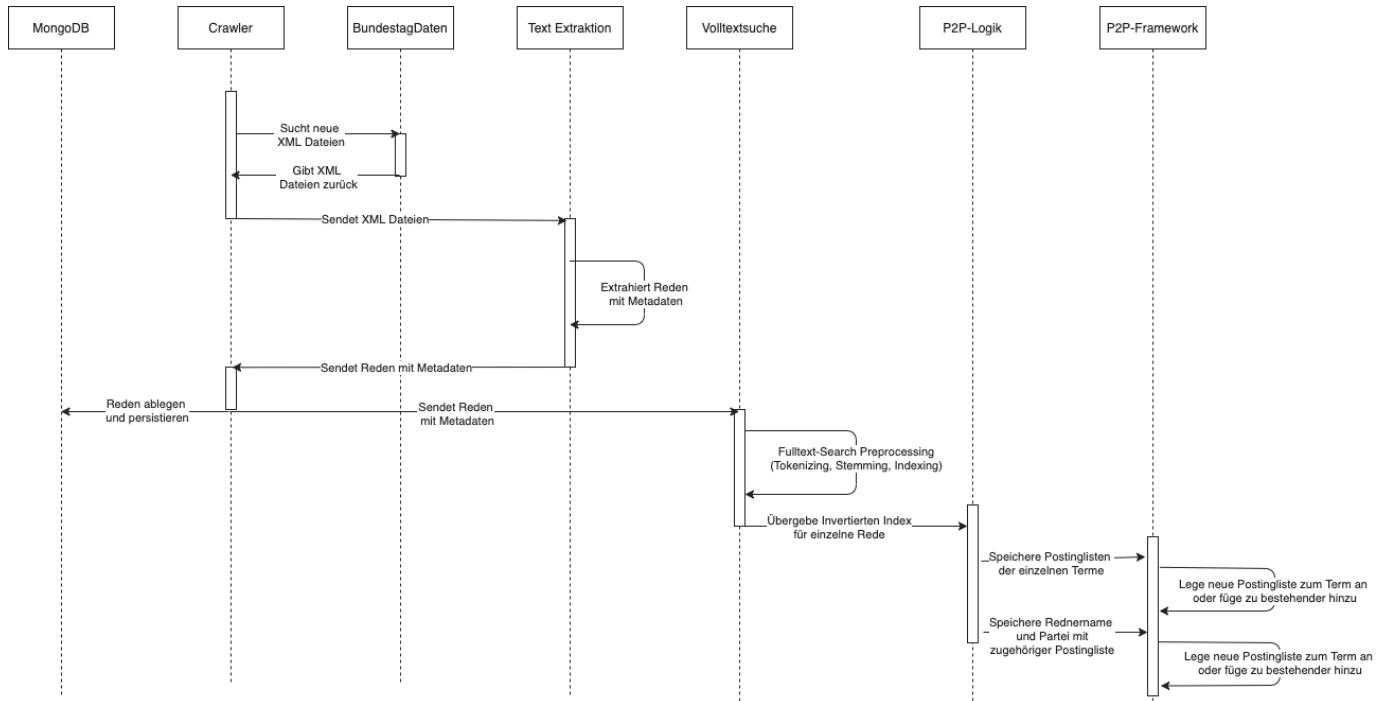




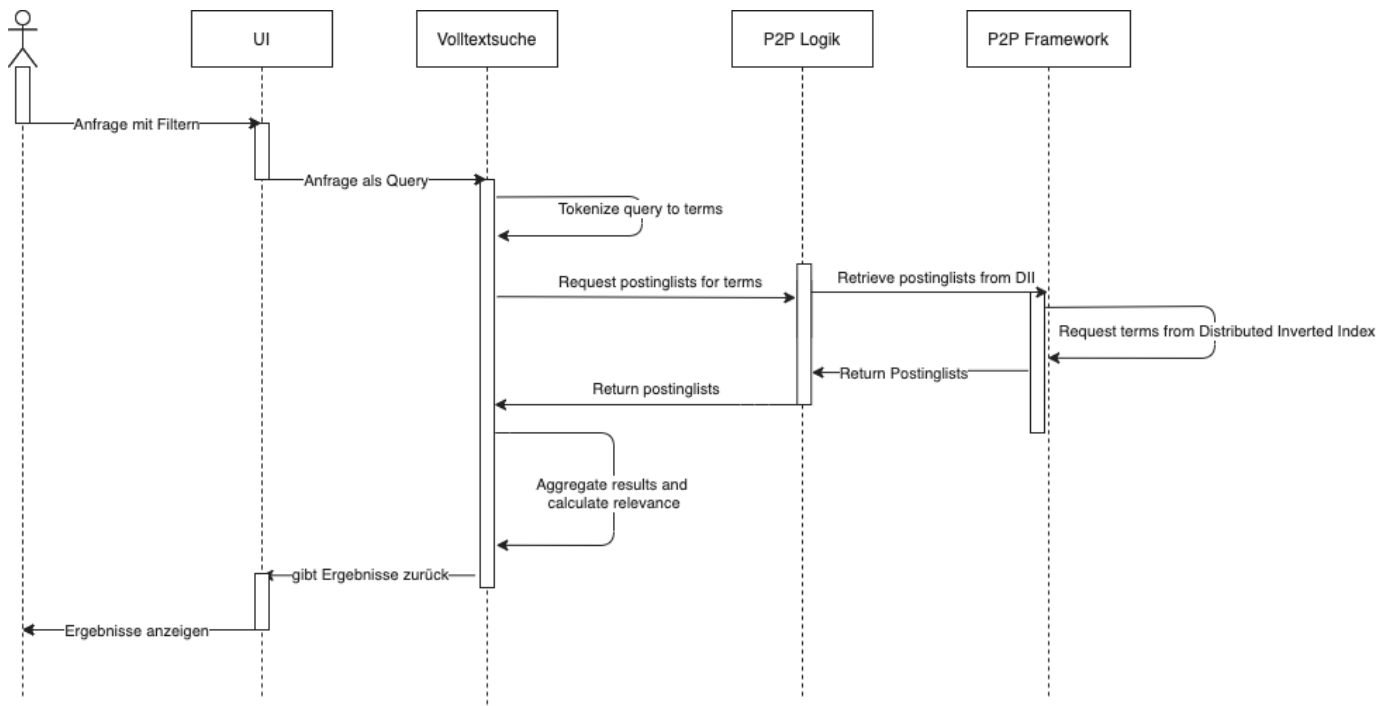
## 2.3 Sequence diagrams about handling speeches

The following two sequence diagrams describe our chosen process for indexing and retrieving speeches.

Indexing speeches:



Retrieving speeches:



## 2.4 Beispiel Partition By Keyword von zentraler Volltextsuche aus

**Drei Dokumente:** d1, d2, d3 **Vier Terme:** t1, t2, t3, t4 **Drei Peers:** p1, p2, p3

Dokumente:

```
d1 = {t1, t2, t3}
d2 = {t4}
d3 = {t1, t2}
```

*Hashen der Terme auf die Peers:* T1 -> p1 T2 -> p2 T3 -> p3 T4 -> p1

### 2.4.1 Indexing

1. Wenn ein Dokument neu indexiert werden soll, kann z.B. durch Hashing des Dokuments der verantwortliche Peer bestimmt werden, der das Dokument vorverarbeitet (tokenizing, stemming, indexing, ...)
2. Alle gefundenen Terme werden gehashed und nach zuständigen Peers zusammengefasst
3. Die Terme werden inklusive der Postinglisten als Batch (über die P2P Logik) an den jeweils zuständigen Peer gesendet
4. Der jeweils zuständige Peer fügt die Terme mit Postinglisten in seinen eigenen (lokalen) Invertierten Index hinzu oder nimmt es in die bereits vorhandene Postingliste auf, falls der Term bereits bekannt ist.

#### Beispiel:

Invertierter Index auf den einzelnen Peers:

```
P1 {
  t1 = [d1, d3]
  t4 = [d2]
}
P2 {
  t2 = [d1, d3]
}
P3 {
  t3 = [d1]
}
```

Der gesamte Index eines jeden Peers könnte nun noch auf den jeweiligen Vorgänger repliziert werden. Fällt ein Peer unerwartet aus, kann der vorherige Peer als Fallback dienen:

```
P1 {
  t1 = [d1, d3]
  t2 = [d1, d3]
  t4 = [d2]
}
P2 {
  t2 = [d1, d3]
  t3 = [d1]
}
P3 {
  t1 = [d1, d3]
  t3 = [d1]
  t4 = [d2]
}
```

Die Redundanz gleicht sich mit steigender Anzahl der Peers aus und könnte ggF. sogar auch auf den Nachfolger ausgeweitet werden.

### 2.4.2 Retrieval

1. Es wird ein Request gegen die Volltextsuche auf einem beliebigem Peer gestellt
  - Das wird in unserem Fall immer der Peer sein, von dessen UI die Suche gestellt wurde
2. Die Volltextsuche extrahiert die einzelnen Terme der Query und ruft diese über die P2P Logik aus dem *Distributed Inverted Index* ab
  - Für den Fall, dass einer der gesuchten Terme auf Peer liegt, von dem die Anfrage ausgeht, sollten Optimierungen vorgenommen werden, so dass der Zugriff einem einfachen lokalen Zugriff gleicht, der aber trotzdem über den *Distributed Inverted Index* abstrahiert wird.

3. Sobald die Volltextsuche die Postinglisten erhalten hat, kann sie diese verarbeiten, das Ranking durchführen und die sortierten Ergebnisse zurück liefern.

- Hier kann überlegt werden, ob die Postinglisten inkrementell zurück gegeben werden, so dass die Volltextsuche schon arbeiten kann, während die P2P-Logik die restlichen Postinglisten noch sucht.

#### Beispiel:

*Anfragen:* A1 = „t1 AND t3“ A2 = „t3 AND t4“ A3 = „t2“

**A1:** Anfrage landet auf beliebigem Peer, dieser greift auf den Invertierten Index zu und baut das Ergebnis zusammen. Fällt der Peer für einer der Terme aus, wird über den DHT (das P2P Framework) auf den Fallback Peer zurückgegriffen. In beiden Fällen vollständiges und zuverlässiges Ergebnis, auch wenn einer der Peers ausfällt. **A2:** Gleiches Verhalten wie bei A1. Vollständiges und zuverlässiges Ergebnis, auch wenn einer der Peers ausfällt. **A3:** Gleiches Verhalten wie bei A1. Vollständiges und zuverlässiges Ergebnis, auch wenn einer der Peers ausfällt

### 2.4.3 Indexing und Retrieval nach Parteizugehörigkeit und Rednername

Damit auch performant nach allen Reden einer Partei oder eines einzelnen Redners gesucht werden kann werden beim Indexing zwei Spezial-Tokens für Partei- und Rednername angelegt. Diese bekommen einen besonderen Prefix, damit sie von einfachen Termen unterschieden werden können. Beim indizieren einer Rede werden dann also jeweils zwei zusätzliche Schlüssel im *Distributed Inverted Index* abgelegt, die beide das entsprechende Dokument beinhalten.

#### Beispiel

d1 = {t1, t2, t3} von Angela Merkel (CDU)

#### Hashing:

```
t1 -> p1
t2 -> p2
t3 -> p3
_speaker:angela Merkel -> p1
_affiliation:cdu -> p2
```

*Der Prefix ist hier nur ein Beispiel und muss im Weiteren noch festgelegt werden*

#### Index:

```
P1 {
  t1 = [d1]
}
P2 {
  t2 = [d1]
}
P3 {
  t3 = [d1]
}
```

### 2.4.4 Argumentation der Design-Entscheidung

Nachteil gegenüber des *Partition By Document* Ansatzes ist, dass die Suche nur als gesamtes Netzwerk wirklich funktionsfähig ist. Bei *Partition By Document* kann jeder Peer eine eigenständige Volltextsuche für alle ihm zugeordneten Dokumente vornehmen.

Vorteil hingegen ist, dass das Netzwerk nicht für jede einzelne Query geflutet (Broadcast) werden muss, sondern lediglich die für die entsprechenden Keywords der Query verantwortlichen Peers angefragt werden müssen.

## 3. Text Processing

---

### 3.1 Crawler

---

#### 3.1.1 Über den Crawler

---

Der Crawler ist dafür zuständig die Bundestagswebseite in regelmäßigen Abständen auf neu veröffentlichte Reden zu überprüfen, diese herunterzuladen, durch externe Anwendungen Reden aus den Dateien zu extrahieren und diese in das P2P Netzwerk weiterzureichen.

## 3.1.2 Installation und Ausführung

---

### Ohne Docker

1. Stellen Sie sicher, dass .NET 5.0 SDK auf Ihrem System installiert ist
2. Klonen Sie das Repository
3. Bauen Sie die Anwendung mit `dotnet publish ./Crawler -c Release`
4. Wechseln Sie zum Ausgabeverzeichnis des Builds `./Crawler/bin/Release/net5.0/publish`
5. Folgen Sie dem Anleitungsschritt "Konfiguration"
6. Führen Sie die Anwendung aus `./Crawler.exe`

### Mit Docker Compose

#### BAUEN

1. Klonen Sie das Repository
2. Führen Sie folgenden Befehl im Root-Verzeichnis des Repositories aus, um das Docker-Image zu bauen: `docker build . --tag crawler`

#### AUSFÜHREN

1. Erstellen Sie eine appsettings.json mit Ihren gewünschten Einstellungswerten.
2. Führen Sie den Docker-Container aus mit `docker-run -v ./appsettings.json:/app/appsettings.json crawler`

#### MOUNTING POINTS

- Der Pfad `/app/data/` muss zum Hostsystem persistiert werden, damit der Crawler tracken kann, welche Protokolle bereits indexiert wurden.
- Die `appsettings.json` Datei muss nach `/app/appsettings.json` gemounted werden, falls die Konfiguration via .json-Datei geschehen soll. Alternativ kann dieser Schritt weggelassen werden, wenn die Konfiguration via Umgebungsvariablen geschieht

### Konfiguration

Die Konfiguration geschieht über die in der Standard-Konfigurationsdatei bereitgestellten Parameter. Über den Einstellungen sind Kommentare zur Erläuterung vorhanden. Es muss entweder die appsettings.json existieren oder es müssen alle vom Standard abweichenden Einstellungen über gleichnamige Umgebungsvariablen übergeben werden. Umgebungsvariablen überschreiben die Werte der appsettings.json.

Wenn z.B. die Einstellung `Interval` überschrieben werden soll mit einer Umgebungsvariable, dann muss die Umgebungsvariable auch `Interval` heißen.

Name	Standardwert	Beschreibung
Interval	<code>* * * * *</code>	Intervall als CRON-Expression, die bestimmt in welchem Intervall der Crawler die Seite des Bundestages überprüft.
InitialDelay	<code>0</code>	Einmalige Verzögerung des Anwendungsstarts in Sekunden.
ChunkDelay	<code>0</code>	Verzögerung zwischen den POST Anfragen an die Indexing-API in Sekunden.
MaximumBatchSize	<code>5</code>	Maximale Anzahl der Reden die in einer POST Anfrage an die Indexing-API übergeben werden.
MongoConnectionString	<code>mongodb:// 0.0.0.0:8430</code>	Verbindungsstring zur Mongo-DB Datenbank, in der der Crawler extrahierte Reden ablegen wird.
MongoDatabase	<code>crawler</code>	Name der Datenbank.
MongoCollection	<code>protocols</code>	Name der Collection.
LocalDbConnectionString	<code>Data Source=data/ local.db</code>	Verbindungsstring zur SQLite Datenbank, die der Crawler nutzt um abzuspeichern welche Dokumente bereits indexiert wurden.
IndexingApiEndpoint	<code>http://0.0.0.0:8421/ api</code>	Endpunkt des P2P Netzwerkes, an den der Crawler Reden zur Indexierung schickt.
IndexingApiTimeout	<code>300</code>	Timeout der HTTP Anfrage der vorangehenden Einstellung.

## 3.2 Textextraktion 19

---

### 3.2.1 Text-Extraktion von Bundestagsreden der 19. Legislaturperiode

---

Nimmt Plenarprotokolle der 19. Legislaturperiode oder späteren im XML-Format entgegen und gibt eine JSON-Datei mit Reden aus dem Protokoll zurück.

### 3.2.2 Titel der Reden

---

Da Bundestagsreden im Protokoll keine traditionellen Titel haben, wurde entschieden, die jeweiligen Tagesordnungspunkte als Titel zu verwenden. Da diese teilweise unverhältnismäßig lang sind, wurde eine Funktionalität zur Kürzung der Titel hinzugefügt. In manchen Fällen werden mehrere Tagesordnungspunkte in Verbindung miteinander abgearbeitet. In diesem Fall enthält der gekürzte Titel nur die gekürzte Version des erstgenannten Tagesordnungspunktes.



### 3.2.3 Nutzung

---

```
python text-extraction-19 <Plenarprotocol xml-file>
```

erzeugt eine JSON-Datei mit dem gleichen Namen im gleichen Ordner, die alle Reden des Protokolls enthält.

```
python title_shortener <speech json-file>
```

fügt gekürzte Titel zu einer bereits existierenden JSON-Datei, falls nicht bereits vorhanden.

## 3.2.4 Anforderungen

---

Python 3.9

### 3.2.5 Format der zurückgegebenen JSON-Datei

---

```
{
  //Tagesordnungspunkt
  "title": "string",
  //Gekürzte Version des Titels
  "title_short": "string",
  //Redner
  "speaker": "string",
  //Zugehörigkeit: entweder Partei oder Amt, so wie im Protokoll vorhanden
  "affiliation": "string",
  //Datum der Rede
  "date": "Date in ISO 8601-1:2019 String Format",
  //Gesamter Inhalt der Rede
  "text": "string"
}
```

## 3.3 Textextraction 18

---

### 3.3.1 Textextraction-18

---

The content-related task of the text extraction-18 was to extract all speeches of the 1st - 18th legislative period, the XML documents passed by the crawler and output them in JSON files. The crawler achieved this by temporarily persisting the plenary minutes as XML files and passing them as parameters to the text extraction-18.

The basic idea is to examine the text block semantically and syntactically in order to find a search algorithm, which will find the text within the text block:

- Title of the speech
- Name of the speaker
- Affiliation
- Date of the speech
- Speech

to a person and extracts it.

---

### 3.3.2 Problems

Most of the problems arose from the assumption that there was consistency in the formatting of the documents.

Problem	Description
Document type	In the 1st - 14th electoral legislature period, a document type was given to the XML files, but from the 15th electoral legislature onwards, this was no longer the case.
Formatting	In some documents, the considered commonalities of the table of contents were also not present, which makes a uniform programme for all electoral legislatures almost impossible.
Naming of the affiliation	<p>There were also many spelling differences in the naming of affiliations. The best example of this is the party "BÜNDNIS 90/DIE GRÜNEN". For the different spellings of this party alone:</p> <ul style="list-style-type: none"> <li>• BÜNDNIS 90/DIE GRÜNEN</li> <li>• BÜNDNIS 90/ DIE GRÜNEN</li> <li>• BÜNDNIS 90  /DIE GRÜNEN</li> <li>• BÜNDNIS 90/DIE GRÜ- NEN</li> </ul> <p>three regular printouts had to be changed to make them work.</p>

The decisive point in the extraction was the search in the table of contents for persons. Within the search itself, however, there were differences that had to be processed separately. The search was implemented with the method `createMap()` in the `SpeechSearch` class. The literal flow of the method is roughly as follows:

- **IF** there is a title
  - the first name is also contained in the same entry and must be saved  
after that, each entry is a name,  
until the entry that contains a title.  
Next loop pass from this title.
- **otherwise**
  - **if** a title is followed by a name
    - **then** each subsequent entry is a name and must be saved,  
until an entry contains a title.  
next loop pass from this title.

If the speakers were found for each title, these entries are saved in a map.

### 3.3.3 Build Jar File

---

```
mvn clean package
```

### 3.3.4 Execute

---

```
java -jar textextraction-18.jar \<plenary minutes xml file>
```

creates a json file from the plenary minutes with all entered speeches

## 4. Fulltext-search

---

### 4.1 P2P-DHT

---

#### 4.1.1 p2p-dht

---

Part of a project at HTW to build a distributed full-text-search. [See the rest of the project.](#)

This repo implements the DHT that hosts the entries of the inverted index for the [full-text-search component](#).

The [slides](#) of our closing presentation include an explanation of how this whole project fits together and takes a deeper look into what this component does if you want to know more.



## 4.1.2 Design

---

- [API Specification, API Examples](#)
- [Framework Choice and Data Distribution Design](#)

## 4.1.3 Run

---

### Install:

```
npm install
```

### Development:

```
npm start
```

### Production:

```
npm run start-prod
```

## 4.1.4 Configure

Configured via environmental variables in `variables.env`. See: [variables.env.example](#).

### HTTP\_PORT

Port for the http server to run on.

Default: `8090`

### HTTP\_LIMIT

The maximum request body size.

Default: `10mb`

See: [body-parser](#)

### PEER\_PORT

Port for the p2p node to run on.

Default: `8070`

### PEER\_IP

IP for the p2p node to run on.

Default: `127.0.0.1` (localhost)

### PEER\_LIST

List of peers, required for joining the network. Leave blank on first peer, that is "creating" the network.

Default: `` (empty)

Format: - [multiaddr](#) incl. ipfs address - seperated by commas

Example:

```
PEER_LIST=/ip4/127.0.0.1/tcp/8071/ipfs/QmdW3RF4Yq4acYc4bgUmxeuJQLb2mQpQmMuDTGir5gQcYM, /ip4/127.0.0.1/tcp/8072/ipfs/QmPP5pdu6Dh93DL7LnQkKU2x8m4BoSrQswjQR5q26PMneg
```

### PEER\_STORAGE

Storage location on disk for the data of the DHT.

Default: `/tmp/datastore`

### PEER\_REDUNDANCY

Specifies on how many different remote nodes a value should be stored.

Default: 2

### PEER\_MPLEX\_SIZE

Specify the max message size of the p2p net's [multiplexer](#).

Default: `10mb`

### DEBUG

Enable logging in the p2p network.

Set it to:

```
libp2p:dht:Q*,libp2p:dht:rpc:get-value:*
```

## 4.1.5 Scripts

---

### Multiaddress:

Print the multiaddress used for connection to the node:

```
npm run addr
```

### Debug:

Print the environmental variables that are being read in:

```
npm run debug
```

### Node testing:

Run a node for testing:

```
npm run spawn 1
```

### Generate peerId locally:

Create the `peerId.json` locally.

```
npm run local-peerid
```

## 4.2 Fulltext-search

---

### 4.2.1 Bundestag Speech Search

---

Simple search engine for querying speeches from the [Bundestag](#).

## 4.2.2 Get started

```
$ git clone https://github.com/htw-projekt-p2p-volltextsuche/fulltext-search
$ cd fulltext-search

$ sbt run
```

### CONFIGURE THE APP

The application configuration can be specified in `src/main/resources/application.conf` in [HOCON](#) notation. It's also possible to override the `application.conf` by java system properties or environment variables.

Environment variables need to be prefixed by `CONFIG_FORCE_` except if there is an `env-alias` specified for the property.

**They will be evaluated in following order** (starting from the highest priority):

1. Environment variable: `export CONFIG_FORCE_SERVER_HOST="0.0.0.0"`
  - **This actually doesn't work!** - use `env-alias` if defined
2. Java system property as argument: `sbt 'set javaOptions += "-Dserver.host="0.0.0.0"'; run'`
3. `application.conf`: `server { host = 0.0.0.0 }`

#### Configuration properties

identifier	description	env-alias	default
server.port	HTTP port of the service	HTTP_PORT	8421
server.host	Host of the service	SERVER_HOST	0.0.0.0
server.log-body	Enables server logging of response bodies	SERVER_LOG_BODY	true
index.storage	Storage policy for the inverted index	INDEX_STORAGE_POLICY	local
index.stop-words-location	File name of the stopwords resource	-	stopwords_de.txt
index.sample-speeches-location	File name of the sample speeches resource	-	sample_speeches.json
index.insert-sample-speeches	Inserts sample speeches on startup when set	-	false
index.distribution-interval	Interval for scanning and distributing the cached index in ms	INDEX_DISTRIBUTION_INTERVAL	120000
index.distribution-chunk-size	Size of each concurrently processed chunk of the cached index	INDEX_DISTRIBUTION_CHUNK_SIZE	100
index.insertion-ttl	Amount of retries for the insertion of single index entry	INDEX_INSERTION_TTL	5
search.cache-size	Size of cache for search results	SEARCH_CACHE_SIZE	5
peers.uri	Entrypoint to the P2P network	-	http://localhost:8090/
peers.log-body	Enables client logging of response bodies	PEERS_LOG_BODY	true
peers.max-wait-queue-limit	Max wait queue limit for requests to peers	PEERS_MAX_WAIT_QUEUE_LIMIT	1024

## INDEX STORAGE POLICIES

The application can be started with three different storage policies. They work as follows:

- `local`
  - Indexing and retrieval are both done locally in memory on the host machine.
- `distributed`
  - Indexing and retrieval are handled by calling the P2P network directly.
  - Using this option, the index request is blocked until all entries are distributed to the P2P network. This can possibly take a long while. Therefore, an appropriate timeout should be set on the calling machine.
- `lazy-distributed`
  - Indexing is done by first storing the index in a local cache and then distributing it on a background thread.
  - The interval for scanning the cached index can be configured with the option `index.distribution-interval`

## RUN TESTS

```
$ sbt test
```

## 4.2.3 Retrieve Search Results

[To the API-Doc's](#)

### SIMPLE QUERIES

Only searches with at least one term in the query fields are valid. All the other fields are optional.

To limit the maximum number of results `search.max_results` can be set to any positive integer.

The simplest possible search request has following form:

```
json
{
  "search": {
    "query": {
      "terms": "your query here..."
    }
  }
}
```

In the above example all the terms specified in `terms` will be combined with *AND*.

To combine the terms with different boolean operators the search you can extend the search with arbitrary additional terms.

```
json
{
  "search": {
    "query": {
      "terms": "find this ...",
      "additions": [
        {
          "connector": "or",
          "terms": "... or that ..."
        },
        {
          "connector": "and_not",
          "terms": "... but not that"
        }
      ]
    }
  }
}
```

### Evaluation order of boolean operators

1. First all `terms` fields are evaluated with *AND* in isolation.
2. The results will then be combined by the specified `connector` and evaluated in the following order:
  - *AND\_NOT*
  - *AND*
  - *OR*

### FILTERED QUERIES

Up until now the queries can be filtered by *speaker* or *affiliation*.

If several filters with same criteria are specified they're combined by *OR*, while the entire set resulting from all filters of same type will be combined by *AND* with the actually specified query results.



```
json
{
  "search": {
    "query": {
      "terms": "some search"
    },
    "filter": [
      {
        "criteria": "affiliation",
        "value": "SPD"
      },
      {
        "criteria": "affiliation",
        "value": "Die Linke"
      },
      {
        "criteria": "speaker",
        "value": "Peter Lustig"
      }
    ]
  }
}
```

## 5. User Interface

---

### 5.1 Problemstellung

---

Die UI soll die Eingabe von präzisen Suchanfragen zu den Bundestagsreden ermöglichen, die Suchergebnisse mit ihren Metadaten anzeigen und beim Klick auf eine einzelne Rede den vollen Text anzeigen. Hierbei gab es keine Vorgaben zur technischen Umsetzung.

## 5.2 Lösungsansatz

---

Die Benutzeroberfläche wird als React-Web-Applikation mit Javascript umgesetzt. React basiert darauf, die Website in Komponenten zu zerlegen und diese gegebenenfalls wiederverwendbar zu machen. Zusätzlich wurde CSS für das Styling der Komponenten benutzt. Ein dynamisches, erweiterbares Formular soll möglichst genaue Suchanfragen ermöglichen. Ursprünglich sollten neue Formularreihen hinzugefügt werden bei denen man einen logischen Operator sowie Typ (Freie Suche, Partei, Redner) frei auswählen kann. Aus diesem Formular wird eine Query erstellt, die dann an die Volltextsuche geschickt wird. Die Volltextsuche gibt unter anderem Dokumenten-Ids zurück anhand derer aus der MongoDB die Reden mit Metadaten angefragt werden können. Die Anzeige der Ergebnisse wird als Ergebnisliste umgesetzt, bei Klick auf einen Listeneintrag wird die volle Rede angezeigt.

## 5.3 Probleme

---

Das Design des Suchformulars musste abgeändert werden, da aufgrund des Aufbaus nicht genau klar war, in welcher Reihenfolge beziehungsweise mit welcher Priorität einzelne Formularreihen ausgewertet werden. Infolgedessen wurde das Suchformular in drei Hauptbereiche eingeteilt: Freie Suche sowie Partei und Redner, die letzteren beiden bieten jedoch keine Auswahl eines logischen Operators mehr an.

Im Laufe des Projekts wurde klar, dass außerdem ein Backend für den Zugriff auf die MongoDB nötig ist, da ein direkter Zugriff aus dem UI nicht möglich ist. Es wurde als simples Node.js Projekt aufgesetzt, das eine GET-Request mit Dokumenten-Id als Übergabeparameter beantworten kann.

Anfangs war eine Sortierung der Suchergebnisse in Diskussion, diese wurde aber bewusst weggelassen, da die Suchergebnisse standardmäßig nach Relevanz sortiert von der Volltextsuche zurückgegeben werden. Eine alphabetisches Sortieren nach Partei oder Titel der Reden erschien obsolet. Durch die Komplexität der Titel wird bei einer alphabetischen Sortierung kein Mehrwert geboten.

## 5.4 Schlussbetrachtung

---

Die Anforderungen wurden weitestgehend erfüllt. Aus Zeitmangel wurden einige Features nicht mehr angeboten, wie zum Beispiel Paginierung und das Anzeigen von sinnvollen Textausschnitten in der Suchergebnisliste.

## 5.5 Verwendung der Software

---

Für volle Funktionalität muss das User Interface über das [Ops](#) gestartet werden.

Für die Nutzung des User Interface benötigt man [Node.js](#). Das User Interface ist intern in Frontend und Backend aufgeteilt, diese müssen wenn man sie lokal und alleinstehend ausführen will separat gestartet werden. Das Frontend wurde mit [Create-React-App](#) initialisiert und lässt sich z.B. mit dem Befehl `npm start` im "frontend" Ordner starten.

Das Backend ist eine simple Node.js Anwendung das sich mit dem Befehl `node index.js` im "backend" Ordner starten lässt.

Es stellt einen Endpunkt für einen GET-Request bereit, dieser kann folgendermaßen `http://{hostadress}:{hostport}/api/protocol/doc_id` erreicht werden und gibt das Protokoll mit der passenden doc\_id zurück, falls dieses gefunden wurde.

## 6. OPS

---

### 6.1 Description short

---

The [OPS repository](#) contains all configuration files and scripts used to deploy peers for the htw-projekt-p2p-volltextsuche.

### 6.2 Description long

---

The peer to peer network consists of nodes with equal functionality. The nodes all have an user-interface delivered via http, a service for searching for terms and p2p-distribution system for managing the inverted index of the search service. From here on such a node is called a "peer".

There is a unique node which additionally to the functionality mentioned above, also hosts a central db holding all the speeches in json format as well as a crawler filling it with speeches. From here on this node is called the "data".

## 6.3 Setup

---

In order to setup a network of peers you need to setup one "data" node as well as as many peers you desire. If you don't need the scaling capability, you can just setup one data node as well.

### 6.3.1 Development

---

#### 1. Setup your firewall to allow communication through these ports:

- 22 (ssh)
- 80 (http)
- 8080 (alternate http port)
- 8421 (fulltext-search)
- 8090, 8070 (p2p-network)
- 8430 (mongodb)
- 8081 (ui-backend)

#### 2. Install docker: <https://docs.docker.com/get-docker/>

#### 3. Install docker-compose: <https://docs.docker.com/compose/install/>

#### 4. clone this repository

#### 5. go into ops/data/

#### 6. create a .env file with the following content

```
MONGO_INITDB_ROOT_USERNAME=<[choose one]>
MONGO_INITDB_ROOT_PASSWORD=<[choose one]>
DATA_HOST=<[the ip of your pc]>
INDEX_STORAGE_POLICY=local
UI_PORT=80
```

#### 8. start the node

- you can start it with `docker-compose up` to see the complete log in the terminal.
- you can start it with `docker-compose up -d` if you want to run it as a daemon.

#### 9. Stop the nodes again with `docker-compose down` if you are done.

### 6.3.2 Production

---

The first four steps are documented for transparency reasons. You can skip them by cloning this repository and executing `install.sh`. The same goes for a testing-environment on which you don't need to setup the firewall and may already have installed docker (possible with root permissions).

*Setup without scripts.*

#### 1. Setup your firewall to allow communication through only these ports:

- 22 (ssh)
- 80 (http)
- 8080 (alternate http port)



- 8421 (fulltext-search)
- 8090, 8070 (p2p-network)
- 8430 (mongodb)
- 8081 (ui-backend)

Route port 80 to port 8080 to enable rootless access of the docker-container port

**2. install docker rootless:** <https://docs.docker.com/engine/security/rootless/>

**3. install docker-compose:** <https://docs.docker.com/compose/install/>

**4. clone this repository**

**5. go into the folder representing this repository**

**6. create a .env file in the folder representing the type of node you want to deploy (peer or data) with the following values**

```
MONGO_INITDB_ROOT_USERNAME=<[for data choose one/for peer use the name of data]>
MONGO_INITDB_ROOT_PASSWORD=<[for data choose one/for peer use the password of data]>
DATA_HOST=<[the ip of the server running the data node]>
INDEX_STORAGE_POLICY=<[local, distributed or lazy-distributed]>
PEER_LIST=<[see step 10 or leave blank for a single node setup]>
UI_PORT=8080
```

[Read more about the index storage policies](#)

**7. go into the folder representing the type of node you want to deploy (peer or data)**

**8. start the nodes**

- you can start them with `docker-compose up` to see the complete log in the terminal.
- you can start them with `docker-compose up -d` if you want to run it as a daemon.

**9. Stop the nodes again with `docker-compose down` after seeing the ascii-art "https"-logo.**

**10. Setup the PEER\_LIST (after running all instances at least once)(only necessary for multi-node setups)**

- copy the output of all the p2pframework containers signaling the "own multiaddr:" into the "PEER\_LIST" var of every node. They should only be separated by a comma. [See](#)
- replace the "0.0.0.0"-part of the adresse with the corresponding ip of the node

**11. Start the nodes again with `docker-compose up`.**

## 6.4 Challenges

---

This project was originally intended to run with kubernetes. Due to limited server resources we changed course to use docker compose instead. Kubernetes was more likely to consume storage by writing to logs that aren't directly associated with kubernetes and are not removed when kubernetes data is cleared. This logging issue was first discovered after the switch to a docker compose based configuration. Due to the late integration phase of the project, there wasn't enough time to react adequately to this requirements change. This led to a time-consuming administration job because instances of services first need to be stopped, their images deleted and then new pulled instead of just pulling and replacing a running service. Furthermore logs needed to be limited to a small size which made debugging an interactive job because missed error logs were overwritten not even a minute later.

## 7. Outlook

---

### 7.1 Outlook

---

This section is dedicated to known imperfections of the project and possible future improvements.