Indexierung und erweiterte Suche für visualisierte TEI-Dokumente in Drupal

Diplomarbeit
von
Dmitrij Funkner

Universität Kassel
Fachbereich 16, Informatik

Erstgutachter: Prof. Dr. Lutz Wegner
Zweitgutachter: Prof. Dr. Albert Zündorf

Kassel, den 06.10.2010

Erklärung

Ich versichere hiermit, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Kassel, den eigenhändige Unterschrift

_____

Vorwort

Das Thema dieser Diplomarbeit ergab sich aus einer Projektarbeit, die in Zusammenarbeit des Fachbereichs 2 sowie des Fachbereichs 16 der Universität Kassel erwachsen ist. Es handelt sich bei dem Projekt um eine Online-Edition des Werkes *Essai sur le récit, ou Entretiens sur la manière de raconter* von François-Joseph Bérardier de Bataut. Für die Visualisierung des in *TEI-Lite* kodierten Buchs war es erforderlich, ein *XSL-Stylesheet* zu konzipieren. Zur Veröffentlichung der so generierten Dokumente dient das Content-Management-System Drupal.

Nach dem erfolgreichen Abschluss der Projektarbeit ergaben sich weiterführende Aufgabenstellungen, die sich vor allem auf die vereinfachte Bereitstellung und die Durchsuchbarkeit der gegebenen Dokumente in Drupal konzentrierten. Die hier vorliegende Diplomarbeit beschäftigt sich mit der erweiterten Suche innerhalb der Dokumente.

Mein Dank gilt Dr. des. Christof Schöch, wissenschaftlicher Mitarbeiter des Fachbereichs 2, der das Editionsprojekt ins Leben rief und die Anfrage an Prof. Dr. Lutz Wegner richtete, die Visualisierung der *TEI-Lite* kodierten Dokumente umzusetzen.

Prof. Dr. Lutz Wegner regte diese Projektarbeit an, begleitete sie stets mit großem Engagement, gutem Rat und konstruktiver Kritik. Hierfür danke ich ihm sehr.

Besonderer Dank gebührt Dipl.-Inform. Dipl.-Math. Sebastian Pape, der bei allerlei Schwierigkeiten immer ein offenes Ohr hatte, keine Mühen scheute und mir mit konstruktiver Kritik zur Seite stand.

Ein weiterer Dank gilt Prof. Dr. Albert Zündorf, der sich bereit erklärte, die Rolle des Zweitgutachters zu übernehmen.

Auch danke ich Roman Kominek, dessen erarbeitetes Modul der vereinfachten Bereitstellung von *TEI-Lite* kodierten Dokumenten dient. Da das in dieser Diplomarbeit erarbeitete Suchmodul eine Abhängigkeit zum Modul von Roman Kominek aufweist, waren Absprachen nötig, die stets reibungslos verliefen.

Kassel, im Oktober 2010 Dmitrij Funkner

Inhaltsverzeichnis

1 Einleitung

1.1 Vorarbeit – Das XSL-Projekt

Dr. des. Christoph Schöch, Mitarbeiter am Lehrstuhl für Romanistik an der Universität Kassel, initiierte im Jahre 2008 ein Editionsprojekt des 1776 erschienen Werkes *Essai sur le récit, ou Entretiens sur la manière de raconter,* geschrieben von François-Joseph Bérardier de Bataut [8].

Im Rahmen des Projekts wird angestrebt, *„eine frei zugängliche, kommentierte Ausgabe“* des Werks in einer *„auf Drupal basierende(n) Publikationsplattform“* bereitzustellen [14].

In Zusammenarbeit mit dem Fachbereich Informatik, der durch die Programmierung und Visualisierung des Werks an dem Editionsprojekt beteiligt ist, wurde im Wintersemester 2009/10 ein Projekt, unter der Betreuung von Prof. Dr. Lutz Wegner und Dipl.-Inform. Dipl.-Math. Sebastian Pape begonnen, welches zum Ziel hatte, ein XSL-Stylesheet zu konzipieren. Das aus dem Projekt hervorgegangene Stylesheet dient der Visualisierung *TEI-Lite* kodierter Texte. So werden beispielsweise Textpassagen die zu einem Vers gehören kursiv angezeigt oder Wörter die in der Originalfassung und der überarbeiteten Version nicht identisch sind, hervorgehoben. Das Stylesheet wurde mit Hilfe eines zusätzlich installierten *XML Content Moduls* in das Content-Management-System (CMS) Drupal eingebunden. Das *XML Content Modul* übernimmt die Umwandlung eines XML-Dokuments in HTML und stellt dieses nach der Transformation für Drupal zur Verfügung.

Des Weiteren benötigt das Stylesheet ein zusätzliches Javascript, das das direkte Anspringen von Seiten, das Aufklappen von Notizen sowie das Wechseln von verschiedenen Textversionen ermöglicht.

Das XSL-Stylesheet wurde auf ein bestimmtes Farbschema abgestimmt, weshalb der Benutzer keine Möglichkeit hat, das Farbschema des Stylesheets zu verändern, ohne in dessen Quellcode Änderungen vorzunehmen. Die Nutzung des Stylesheets setzt somit Programmierkenntnisse sowie grundlegende Kenntnisse im Umgang mit Drupal voraus, was für den Benutzer, je nach dessen Kenntnisstand, ein Ärgernis oder gar eine unüberwindbare Hürde darstellen kann.

Aus dieser Problematik resultierte nach der abgeschlossenen Projektarbeit zunächst, einen Lösungsansatz zu finden, der die vereinfachten Einbindung des Stylesheets und des Javascripts in Drupal ermöglicht. Dieser Aufgabe widmet sich der Kommilitone Roman Kominek.

Drupal bietet weiterhin ein Suchmodul an, das jedoch aufgrund verschiedener weiter unten aufgeführter Eigenschaften ungeeignet für die Nutzung bezüglich des Editions-Projekts ist. Die Indexierung und erweiterte Suche für visualisierte TEI-Dokumente in Drupal stellt somit ein weiteres zu erarbeitendes Themenfeld dar, mit dem sich diese Diplomarbeit auseinandersetzt.

1.2 Aufgabenstellung

Der Wunsch nach einer angepassten Suche für TEI standardisierte Dokumente beruht auf der Tatsache, dass das in Drupal enthaltene Such-Modul für diesen Zweck schlicht ungeeignet ist. Weder bietet es die Möglichkeit die Suche auf dokumentenspezifische Eigenschaften abzustimmen, noch sind die angezeigten Treffer sinnvoll verlinkt. Die Links, die auf der Ergebnisseite der Drupal-Suche präsentiert werden, stellen Verlinkungen zu einer Node[1] dar, jedoch nicht zur Textstelle auf der sich eine gefundene Zeichenkette befindet. Bei umfangreichen Texten könnte das manuelle Auffinden der Zeichenkette mühsam sein.

Um eine Suche zu ermöglichen die die beschriebenen Mängel vermeidet, ist ein Index nötig der die Eigenschaft bietet, die Suche gemäß der intendierten Suchkriterien anzupassen. Der neu zu konzipierende Index muss dabei in der Lage sein, dokumentenspezifische Eigenschaften die im TEI-Standard kodiert vorliegen, zu berücksichtigen.

Des Weiteren ist es notwendig ein Suchformular zu implementieren. Um die Suchtreffer dem Benutzer zu präsentieren muss eine zusätzliche Seite bereitgestellt werden, auf der die jeweiligen Suchergebnisse angezeigt werden. Da die Suchtreffer möglicherweise nicht die gewünschten Ergebnisse repräsentieren können, muss wiederum ein weiteres Suchformular auf der Ergebnisseite zur Verfügung stehen. Die Suche sollte zudem, nach Möglichkeit, durch weitere Optionen einschränkbar sein, eine anschauliche Präsentation bieten und im Gegensatz zur Standard-Suche, für TEI kodierte Dokumente brauchbare Verlinkungen erzeugen.

Zu der Erzeugung der Links ist bereits im Verlauf der Projektarbeit ein Verfahren[2] erarbeitet worden. Dieses Verfahren ermöglicht einen Sprung zu einem Ankerpunkt, der durch eine Buchseite eines Kapitels gekennzeichnet ist. Dieses bereits eingesetzte Verfahren kann bei dem Indexiervorgang ebenfalls genutzt werden.

Diese Ausarbeitung setzt sich angesichts der beschriebenen Aspekte und Probleme der Drupal-Standard-Suche zum Ziel, den geschilderten Lösungsansatz mit der Implementierung eines Moduls praktisch umzusetzen. Der modulare Aufbau von Drupal bietet zum Zweck der Entwicklung einer verbesserten Suche eine hervorragende Möglichkeit.

1.3 Aufbau der Diplomarbeit

Der Aufbau dieser Ausarbeitung ist in fünf Kapitel unterteilt. Zu Beginn der Ausarbeitung werden die grundlegenden Aspekte von *TEI-Lite* und Drupal näher beleuchtet. Vorerst werden die Eigenschaften der im *TEI-Lite* Format vorliegenden Dokumente erläutert. Im nächsten Schritt wird näher auf das Content-Management-System Drupal eingegangen. Dies dient dem Zweck, grundlegende Informationen zur Bewältigung der Aufgabenstellung bereitzustellen. Im Anschluss wird der Ansatz zur Lösung der gesamten Aufgabenstellung vorgestellt.

Das dritte Kapitel ist in der Art eines Benutzerhandbuchs konzipiert. Der Umgang und die Funktionen die das Modul nach abgeschlossenen Programmierarbeiten bietet, werden hier geschildert.

---

1 Nähere Informationen zu Nodes sind in Kap. *2.2* aufgeführt.

2 Ein Javascript, das bereits dieses Verfahren nutzt.

Dem folgt ein Kapitel zur Implementierung, in welchem die praktische Umsetzung der Aufgabenstellung beschrieben wird. Dieses Kapitel hat eher technischen Charakter. Es werden unter Zuhilfenahme von Quelltext-Auszügen Details zu einzelnen Entwicklungsstadien präsentiert.[3]

Eine Zusammenfassung der Ergebnisse sowie ein Ausblick auf etwaige weitere Möglichkeiten des Modulausbaus runden die Ausarbeitung ab.

3 Für weiterführende Informationen siehe Anhang A – Systemspezifikationen.

2 Grundlagen und Konzept

2.1 TEI-Standard

Im Bereich der digitalen Textverarbeitung, insbesondere für wissenschaftliche Zwecke, hat sich ein standardisiertes Dokumentenformat durchgesetzt. Es handelt sich um das so genannte TEI-Standard, das im Jahre 1987 von der Text Encoding Initiative (TEI), einer vornehmlich aus Philologen bestehenden Organisation, eingeführt wurde. Uwe M. Borghoff, Peter Rödig, Jan Scheffczyk und Lothar Schmitz beschreiben die Text Encoding Initiative wie folgt:

> *„TEI has defined an international interdisciplinary standard for encoding any kind of prose or linguistic text in a common format. […] TEI has also established guidelines that support the whole process of electronic publishing, i.e., analyzing texts, digitalizing texts, markup, and publishing.“*[1, S. 154]

Dieses Format basiert auf XML und ist in einer Metasprache definiert, die zur Zeit die Versionsnummer fünf trägt [9].

Mit Hilfe des TEI-Standards lassen sich Texte einheitlich und unabhängig des Betriebssystems kodieren. Daraus ergeben sich zahlreiche Vorteile bezüglich der Darstellung solcher Dokumente. Das Dokumentenformat bietet, da es auf XML basiert und den Ursprung in SGML (Standard Generalized Markup Language) hat, weitere Vorzüge. So liegen beispielsweise in TEI kodierten Dokumenten strukturierte Daten vor, die bestimmten Konventionen unterliegen.

Demnach stehen zur Bearbeitung der Aufgabenstellung Dokumente bereit, die anhand des TEI-Standards kodiert sind. Während der Projektarbeitsphase wurde bereits mit solchen Dokumenten gearbeitet. Diese wurden unter Berücksichtigung des *XSL-Stylesheets*, das einige der TEI-Tags verarbeitet, mittels des *XML Content Moduls* für Drupal in HTML transformiert.

Für die Indexierung des zu konzipierenden Moduls gilt es, beim Erfassen einzelner Wörter die TEI-Tags zu berücksichtigen, von denen die Wörter umschlossen werden. Einem auf diese Art erzeugten Index ist die Zugehörigkeit eines Wortes zu einem Tag bekannt. Dies ermöglicht eine Suchanfrage auf bestimmte Tags einzuschränken.

2.2 Das Content-Management-System Drupal

Unter dem Begriff Drupal verbirgt sich die Bezeichnung eines Content-Management-Systems (CMS) und Frameworks, das vom belgischen Informatiker Dries Buytaert konzipiert wurde. Es ist in der Programmiersprache PHP geschrieben und steht unter der GNU General Public License.

Die aktuelle stabile Version von Drupal ist zur Zeit 6.19 (8. September 2010). Das Modul wird jedoch für die Version 7 entwickelt, die zum gegebenen Zeitpunkt in der 7.0-alpha6 vorliegt. Der Grund hierfür besteht in diversen Verbesserungen, die diese Version

im Gegensatz zu älteren Versionen aufweist. Zudem ist nicht absehbar, ob das Modul nach der Fertigstellung weiter gepflegt und ausgebaut wird. Es soll daher sichergestellt werden, dass das Modul in der neuen, verbesserten Version bereitsteht, da innerhalb dieser Version Änderungen an der *Drupal-API* vorgenommen worden sind. Somit ist die Abwärtskompatibilität des hier entwickelten Moduls nicht gegeben.

Drupal bietet die Möglichkeit, Inhalte im Internet zu veröffentlichen und diese zu bearbeiten. Der Aufbau von Drupal ist modular gestaltet. Die Zusammensetzung ergibt sich aus dem Systemkern, dem so genannten *Core* und den Modulen, die zusätzliche Funktionalitäten bieten.

Durch die Vielzahl an Modulen, die für Drupal existieren, ist es mit diesem CMS möglich, die Fähigkeiten und Einsatzmöglichkeiten des Systems auf individuelle Ansprüche anzupassen und zu erweitern.

Mit Programmierkenntnissen ist es jedem frei überlassen, unter Berücksichtigung der GNU General Public License, die eigenen Wünsche an Funktionalitäten und Erweiterungen zu entwickeln und diese wiederum anderen Nutzern zur Verfügung zu stellen. Aufgrund des modularen Aufbaus ist es ratsam an diesem Grundgedanken festzuhalten und derartige Erweiterungen als Module umzusetzen.

Des Weiteren besitzt Drupal eine Datenbankabstraktionsschicht. Das Besondere daran ist, dass sich der Entwickler nicht darum kümmern muss, welches Datenbanksystem zum Einsatz kommt. Damit erübrigt sich eine Anpassung durch separate Datenbankabfragen für unterschiedliche Datenbanksysteme, da dies durch die Abstraktionsschicht von Drupal übernommen wird [3, S. 129f].

Mit Drupal in Verbindung stehende Begrifflichkeiten sind *Modul*, *Hook, Filter*, *Node, Block* und *cron* welche im folgenden näher erläutert werden. Die Einschränkung auf speziell diese Termini resultiert aus der praktischen Verwendung innerhalb dieser Ausarbeitung.

- **Module:** Da es sich bei Drupal um ein modulares Framework handelt, sind die jeweiligen Funktionen in Modulen enthalten. Diese lassen sich beliebig aktivieren beziehungsweise deaktivieren, was eine große Flexibilität gewährleistet [3, S. 36].

- **Hook:** Der Begriff *Hook* leitet sich aus dem Englischen ab und steht für Haken oder auch Greifer. In diesem Zusammenhang kann die Bedeutung von Hook wort wörtlich als Einhaken oder Einklinken in das Drupal-System aufgefasst werden. Somit besteht mittels Drupal-Hooks die Möglichkeit mit dem Drupal-System zu interagieren. Genauer betrachtet ist ein Hook eine PHP-Funktion, die einen Aufbau wie beispielsweise `foo_bar()` aufweist, wobei `foo` den Modulnamen, der diese Funktion implementiert, darstellt und `bar` für den Namen des Hooks steht [19].

- **Filter:** Drupal stellt mit dem Filter-Modul, das einen Teil des Drupal-Kernsystems darstellt, Funktionalitäten zum Filtern von Textformaten bereit. Das Modul bietet diverse Möglichkeiten, in Drupal veröffentlichte Inhalte zu filtern. Als Beispiel sei hier der Filter angegeben, der für die Zeichenkette `<br>`, das einen Zeilenumbruch erzeugt, eine Ersetzung durch das HTML-Tag Paragraph `<p>` vornimmt. Filter können jeweils im administrativen Bereich Drupals für Textformate gesetzt werden. Darüber hinaus können Module, zusätzlich zum in Drupal enthaltenen Filter-Modul, weitere Filter anbieten. Für Textformate können mehrere solcher

Filter aktiviert werden. Ab der Drupal-Version 7 besteht zusätzlich die Möglichkeit, Gewichtungen für die Reihenfolge festzulegen, in der die Filter zum Einsatz kommen [20].

- **Node:** Das Wort *Node* leitet sich aus dem Englischen ab und bedeutet soviel wie Knoten oder Knotenpunkt. In Drupal werden alle Inhalte von Nodes verwaltet, somit bilden sie das Kernkonzept Drupals. Ein Node an sich ist ein strukturierter Inhaltstyp und kann beispielsweise eine Buchseite, eine Story oder eine Umfrage darstellen [4, S.68].

- **Block:** Als Blöcke werden die sichtbaren Boxen der Randfenster Drupals bezeichnet, auch *Sidebar* genannt. Diese Blöcke werden von verschiedenen Drupal-Modulen generiert, sind jedoch gleichsam auch selbständig kreierbar. Beispiele für Blöcke sind: Log-in Fenster, Navigation [21].

- **cron:** Der cron-Deamon ist eine Jobsteuerung, die auf Unix basierten Systemen wiederkehrende Aufgaben zu bestimmten Zeiten ausführt [40]. In Drupal besteht die Möglichkeit, bestimmte Aufgaben durch die Implementierung von hook_cron() – beispielsweise datenbankspezifische Aufgaben – auf cron-Läufe zu verlagern.

Mit den genannten Eigenschaften des Content-Management-Systems Drupal, lässt sich die Implementierung und Einbindung der Suchfunktionalität für TEI standardisierte Dokumente in Drupal als Modul realisieren.

2.3 Bereits existierende Drupal-Such-Module

Für Drupal steht eine große Anzahl von Modulen, auch Add-Ons oder Erweiterungen genannt, bereit, mit denen die Core-Funktionen weiter ausgebaut werden können. Unter diesen sind zahlreiche Module zu finden, die entweder der Erweiterung der Drupal-Suche dienen oder unabhängig von der Standard-Suche umgesetzt sind.

Um das Rad nicht neu erfinden zu müssen, wurde zunächst das *Drupal-Search-Modul* sowie einige andere zur Verfügung stehenden Module näher betrachtet. Für das Vorhaben erwies sich das *Drupal-Search-Modul* als Grundkonzept am geeignetsten. Viele der anderen Module zielen einerseits nur darauf ab, die Drupal-Suche zu verbessern beziehungsweise zu modifizieren. Andererseits verfolgen einige Module komplett andere Ansätze, deren Konzepte wiederum ungeeignet für das Umsetzen der hier zu bearbeitenden Aufgabenstellung sind.

Eine Suche unter Einbindung der *Google-Suche* wurde ausgeschlossen, da solch ein Modul bereits existiert und dieses eine Abhängigkeit zum *Drupal-Search-Modul* aufweist. Auch steht es nicht für die Drupal 7 Version bereit. Der alleinige Einsatz der *Google-Suche* behebt zudem das bereits geschilderte Problem des Sprungs zur gewünschten Textstelle nicht. Des Weiteren würde eine Abhängigkeit zu der *Google-Suche* aufgebaut, was zum einen keine Ausfallsicherheit und zum anderen keinen Datenschutz garantiert. Somit stand fest, dass ein Modul implementiert werden muss, das vom Gesamtkonzept ähnlich der Standard-Suche ist, jedoch beim Indexieren und den Suchanfragen die Option bietet,

dokumentenspezifische Eigenschaften zu berücksichtigen.

2.4 Konzept

Nachdem grundlegende Eigenschaften und Funktionen von *TEI-Lite* Dokumenten sowie des Content-Management-Systems Drupal analysiert wurden, kann an dieser Stelle ein Konzept zur Abarbeitung der Aufgabenstellung vorgestellt werden.

Einige Kenntnisse im Umgang mit Drupal wurden bereits in der Projektarbeit erworben. Zusätzlich wurde das bereits existierende *Drupal-Search-Modul* analysiert. Dazu diente sowohl die *Drupal-API* Referenzseite [17], als auch Tutorials und Blogbeiträge und ähnliche Seiten, die sich mit dieser Thematik beschäftigen (vgl. [15,16]). Daraus ergab sich ein Konzept des zu implementierenden Moduls, das folgende Kernpunkte beinhaltet:

Abhängigkeit des Moduls

- Eine Abhängigkeit des Moduls muss zu dem Modul aufgebaut werden, das die dafür nötigen Dokumentenformate bereitstellt.[4]
- Jegliche Abhängigkeit zum *Drupal-Search-Modul* wird unterbunden. Einerseits bietet das *Drupal-Search-Modul* keinen für diese Dokumente geeigneten Index, andererseits wird aus diesem Grund die Suche mit dokumentenspezifischen Eigenschaften unbrauchbar.

Installation und Deinstallation des Moduls

- Folgend auf die Installation des Moduls werden die Datenbanktabellen für die Indexierung angelegt.
- Nach der Installation sowie der Aktivierung des Moduls sollte unmittelbar die Indexierung der bereits vorhandenen Inhalte in die bestehenden Datenbanktabellen eingeleitet werden. Dazu dient ein cron-Lauf.
- Im Anschluss an die Deaktivierung des Moduls werden lediglich die Modul-Funktionalitäten abgestellt.
- Während der Deinstallationsroutine werden alle in der Datenbank angelegten Tabellen samt Inhalt gelöscht.

Konfiguration des Moduls

- Eine Konfigurationsseite für das Modul wird im administrativen Bereich von Drupal zur Verfügung gestellt.
- Weiterhin sind Einstellmöglichkeiten bezüglich benutzerspezifischer Rechte des Moduls bereitgestellt.
- Erweiterte Einstellmöglichkeiten für Farben[5] sowie Knopftexte und sonstige Texte, die in den Formularen auftauchen, werden zugefügt.

4 Dies ist notwendig, da das Modul nur für von *TEI-Lite* nach HTML transformierte Formate geeignet ist.
5 Falls ein Administrator ein Farbschema ändern möchte und das Modul an dieses anpassen kann.

Indexierung

- Die Indexierung dient der Erfassung von Texten, die als TEI kodierte Inhalte in Drupal vorliegen. Dazu werden die drei während der Installation erzeugten Datenbanktabellen genutzt. Eine der Tabellen wird zur Erfassung einzelner Wörter pro Node benötigt. Jedem Wort wird die Position zugewiesen, an der es sich befindet. Die Position ergibt sich aus der Node-Nummer und der Seitenzahl. Des Weiteren werden den Wörtern, zusätzlich zu der Position, die verschiedenen Tags zugewiesen, von denen sie umschlossen werden. Beispielsweise können diese zu einer bestimmten Textversion[6] zugehörig sein oder in einer Notiz des Autors vorkommen. Die zweite Datenbanktabelle dient der seitenweisen Erfassung von bereinigten Texten, wobei sich bereinigt[7] auf Sonderzeichen bezieht. Dadurch kann sie für die Suche ganzer Sätze eingesetzt werden. Jedem dieser Texte wird eine Seitenzahl und eine Node-Nummer zugewiesen, um die Position des jeweiligen Textes berücksichtigen zu können. Das Erfassen der Positionen für Texte sowie einzelne Wörter wird somit einen wesentlichen Vorteil gegenüber der Standard-Suche von Drupal bieten, da diese lediglich das Erfassen des Vorkommens auf einer Node bietet. Die dritte Datenbanktabelle erfasst alle Texte seitenweise, jedoch mit allen enthaltenen Sonderzeichen. Diese wird für die Erzeugung von Textausschnitten genutzt.
- Die Indexierung neuer Dokumente, deren Daten in die drei Datenbanktabellen aufgenommen werden, sollte unmittelbar nach dem Hochladen der Inhalte mit Hilfe eines cron-Laufs angestoßen werden.

Suche

- Das Suchformular erscheint auf Seiten, auf denen transformierte TEI-Dokumente angezeigt werden. Es dient der Auslösung einer Suchanfrage an die Datenbank .
- Beim Auslösen einer Suchanfrage durch das Betätigen eines Knopfes, erfolgt eine Umleitung zu der Seite, auf der die Suchergebnisse präsentiert werden.
- Die Ergebnisseite enthält neben den Suchtreffern ein weiteres Suchformular, das einer erweiterten Suche dient.
- In diesem Suchformular können Einschränkungen in Bezug auf das Vorkommen von Wörtern gesetzt werden, die von bestimmten Tags umschlossen sind.
- Bei einem Klick auf einen Suchtreffer wird eine Weiterleitung zur Textstelle eingeleitet, die den gefundenen Suchschlüssel beinhaltet.

Diese Punkte stellen den Leitfaden für die Implementierung des Suchmoduls dar und können als Meilensteine betrachtet werden.

Im nächsten Kapitel wird die Handhabung des fertigen Moduls vorgestellt. Dies dient einerseits der Präsentation der Funktionalitäten des Moduls und andererseits dem besseren Verständnis des Implementierungsvorgangs.

---

6 Originalfassung oder korrigierte Fassung des Textes.

7 Dazu gehören Kommata, Semikolon, Anführungszeichen, etc. Jedoch nicht die in französischen Texten vorkommenden Vokale mit Akzenten.

3 Arbeiten mit dem TEISearch-Modul

Das Arbeiten und der Umgang mit dem *TEISearch-Modul* lässt sich in fünf Abschnitte unterteilen. Durch die Installation beziehungsweise Aktivierung des Moduls werden zunächst die Funktionen, die das Modul mit sich bringt, für Drupal verfügbar gemacht. Die Konfiguration des Moduls beinhaltet diverse Einstellungsmöglichkeiten für Farben, mit denen die Suchtreffer hervorgehoben werden sollen sowie die Position des Suchformulars, das auf Seiten mit transformierten TEI-Dokumenten bereitgestellt wird. Der Abschnitt zum Umgang mit der *TEISearchbox* und der Ergebnisseite beschreibt die Möglichkeiten, die die Suchformulare bieten und auf welche Art die Suchergebnisse zu interpretieren sind. Den letzten Punkt stellt die Deinstallation des Moduls dar.

3.1 Installation und Aktivierung des TEISearch-Moduls

Um das *TEISearch-Modul* installieren zu können, muss ein laufendes Drupal in der Version 7 vorliegen.

Das Modul muss zunächst für Drupal bereitgestellt werden. Dies geschieht, indem der Ordner mit dem entsprechenden Inhalt des *TEISearch-Moduls* im Verzeichnis `/.../drupal/sites/all/modules` abgelegt wird. Zu beachten ist, dass Drupal für das Verzeichnis des Moduls bei der Installation die nötigen Rechte hat, damit das Modul zunächst eine initiale CSS-Datei erzeugen kann. Des Weiteren sollte sichergestellt werden, dass Drupal nach der Erzeugung der CSS-Datei Leserechte für diese besitzt, da sonst das Suchformular nicht erscheinen wird.

Nachdem diese Vorbereitungen getroffen wurden, wird das *TEISearch-Modul* im administrativen Bereich unter dem Abschnitt Module zu finden sein. Das Modul besitzt eine Abhängigkeit zum *TEIChi-Modul* und kann nur gleichzeitig mit diesem aktiviert werden oder wenn das *TEIChi-Modul* bereits aktiv ist (vgl. Abb. 3-1). Nach der Aktivierung des Moduls können nun die Rechte den eigenen Wünschen entsprechend verändert werden, wobei Drupal weiterhin für die Datei `/.../drupal/sites/css/teisearch.css` Schreibrechte benötigt, um spätere Änderungen, wie Farben und Position der *TEISearchbox* sowie Knopfbezeichnungen etc. speichern zu können.

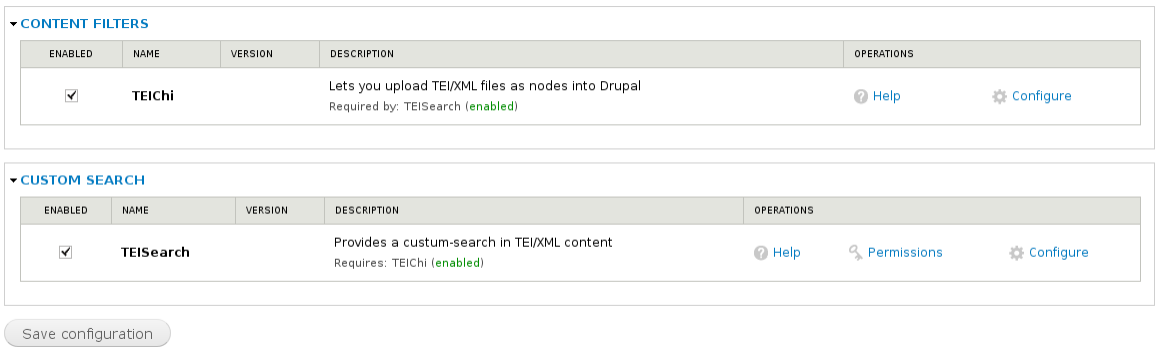

Abb. 3-1 *Module TEIChi und TEISearch sind aktiv und bieten unter OPERATIONS Einstellmöglichkeiten an*

An dieser Stelle sei darauf hingewiesen, dass das *TEISearch-Modul* keinerlei Abhängigkeit zum im Drupal enthaltenen *Search-Modul* aufweist. Demnach ist es, falls nur die *TEISearch-Suche* zum Einsatz kommen soll, nicht nötig das *Search-Modul* zu aktivieren.

Wenn vor der Installation oder während Inaktivität des *TEISearch-Moduls* TEI kodierte Inhalte in Drupal hochgeladen wurden, muss ein `cron`-Lauf ausgeführt werden. Dabei werden alle Nodes, die *teichinode*-Textformate enthalten, zum Index des Moduls hinzugefügt. Dies gilt auch nach dem Hochladen, während das Modul aktiv ist. Nach dem erfolgreichen Hinzufügen einer *teichinode* erscheint ein Hinweis, dass das *TEISearch-Modul* aktiv ist und ein `cron`-Lauf ausgeführt werden soll, damit der neu erstellte Inhalt für die Suche zur Verfügung gestellt werden kann.

3.2 Konfiguration

Das *TEISearch-Modul* bietet die Möglichkeit, festzulegen welche Benutzer die Suche[8] nutzen und Einstellungen am Modul vornehmen dürfen. Dazu erscheint nach der Aktivierung des Moduls ein Eintrag mit der Bezeichnung *Permissions* (vgl. Abb. 3-1) neben dem Modulnamen. Nach einem Klick auf diesen Link gelangt man in den administrativen Bereich von Drupal, in dem benutzerspezifische Einstellungen bezüglich der Rechte gesetzt werden können. Unter anderem ist ein Eintrag für das *TEISearch-Modul* zu sehen, in dem sich die Benutzerrechte verwalten lassen (vgl. Abb. 3-2).

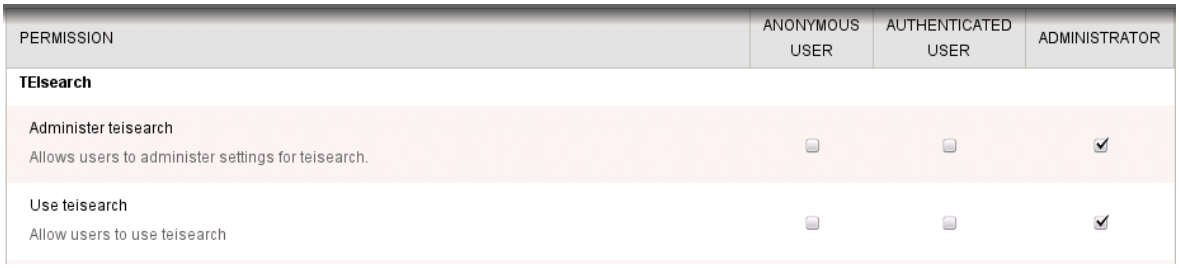

Abb. 3-2 *Permissions-Menü mit Kontrollkästchen für benutzerspezifische Einschränkungen*

Nachdem die benutzerspezifischen Berechtigungen an die Bedürfnisse angepasst wurden, können weitere Modul-Einstellungen vorgenommen werden.

Um sicherzustellen, dass der *TEISearch-Filter*[9] aktiv ist, sollte im administrativen Bereich die Seite für Textformate aufgesucht werden. Dort steht eine Übersicht mit den für Drupal verfügbaren Textformaten bereit. Zu dieser Übersicht gelangt man über *Configuration → Text formats*[10]. Darunter ist ein Textformat namens *teichinode* aufgeführt. Mit einem Klick auf *configure* für dieses Format gelangt man zu den Optionen, die für dieses Textformat gesetzt werden können. Unter anderem ist im Bereich *Enabled filters* der *TEISearch Filter* verfügbar (vgl. Abb. 3-3). Bei diesem sollte ein Häkchen gesetzt sein.

8 Hier ist nur das Durchsuchen aller Nodes gemeint. Die Suche aus der ursprünglichen *TEISearchbox* für die aktuelle Node ist für alle Benutzer erlaubt.
9 Der *TEISearch-Filter* muss aktiv sein, da sonst die *TEISearchbox* nicht angezeigt wird.
10 Stand: Drupal 7 alpha6-Version

Falls dies nicht der Fall ist, sollte es nachgeholt werden[11].

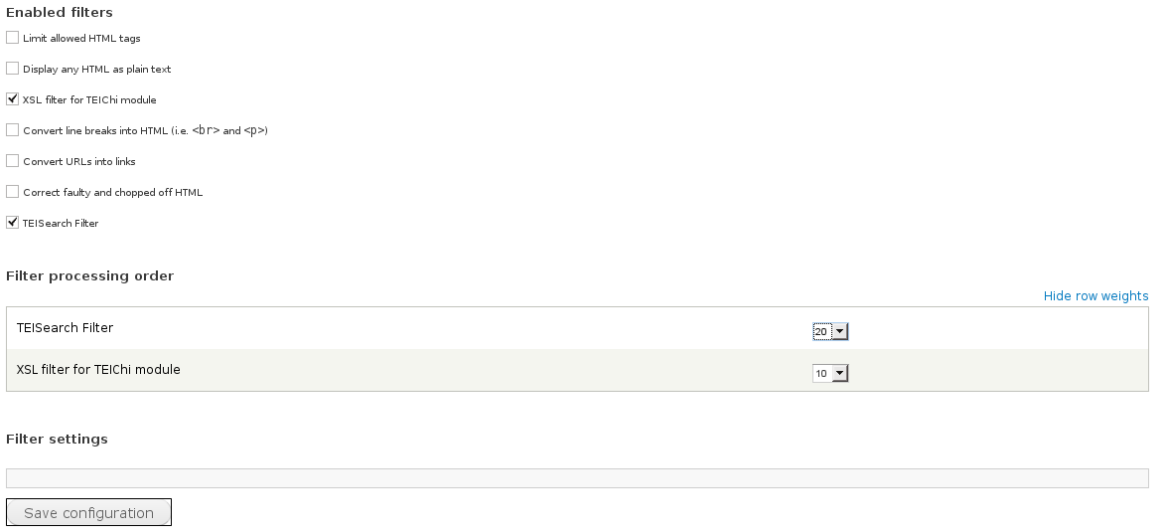

Abb. 3-3 *Filter-Menü für teichinode-Textformate*

Unter *Filter processing order* sollte nach einem Klick auf *Show row weights* überprüft werden, ob der eingetragene Wert für *TEISearch Filter* größer als der des *XSL filter for TEIChi module* ist. Das ist wichtig, da diese Werte die Reihenfolge für die Ausführung der Filter festlegen und der Filter des *TEIChi-Moduls* zuerst ausgeführt werden muss. Nach dieser Einstellung erscheint die *TEISearchbox* auf den Nodes, die *teichinode*-Textformate beinhalten.

Nachdem die grundlegenden Einstellungen des Moduls vorgenommen wurden, können weitere Konfigurationen erfolgen. Auf der Konfigurationsseite von Drupal für Module sind beim *TEISearch-Modul* neben dem *Permissions*-Link zwei weitere Links zu sehen. Einer davon hat Bezeichnung *Help*, der andere *Configure* (vgl. Abb. 3-1). Über beide Links gelangt man zur Konfigurationsseite des Moduls. Der Link *Help* führt zu einem Menü, das alle Konfigurationsseiten des Moduls anzeigt[12]. Der Link *Configure* stellt eine Weiterleitung zu *TEISearch Preferences* dar.

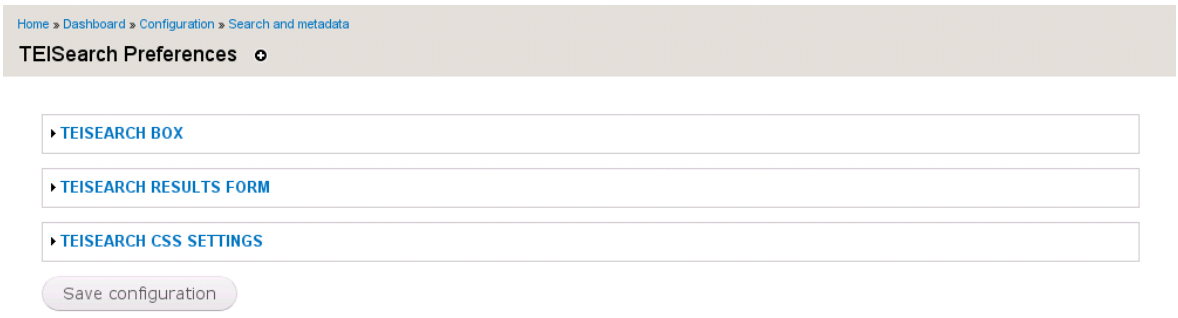

Abb. 3-4 *TEISearch Preferences-Menü mit zugeklappten Untermenüs*

Beim Aufruf dieser Seite sind drei ausklappbare Menüs zu sehen (vgl. Abb. 3-4). Hier können diverse Einstellungen für die *TEISearchbox*, das Suchformular auf der Ergebnis-

11 Dieser Fall kann eintreten, wenn das *TEIChi*- und *TEISearch-Modul* nicht gleichzeitig aktiviert wurden.
12 Bei diesem Modul sind es genau zwei Konfigurationsseiten, *Permissions* und *TEISearch Preferences*.

seite sowie CSS Eigenschaften des *TEISearch-Moduls* vorgenommen werden.
Im Abschnitt *TEISEARCH BOX* kann der Knopftext für den Such-Knopf der *TEISearchbox* festgelegt werden (vgl. Abb. 3-5). In der aktuellen Version des *TEISearch-Moduls* wurde das *SELECT DROPDOWN TEXT*-Untermenü ausgeblendet, da die neue *TEISearchbox* die Dropdown-Liste nicht mehr beinhaltet (vgl. Kap. 4.4.2).

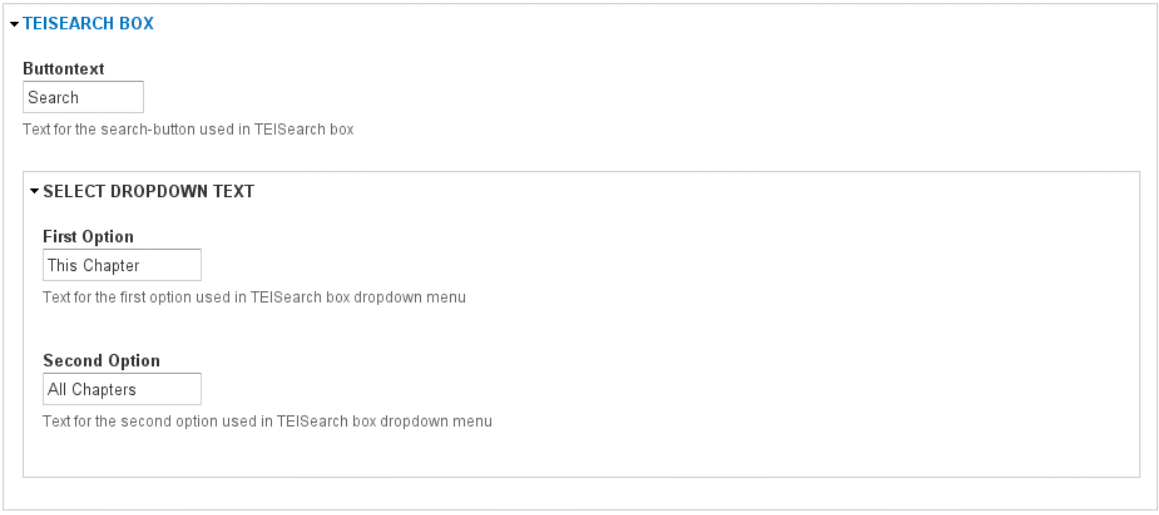

Abb. 3-5 *Aufgeklapptes Konfigurationsmenü der TEISEARCH BOX*

Der Abschnitt *TEISEARCH RESULTS FORM* bietet die Möglichkeit, den Knopftext, einen Infotext sowie die Texte für die Kontrollkästchen, die im Formular der Ergebnisseite erscheinen, zu bearbeiten. Für diese können die voreingestellten Texte nach Wunsch verändert werden.

Die Position und Farbwerte der *TEISearchbox* sowie für die Farbe der hervorgehobenen Suchtreffer können im Bereich *TEISEARCH CSS SETTINGS* gesetzt werden. Beim Speichern dieser Werte ist darauf zu achten, dass Drupal die nötigen Schreibrechte für die CSS-Datei besitzt, damit die vorgenommenen Änderungen gespeichert werden können.

3.3 Umgang mit der TEISearchbox

Nach allen Konfigurationsmaßnahmen wird die *TEISearchbox* näher betrachtet. Die *TEISearchbox* ist ein Formular, das der Suche dient. Sie enthält ein Eingabefeld sowie einen Such- und Reset-Knopf (vgl. Abb. 3-6). Die *TEISearchbox* befindet sich rechts neben dem Text. Die Position entspricht dem voreingestellten Wert, wobei die Ausrichtung in *TEISearch Preferences* verändert werden kann.

Um eine Suche auszuführen, muss zunächst ein gültiger Suchschlüssel in das Eingabefeld eingetragen werden. Ein gültiger Schlüssel entspricht einer Zeichenkette, die mehr als zwei Zeichen enthält. Zum Starten der Suche muss der Such-Knopf betätigt werden. Darauf erfolgt eine Umleitung zur Ergebnisseite.

Der Reset-Knopf setzt die Suche zurück. Mit Zurücksetzen ist hierbei das Entfernen aller Markierungen im Text, die die Suchtreffer hervorheben und das Leeren der Suchbox gemeint.

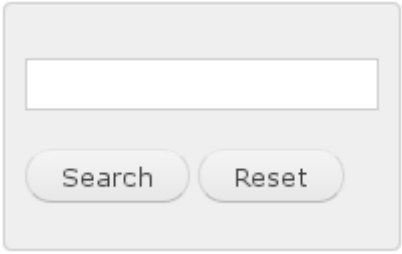

Abb. 3-6 *Das Suchformular TEISearchbox*

Im folgenden Abschnitt wird die ursprüngliche Form der *TEISearchbox* beschrieben. Die Änderungen ergaben sich aus einer Rücksprache mit dem Projektinitiator. Da die alte Ausführung dieses Formulars weiterhin im Quelltext enthalten ist und bei Bedarf zu dieser zurückversetzt werden kann, wird nun näher auf deren Funktionsweise eingegangen.

3.4 Arbeiten mit der ursprünglichen TEISearchbox

In dieser Ausführung der *TEISearchbox* ist ein Eingabefeld, eine Dropdown-Liste und ein Knopf zum Bestätigen einer Suchanfrage vorhanden (vgl. Abb. 3-7 u. 3-8).

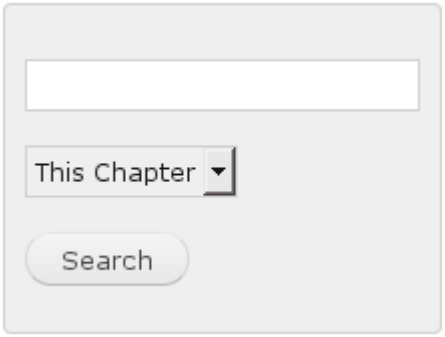

Abb. 3-7 *Ursprüngliche TEISearchbox mit zugeklapptem Menü*

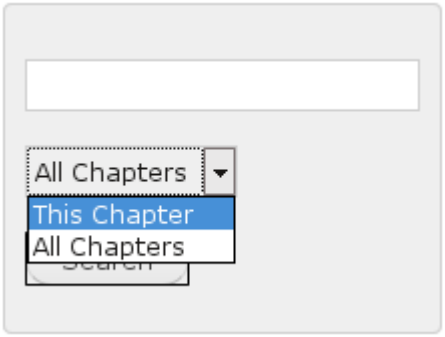

Abb. 3-8 *Ursprüngliche TEISearchbox mit aufgeklapptem Menü*

Aus diesem Formularfeld werden Suchanfragen gestellt. Die Dropdown-Liste bietet die Möglichkeit, die Suche entweder für die aktuelle oder für alle Nodes durchzuführen. Der voreingestellte Wert entspricht der Suche auf dem aktuellen Node. Der Unterschied dieser beiden Optionen besteht darin, dass beim Durchsuchen des aktuellen Nodes ein Sprung zur Seite beziehungsweise zu der Textstelle des ersten Treffers erfolgt und beim Durchsuchen aller Nodes eine Weiterleitung zur Ergebnisseite eingeleitet wird.

Angenommen, es handelt sich auf einem aktuellen Node um einen französischen Text, der die Zeichenkette „*parsemées de plusieurs*“ beinhaltet. Die Suche nach dieser Zeichenkette, mit Hilfe der Suche auf dem aktuellem Node, liefert einen Treffer. Es erfolgt ein Sprung zu der Seite im Text, auf der sich die gefundene Zeichenkette, die hervorgehoben dargestellt wird, befindet (vgl. Abb. 3-9).

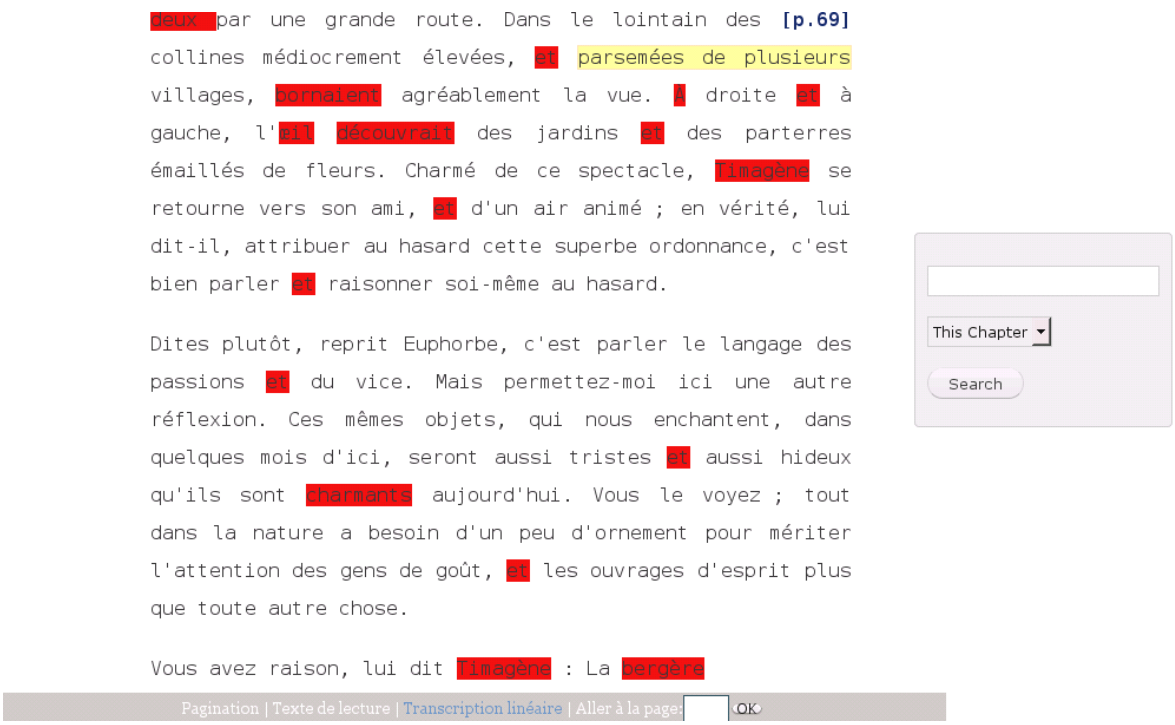

Abb. 3-9 *Gelb hervorgehobener Treffer im Text*

In dieser Abbildung ist die Zeichenkette mit gelber Farbe unterlegt und mit einem leicht abgeänderten Gelbton umrandet. Diese Farben sind voreingestellte Werte, die wiederum im Konfigurationsmenü des Moduls an die eigenen Bedürfnisse angepasst werden können.

Beim Setzen der Option zum Durchsuchen aller Nodes erfolgt eine Weiterleitung zur *TEISearch Results*-Seite, die die Suchergebnisse für alle Nodes anzeigt.

3.5 Die Ergebnisseite

Wie in den Abschnitten zuvor beschrieben, gelangt man zu der Ergebnisseite mit Hilfe der *TEISearchbox*.

Auf der Ergebnisseite befindet sich ein weiteres Suchformular. Dieses beinhaltet ein ausklappbares Menü, in dem suchspezifischen Optionen gesetzt werden können. Diese erscheinen in Form von Kontrollkästchen, durch die Einschränkungen der Suche für bestimmte Tags vorgenommen werden können. Das gilt jedoch nur, wenn die eingegebene Zeichenkette ein einzelnes Wort darstellt. Falls in das Eingabefeld eine Zeichenkette mit mehreren Wörtern, die durch ein Leerzeichen getrennt sind, eingetragen wird und zusätzlich Tags in Kontrollkästchen ausgewählt werden, wird die Suche für jedes einzelne Wort ausgeführt. Für TEI kodierte Dokumente bietet dieses Formular die Suche nach Wörtern, die von quotes-, notes-, quotes in notes-, reg-, corr-, orig- und sic-Tags umschlossen werden (vgl. Abb. 3-10).

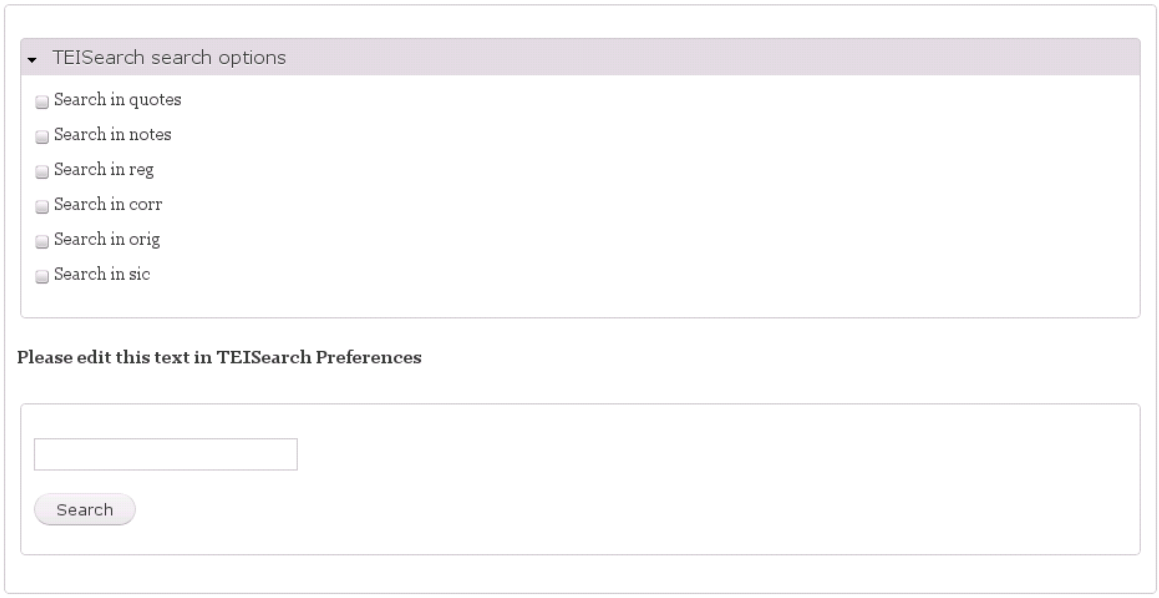

Abb. 3-10 *Aufgeklapptes TEISearch Results-Formular mit Kontrollkästchen für Suchoptionen*

Aufgrund der Suchlogik beziehungsweise des Suchalgorithmus, ist es in dieser Version des *TEISearch-Moduls* nicht möglich nach Sätzen zu suchen, die in einem oder mehreren Tags vorkommen.

Die Suche arbeitet für Sätze wie folgt:

- ➔ Falls ein Satz in das Eingabefeld eingetragen wird, versuche diesen Satz, falls keine Option für Tags gesetzt wurde, in der entsprechenden Datenbanktabelle zu finden und auszugeben.[13]
- ➔ Wenn solch ein Satz nicht in dieser Tabelle existiert, versuche, jedes einzelne Wort, verknüpft durch ein *ODER* zu finden und auszugeben.

Für einzelne Wörter:

- ➔ Falls ein einzelnes Wort in das Eingabefeld eingetragen wird, eventuell mit gesetzten Optionen für Tags, versuche dieses Wort in der entsprechenden Datenbanktabelle, unter Berücksichtigung der gesetzten Optionen zu finden und auszugeben.
- ➔ Wenn solch ein Wort in der Tabelle nicht existiert, versuche alle ähnlichen Wörter in der selben Tabelle, unter Berücksichtigung der gesetzten Optionen zu finden und auszugeben.[14]

Dieses Suchformular ist für den Fall gedacht, dass die Suche aus der *TEISearchbox* heraus nicht die gewünschten Treffer liefert, so dass an dieser Stelle eine erneute Suche ausgeführt werden kann, eventuell durch Einschränkung auf bestimmte Tags mit Hilfe der Optionen, die durch die Kontrollkästchen geboten werden.

Falls eine Suche erfolgreich war und Ergebnisse erzielt wurden, werden diese unterhalb

13 Dies bedeutet, dass eine Wortfolge nach der gesucht werden soll nicht in Anführungszeichen oder Ähnliches gesetzt werden muss.

14 Ähnliche Wörter sind in diesem Fall in Form von %wort%, d.h. im Fall %que% wären auch die Wörter 'quelque', 'quelle', 'unique' etc. Suchtreffer.

des Suchformulars dargestellt. Diese sind nach Seiten sortiert, pro Seite eines Kapitels beziehungsweise einem Node. Angenommen es befinden sich auf einem *Node '1'* und einem *Node '9'* die gleichen Seitenzahlen, zum Beispiel die *Seite '4'*, die beide das gesuchte Wort beinhalten, dann werden diese zunächst für den *Node '1'* und *Seite '4'* und direkt darunter für *Node '9'* und *Seite '4'* ausgegeben.

Hierbei macht es kein Unterschied, ob es sich bei einem Node um ein Buch, Buchkapitel oder sonstige Formate handelt. Der Node muss lediglich die Eigenschaft besitzen, ein *teichinode* zu sein und eine eindeutige beziehungsweise einmalige Node-Nummer haben. Da es in Drupal nicht möglich ist mehrere Nodes mit der selben Node-Nummer zu erstellen, kommt diese Tatsache dem *TEISearch-Modul* entgegen.

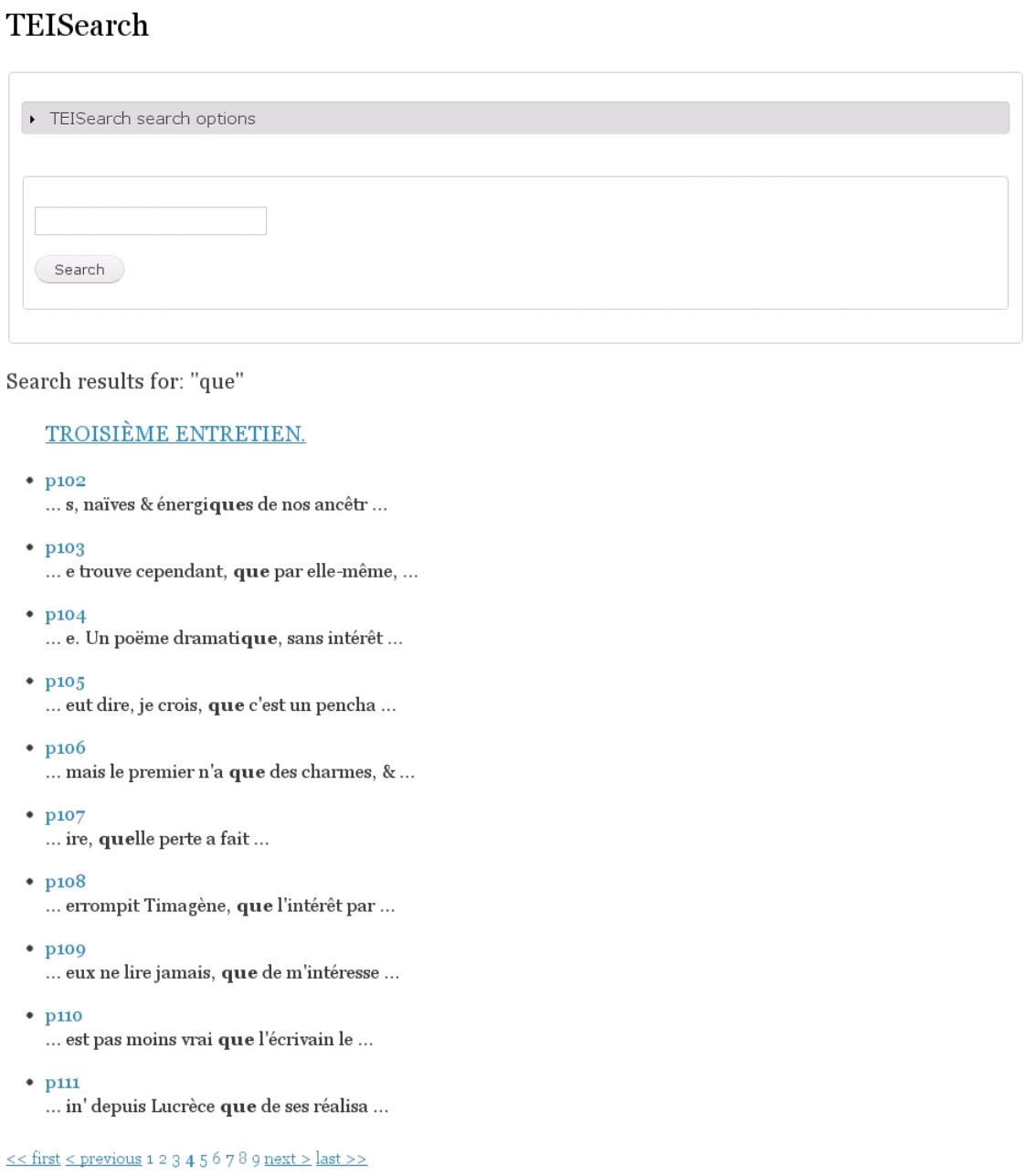

Abb. 3-11 *Ergebnisseite mit Treffern für eine Suche nach "que"*

Angenommen, es wird eine Suche für das Wort „*que“* ausgeführt, die *94* Treffer liefert. Diese Treffer entspringen alle aus einem Node, der den Titel „*TROISIÈME ENTRETIEN“* hat (vgl. Abb. 3-11). Hier ist erkennbar, wie eine Ausgabe von relativ vielen Treffern aussieht. Zunächst ist unter *Search results for:“que“* die Zeichenkette aufgeführt, zu der die Suchergebnisse gehören. Unterhalb ist der Titel des Nodes sichtbar, auf dem sich die Treffer befinden. Die einzelnen Treffer werden einmalig pro Seite jedes Nodes aufgeführt. Hintergrund hierfür ist die Tatsache, dass es nutzlos ist, mehrfach die selben Links zu erzeugen, die zur selben Seite und somit zur selben Textstelle umleiten würden. Was jedoch Sinn machen würde, ist die Ausgabe aller Textausschnitte, die den Suchbegriff auf einer Seite beinhalten. In dieser Version des *TEISearch-Moduls* wird immer nur ein Textausschnitt produziert und zwar für das erste Vorkommen der gesuchten Zeichenkette auf der jeweiligen Seite. Nach einem Klick auf einen der Links erfolgt ein Sprung zu derjenigen Seite im Text, auf der sich die gefundene Zeichenkette, die hervorgehoben dargestellt wird, befindet.

Unter den einzelnen Suchtreffern befindet sich ein Pager. Wie in Abbildung 3-11 zu sehen ist, ist die Zahl vier im Pager fett hervorgehoben. Das bedeutet, dass man sich auf der Trefferseite *4* befindet, auf der die Treffer *30* bis *40* aufgelistet werden. Mit Hilfe des Pagers ist die Möglichkeit zur Navigation durch alle Suchtreffer gegeben.

3.6 Deaktivierung und Deinstallation des TEISearch-Moduls

Der Unterschied zwischen Deinstallation und Deaktivierung besteht darin, dass bei der Deaktivierung das Modul inaktiv wird und dessen Funktionen nicht mehr zur Verfügung stehen. Bei der Deinstallation hingegen werden zusätzlich alle vom Modul angelegten Datenbanktabellen samt Inhalt gelöscht.

Das Modul lässt sich im administrativen Bereich von Drupal-Modulen deaktivieren, indem das Häkchen beim *TEISearch-Modul* entfernt wird und die Einstellungen gespeichert werden (vgl. Abb. 3-1). Nach der Deaktivierung des Moduls ist die Option zur Deinstallation verfügbar, die durch einen Klick auf den *Uninstall-Tab* erreicht werden kann.

Beim erneuten Aktivieren des Moduls, nach einer Deinstallation, ist ein `cron`-Lauf nötig, damit alle *teichinode* Textformate erneut indexiert werden.

Folgend wird der Prozess der Implementierung im Vordergrund stehen. Die Umsetzung wird Schritt für Schritt, teilweise unter Zuhilfenahme von Quelltextausschnitten, detailliert präsentiert.

4 Implementierung

Es folgten zunächst erste Programmierversuche, um ein Gefühl für das Arbeiten mit der *Drupal-API* zu bekommen. Hierfür wurde der Versuch unternommen, ein Modul zu implementieren, welches einen Block mit einem Formular erzeugt.

Die Implementierung des Test-Moduls verlief zunächst reibungslos, doch nach einigen Überlegungen, an welcher Stelle der Block samt Formular erscheinen soll, erwies sich der Ansatz das Suchformular mit Hilfe eines Blocks umzusetzen als ungeeignet. Da das Modul für TEI-Textformate genutzt werden soll, ist es nicht notwendig, den Block auf allen Seiten anzuzeigen. Sollte der Block zudem nur auf bestimmten Seiten angezeigt werden, müssten die Seiten, auf denen der Block angezeigt werden soll, manuell eingetragen werden. Unter Berücksichtigung der Benutzerfreundlichkeit wurde der erste Ansatz verworfen und der Idee nachgegangen, das Suchformular als einen Filter einzubinden. Der Filter würde dabei nur dann zum Einsatz kommen und das Formular erzeugen, wenn eine Seite mit einem Inhalt aufgerufen würde, die bereits mittels des XSL-Stylesheets transformiert wurde. Nähere Details zum Filter für `teichinode`-Textformate sind im Kap. 4.4.1 zu finden.

So stellten diese Versuche die ersten praktischen Übungen im Umgang mit der *Drupal-API* dar.

Die Implementierung kann in drei Entwicklungsphasen unterteilt werden. Zunächst erfolgt die Implementierung der Einstellmöglichkeiten, die von einem Seitenadministrator verwaltet werden können. Dazu gehören sowohl die Verwaltung von Zugriffsrechten, wie auch die optischen Einstellungen, die das Modul bereitstellt. Des Weiteren gehören zu diesem Abschnitt die Installation, Aktivierung, Deaktivierung sowie die Deinstallation des *TEISearch-Moduls*[15].

Den zweiten Schritt stellt die Implementierung des Index dar. Hier wird die Erfassung und Speicherung von Daten, die den Index in Form von Drupal-Datenbanktabellen bilden werden, thematisiert. Dazu werden TEI kodierte Inhalte untersucht und für den Index irrelevante Zeichenketten herausgefiltert. Nach dem Filtervorgang werden die daraus hervorgegangenen Daten dem Index zugeführt.

Der dritte Abschnitt ist der Implementierung der Suche gewidmet. In diesem werden die Suchalgorithmen, Suchformulare, Darstellung der Treffer mit Hilfe von *PHP-Templates* (vgl. [26]) und das Hervorheben von Funden beschrieben.

4.1 Modulbasis

Nach dem weiter oben beschriebenen Testversuch folgte die Implementierung des Such-Moduls. Als Entwicklungsumgebungen standen für den gesamten Implementierungsvorgang zwei Rechner bereit.[16]
Zunächst wurden die Dateien `teisearch.module`, `teisearch.info` und `teisearch.install`, im Verzeichnis:

15 Namensgebung für Modul wird weiter unten aufgeführt.
16 Vgl. hierzu: Anhang A – Systemspezifikationen.

```
/.../drupal/sites/all/modules/teisearch/
```

angelegt, die das Grundgerüst des Moduls darstellen. An dieser Stelle wird nur der Inhalt der Datei `teisearch.info` (vgl. [24]) aufgeführt, der wie folgt aussieht:

```
; $Id$
name = "TEISearch"
description = "Provides a custum-search in TEI/XML content"
core = 7.x

files[] = teisearch.module
package = "Custom Search"
dependencies[] = teichi
configure = admin/config/search/teisearch
```

Mit `name` wird der Modulname festgelegt, der im administrativen Bereich für Module in Drupal erscheint. `description` enthält eine Beschreibung des Moduls, die neben dem Modulnamen sichtbar ist. Der Eintrag `core` legt die Kompatibilität des Moduls fest, hier Drupal in Version 7. In `files` ist die Datei `teisearch.module` angegeben, die das eigentliche Modul repräsentiert. Der Eintrag `package` legt die Gruppe fest, der das Modul angehört, in diesem Fall `Custom Search`. Mit Hilfe von `dependencies[] = teichi` bekommt das Modul eine Abhängigkeit zum *TEIChi-Modul* zugeordnet, was bedeutet, dass das *TEISearch-Modul* ohne das aktive *TEIChi-Modul* nicht aktiviert werden kann. Da das *TEISearch-Modul* für Textformate eingesetzt werden soll, die das *TEIChi-Modul* bereitstellt, ist diese Abhängigkeit unumgänglich. Das ist der wesentliche Grund, weshalb hier auf die Datei `teisearch.info` explizit eingegangen wurde.

Weitere Dateien werden im Verlauf des Kapitels, jeweils an geeigneter Stelle, näher beleuchtet.

Insgesamt besteht das Modul aus den folgenden Dateien, die mit Hilfe des Befehls `tree` für den Ordner `teisearch` geliefert werden:

```
teisearch
|-- css
|   `-- teisearch.css
|-- teisearch-result.tpl.php
|-- teisearch-results.tpl.php
|-- teisearch.info
|-- teisearch.install
`-- teisearch.module

1 directory, 6 files
```

Wobei:

- `teisearch.css` CSS-Eigenschaften beinhaltet, die für die Positionierung und Festlegung von Farbwerten für die Suchformulare eingesetzt werden
- `teisearch-result.tpl.php` für die Darstellung einzelner Suchtreffer auf der Ergebnisseite benötigt wird

- `teisearch-results.tpl.php` für die Darstellung aller Suchtreffer auf der Ergebnisseite benötigt wird
- `teisearch.info`, wie weiter oben bereits beschrieben, einige Moduleigenschaften definiert
- `teisearch.install` die Installations- sowie Deinstallationsroutine des Moduls festlegt
- `teisearch.module` die Kernfunktionen des Moduls beinhaltet

Die Entstehung des Modulnamen *TEISearch* sowie die gleichnamigen Datei-Präfixe sind eine Zusammensetzung aus den TEI kodierten Dokumente und der Suchfunktion, die das Modul auszeichnen.

Eine kurze Bemerkung: Da mehrere Dateien aufgeführt wurden, die zusätzlich zum `teisearch.module`-Quelltext für die Funktionalität des Moduls beinhalten, werden diese Dateien an der entsprechenden Stelle explizit erwähnt. Falls keine Erwähnung stattfindet, ist die Rede vom Quelltext der Datei `teisearch.module`.

4.2 Administrative Eigenschaften des TEISearch-Moduls

4.2.1 Installation und Aktivierung

Beim erstmaligen Aktivieren des Moduls, das als Installation betrachtet werden kann, wird der Quelltext der `teisearch.install`-Datei ausgeführt. In dieser befindet sich die Installationsroutine für das *TEISearch-Modul*. Die Installationsroutine besteht darin, die für die Indexierung erforderlichen Tabellen und Felder in der Drupal-Datenbank anzulegen. Diese Tabellen repräsentieren den Index und dienen der Speicherung der zu indexierenden Daten.

Für die Erzeugung der Tabellen wird `hook_install()` auf die folgende Weise implementiert:

```
function teisearch_install(){
  drupal_install_schema("teisearch_index");
  drupal_install_schema("teisearch_dataset");
  drupal_install_schema("teisearch_non_cleaned_up_dataset");
  drupal_set_message("Installing the TEISearch module...");
}
```

Demnach werden die Folgenden drei Tabellen angelegt:

```
-"teisearch_index"
-"teisearch_dataset"
-"teisearch_non_cleaned_up_dataset"
```

Die Eigenschaften der Tabellen beschreibt die Funktion `teisearch_index`

`_schema()`. Der folgende Auszug aus dieser Funktion soll verdeutlichen, wie Definitionen einzelner Felder aufgebaut sind:

```
$schema['teisearch_dataset'] = array(
  'description' => 'Stores items that will be searched.',
    'fields' => array(
      'page' => array(
      'type' => 'varchar',
      'length' => 20,
      'not null' => TRUE,
      'description' => 'The page the text is
                        associated with.'
  ),
);
```

Hierbei handelt es sich um den Ausschnitt für die Definition der `teisearch_dataset` -Tabelle. Diese legt die Beschreibung des Schemas fest und definiert ein Feld namens `page` vom Typ `varchar`, unter anderem mit der Eigenschaft für die Länge der Zeichenkette von `20` Zeichen. Das geschieht analog hierzu für alle anderen Felder der Tabelle.

Die Tabellen, die mit dem SQL-Befehl `DESCRIBE` und jeweils mit Tabellennamen als Parameter abgefragt wurden, weisen nach der Erstellung die folgenden Eigenschaften auf:

```
$mysql> DESCRIBE `teisearch_dataset`;
+-------------+------------------+------+-----+---------+-------+
| Field       | Type             | Null | Key | Default | Extra |
+-------------+------------------+------+-----+---------+-------+
| page        | varchar(20)      | NO   | PRI | NULL    |       |
| textversion | varchar(20)      | NO   | PRI | NULL    |       |
| nodenr      | int(10) unsigned | NO   | PRI | 0       |       |
| data        | longtext         | NO   |     | NULL    |       |
+-------------+------------------+------+-----+---------+-------+

$mysql> DESCRIBE `teisearch_non_cleaned_up_dataset`;
+-------------+------------------+------+-----+---------+-------+
| Field       | Type             | Null | Key | Default | Extra |
+-------------+------------------+------+-----+---------+-------+
| page        | varchar(20)      | NO   | PRI | NULL    |       |
| textversion | varchar(20)      | NO   | PRI | NULL    |       |
| nodenr      | int(10) unsigned | NO   | PRI | 0       |       |
| data        | longtext         | NO   |     | NULL    |       |
+-------------+------------------+------+-----+---------+-------+

$mysql> DESCRIBE `teisearch_index`;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| nodenr          | int(11)     | NO   | PRI | 0       |       |
| word            | varchar(50) | NO   | PRI |         |       |
| searchtags      | varchar(40) | NO   | PRI |         |       |
| page            | varchar(20) | NO   | PRI |         |       |
```

```
| numeric_word_id | float       | NO   | PRI | 0       |       |
+-----------------+-------------+------+-----+---------+-------+
```

4.2.2 Das Konfigurationsmenü

Die Konfiguration des Moduls ist in drei Abschnitte unterteilbar. Einen Abschnitt davon bildet das sogenannte Permissions-Menü, das diverse Möglichkeiten zum Einschränken für benutzerspezifische Rechte bietet. Der zweite Abschnitt ist das Bereitstellen eines Filters im administrativen Bereich Drupals für Textformate, der für die Erzeugung eines Suchformulars sowie das Hervorheben von Suchtreffern benötigt wird. Den letzten Abschnitt bildet das administrative Menü für das *TEISearch-Modul*. Durch dieses Menü wird einem Administrator die Möglichkeit geboten, das Modul durch Änderungen von Farbwerten, Position des Suchformulars sowie Texten, die in Suchformularen erscheinen, individuell zu gestalten.

Permissions-Menü
Zunächst bietet das Modul die Möglichkeit, benutzerspezifische Einschränkungen für die Suche und Konfiguration bereitzustellen. Dies geschieht mittels der Implementierung von `hook_permission()`:

```
function teisearch_permission() {
  return array(
    'administer teisearch' =>  array(
      'title' => t('Administer teisearch'),
      'description' => t('Allows users to administer
          settings for teisearch.'),
    ),
    'use teisearch' => array(
      'title'       => t('Use teisearch'),
      'description' => t('Allow users to use teisearch')
    ),
  );
}
```

Damit ist es möglich, anhand der definierten Werte im Array `administer teisearch` sowie `use teisearch`, Zugriffsrechte zu berücksichtigen. Im Administrationsbereich von Drupal kann dadurch festgelegt werden, für welche Benutzer bestimmte Richtlinien gelten sollen.

Bereitstellung des Filters
Der zweite Punkt besteht aus der Filteroption für Textformate, die als TEI kodierte Inhalte in Drupal vorliegen. Diese Dokumentenformate werden im administrativen Bereich für Textformate unter der Bezeichnung `teichinode` aufgeführt. Das Textformat `teichinode` wird vom *TEIChi-Modul* erzeugt, zu dem das *TEISearch-Modul* eine Abhängigkeit aufweist. Mit der Implementierung der Funktion `hook_filter_info()` wird der Filter für die beschriebenen Inhalte erzeugt:

```
function teisearch_filter_info() {
   $filters['teisearch'] = array(
      'title' => t('TEISearch Filter'),
      'process callback' =>
                          '_teisearch_tags_filter_process',
      'tips callback' => '_teisearch_filter_tips',
      'weight' => 20,
   );
  return $filters;
}
```

Die Funktion definiert den Filter `teisearch`, der den Titel *TEISearch Filter* trägt. Das Array-Element `'process callback'` legt die auszuführende Funktion `_teisearch_tags_filter_process` fest, die den Filtervorgang einleitet. Sobald ein Node mit einem Textformat `teichinode` aufgerufen wird, beginnt die Filterung in Form von Textersetzungen an dieser Stelle. Näheres zur eigentlichen Filterfunktionalität ist in Kap. 4.4.1 zu finden. Diese Einstellungen für den *TEISearch Filter* lassen sich im Administrationsmenü unter dem Unterpunkt 'Text formats' vornehmen.

***TEISearch Preferences*-Menü**

Den letzte Punkt der Implementierung des Konfigurationsmenüs stellt das so genannte *TEISearch Preferences*-Menü dar. Dieses Menü lässt die Konfiguration der Position, Farbwerte und Texte für die *TEISearchbox* sowie für die Suchformulare zu. Dies wird in der Funktion `teisearch_admin_settings()` implementiert. Da diese Funktion sehr umfangreich ist und die Struktur für jedes Element sehr ähnlich ist, wird hier nur ein Element zur Verdeutlichung des Aufbaus als Beispiel herangezogen:

```
$form['teisearch_searchresults_form']
  ['teisearch_checkboxes']['teisearch_checkbox_quotes'] =
  array(
    '#type' => 'textfield',
    '#title' => t('quotes'),
    '#description' => t('Text for the quotes checkbox
                                      used in TEISearch'),
    '#default_value'=>variable_get(
         'teisearch_checkbox_quotes','Search in quotes'),
    '#size' => 30,
  );
```

Dieser Quelltextauszug beschreibt ein Eingabefeld vom Typ `'textfield'` mit dem Titel `quotes` sowie eine Beschreibung für das Feld und einen vordefinierten Text `Search in quotes`. Für das Eingabefeld, in das eine vom voreingestellten Text abweichende Bezeichnung eingetragen werden kann, wird eine feste Länge von 30 Zeichen zugewiesen.

Es handelt sich hierbei um den Eintrag eines Formulars, der auf der Konfigurationsseite des Moduls auftaucht. Dieses Formular hat den Zweck, den vordefinierten Text für ein

Kontrollkästchen des Suchformulars der Trefferseite zu ändern.

Nach der Definition aller Elemente des *TEISearch*-Konfigurationsformulars in der Funktion `teisearch_admin_settings()` ruft diese eine weitere Funktion auf, die eine CSS-Datei anlegt. In diese Datei werden die gesetzten Werte für das Aussehen und die Ausrichtung der *TEISearchbox* und des Suchformulars der Ergebnisseite sowie die zu hervorhebenden Wörter gespeichert:

```
function teisearch_css_create() {
...
$css=<<<CSS
    #teifieldset{
      position:{$position};
      right:{$right};
      top:{$top};
      background-color:{$bgcolor_teifieldset};
      z-index:2;
    }


    .teihighlight{
      background-color:{$bgcolor_teihighlight};
      border:1px solid {$border};
    }
CSS;
$cssPath=drupal_get_path('module','teisearch')."/css/
                                                teisearch.css";
file_put_contents($cssPath,$css);
}
```

Somit erzeugt die Funktion `teisearch_css_create()` eine Zeichenkette mit dem Inhalt für die CSS-Datei, wobei die Werte für Position und Farben aus dem Formular des *TEISearch Preferences*-Menüs abgerufen werden und schreibt diese mit Hilfe von `file_put_contents($cssPath,$css)`. `$cssPath` enthält hierbei den Pfad der zu erzeugenden Datei und `$css` die Zeichenkette, die in die Datei geschrieben wird. Falls diese bereits existiert, wird der Inhalt überschrieben, falls nicht, wird sie neu angelegt.

Beim Speichervorgang muss darauf geachtet werden, dass Drupal die nötigen Rechte besitzt, um den Inhalt beziehungsweise die Datei erzeugen zu können. Falls dies nicht der Fall sein sollte, ist die Folge daraus, dass Drupal entweder mit voreingestellten Werten arbeitet oder das Suchformular im schlimmsten Fall nicht auf einem Node erscheint.

4.2.3 Deaktivierung und Deinstallation

Die Deinstallationsroutine für das Modul befindet sich, wie auch die Installationsroutine, in der `teisearch.install`-Datei. Beim Deinstallieren des Moduls werden die Tabellen samt Inhalt vollständig aus der Drupal-Datenbank gelöscht, die während der Installation erzeugt wurden. Das bedeutet für das erneute Aktivieren, dass alle Nodes durch

Ausführung von `cron` im Index neu erfasst werden müssen. Diese Deinstallationsroutine wird durch die nachstehende Funktion eingeleitet:

```
function teisearch_uninstall(){
  drupal_uninstall_schema("teisearch_index");
  drupal_uninstall_schema("teisearch_dataset");
  drupal_uninstall_schema("teisearch_non_cleaned_up_dataset"
                          );
  drupal_set_message("Uninstalling the TEISearch
                          module...");
}
```

Die Funktion `teisearch_uninstall()` ruft für jedes existente Schema `drupal_uninstall_schema()` mit dem Namen des Schemas als Parameter auf. Dadurch wird der Löschvorgang der Tabellen ausgelöst.

4.3 Indexierung

Die Indexierung ist in drei Bereiche aufgeteilt. Dazu werden die drei Tabellen, die während der Installation des Moduls in der Drupal-Datenbank angelegt wurden, genutzt und mit Daten gefüllt.

Eine dieser Tabellen wird zur Erfassung aller Wörter samt den jeweils zugewiesenen Eigenschaften wie die Seitennummer, Tags, Node-Nummer, etc. genutzt.

Die zweite Tabelle nimmt aufeinanderfolgende Worte beziehungsweise Sätze, die sich auf ein und derselben Seite befinden, pro Node, Textversion und Seite auf. Die Textabschnitte werden zunächst von überflüssigen Sonderzeichen, Leerzeichen, etc. bereinigt und anschließend in die Datenbank geschrieben.[17]

Eine weitere Datenbanktabelle, die für das Hervorheben von Suchtreffern genutzt wird, dient der Speicherung von aufeinanderfolgenden Wörtern beziehungsweise Sätzen. Die Differenz zur zweiten Datenbanktabelle besteht darin, dass die Sätze nicht bereinigt werden müssen.

Die drei Abschnitte der Indexierung beschreiben das Erfassen von Daten in die drei Datenbanktabellen. Die Indexierung wird durch die folgende Funktion angestoßen, die während einer Ausführung von `cron` aufgerufen wird:

```
function teisearch_cron() {
  teisearch_index();
}
```

Die Nutzung von `cron` war ursprünglich nicht geplant. Zunächst wurden sämtliche Inhalte, im Anschluss an das Hochladen, indexiert. Als der Versuch unternommen wurde, zehn Inhalte in einem Lauf zu Indexieren, kam es zu folgender Fehlermeldung:

```
PHP Fatal error: Maximum execution time of 30 seconds
```

17 Zur genauen Struktur der Tabellen siehe Kap. 4.2.1.

exceeded...

Das war der Auslöser für den Einsatz von cron für den Indexiervorgang. Denn falls der Server den Vorgang im gegebenen Zeitrahmen nicht bewältigt kann, kommt es zum Abbruch ohne den Index vollständig erzeugt zu haben. Beim erneuten Versuch die Inhalte zu indexieren, würde die Funktion, die den Indexiervorgang ursprünglich eingeleitet hat, die bereits teilweise indexierten Nodes übergehen. Dies ist keinesfalls ein stabiles Verfahren zur Erzeugung eines Index. Die Zeitspanne von 30 Sekunden, die auf der Testumgebung gesetzt war, kann zudem leicht überschritten werden, wenn der Server zu diesem Zeitpunkt hinreichend ausgelastet ist.

Es besteht zwar die Möglichkeit die Zeitspanne in der entsprechenden Datei, der php.ini, zu erhöhen, wobei dieser Ansatz nicht sinnvoll ist, da der Server keine Anfragen während des Indexiervorgangs bearbeiten kann. Zudem ist teilweise diese Zeitspanne vom Provider festgelegt und somit nicht ohne weiteres änderbar. Demnach dient nun cron für die Ausführung einer Indexierung, da derartige zeitliche Beschränkungen damit umgangen werden.

4.3.1 Indexierung einzelner Wörter

Mit der Funktion teisearch_index(), die während eines cron-Laufs aufgerufen wird, beginnt der Indexiervorgang. In dieser Funktion wird geprüft, ob Inhalte zum Indexieren vorhanden sind. Dazu werden Informationen über Nodes abgerufen, die vom *TEIChi-Modul* verarbeitet wurden:

```
$teinodes = array_keys(teichi_get_node_infos());
```

wobei die Funktion teichi_get_node_infos() die dazu benötigten Werte liefert.

Die Werte, die in $teinodes gespeichert werden, unterliegen einem Vergleich mit den Werten der teisearch_dataset-Tabelle, wobei teisearch_dataset Texte pro Node, Seite und Textversion beinhaltet. Falls die Node-Nummern bereits in teisearch_dataset erfasst sind, werden die nächsten Nodes untersucht. Falls eine der Nodes, die in $teinodes zwischengespeichert sind, noch nicht erfasst wurden, wird mit der Indexierung dieser Nodes fortgefahren. Hierzu wird die Funktion:

```
teisearch_add_node_to_index($nodenr, $text);
```

aufgerufen und die Node-Nummer samt dem zu indexierenden Text mit der Variable $text übergeben, die den Inhalt der Node repräsentiert. Zur Erinnerung, diese Texte liegen im Format nach dem *TEI-Lite-Standard* vor.

Die minimale Zeichenlänge, die ein Wort haben muss, um in den Index aufgenommen zu werden, ist drei. Die Tags, die beim Indexieren berücksichtigt werden müssen, sind im $tags-Array definiert und mit Standardwerten versehen:

```
$tags = array(
  'pb' => 2,
  'note' => 3,
```

```
  'quote' => 3,
  'orig' => 4,
  'reg' => 4,
  'corr' => 4,
  'sic' => 4,
);
```

wobei `pb` für das *TEI*-Tag `<pb>` steht, das einen Seitenumbruch samt Seitennummer beinhaltet sowie die restlichen Tags, die die Zugehörigkeit der Wörter beziehungsweise Tags von denen die Wörter umschlossen sind, markieren.

Im nächsten Schritt wird der Text für die Indexierung vorbereitet, indem für den Index unnötige Abschnitte des Textes entfernt werden, so zum Beispiel der gesamte `teiHeader` (vgl. [29]) oder auch interne Anmerkungen, die nicht durchsuchbar sein sollen. Hierfür wird der Text als Zeichenkette untersucht und es werden mit Hilfe von PHP-String-Funktionen diverse Ersetzungen durchgeführt.

Eine besondere Ersetzung, die beim Bearbeiten der Zeichenketten vorgenommen wird, ist die Seitennummer, die am Anfang des Textes erzeugt wird. Es ist durchaus möglich, dass ein neues Kapitel beginnen kann, das sich auf der gleichen Seite wie das Kapitel zuvor befindet. In dem Fall würde dies bedeuten, dass ein Dokument, das ein Kapitel darstellt, beginnt ohne eine Seitennummer an dessen Anfang zu haben. Da der Index so konzipiert wurde, dass jedem Wort eine Seite zugewiesen werden muss, um dessen Position exakt bestimmen zu können, erfolgt an dieser Stelle eine künstliche Implantation einer Seitennummer für die zu indexierenden Inhalte. Der künstliche Seitenumbruch am Anfang des Texts wird wie folgt erzeugt:

```
$pos = strstr($text, '<pb p="p');
$firstpage_number = preg_replace('#"></pb>(.*)#s',
                                 '',substr($pos, 8, 20)) - 1;
$text = str_replace('<text>', '<pb
                        p="p'$firstpage_number.'">',$text);
```

Hierbei wird das erste Vorkommen eines Seitenumbruchs mit `strstr($text, '<pb p="p')` ermittelt, die Seitennummer extrahiert, zugleich dekrementiert und anschließend durch die Ersetzung des `<text>`-Tags erzeugt. Das `<text>`-Tag befindet sich am Anfang der bearbeiteten Zeichenkette, nachdem der gesamte `teiHeader` entfernt wurde, und ist für die Weiterverarbeitung während der Indexierung irrelevant, weshalb es ohne Bedenken ersetzt werden kann.

Nachdem die Zeichenkette von nicht benötigten Daten bereinigt wurde, wird sie an denjenigen Stellen, an denen Tags vorkommen, die im Array `$tags` definiert sind, mit der Funktion `preg_split()` aufgetrennt. Jeder dabei entstandene Teil, ein sogenannter Split, kann entweder ein Tag oder ein vor der Trennung von Tags umschlossener Text sein. Beispiel:

```
...
Split 3:<pb>
Split 4:'ein Text zwischen pb-Tags'
Split 5:<quote>
Split 6:'ein Text zwischen quote-Tags und pb-Tags'
```

```
Split 7:</quote>
Split 8:'ein Text der nicht in quote-Tags steckt, aber in
                                                        pb'
Split 9:</pb>
...
```

Im aufgeführten Beispiel ist zu sehen, dass sich Tags mit nicht-Tags in jedem Split fortlaufend abwechseln. Diese Eigenschaft wird ausgenutzt, indem boolesche Markierungen eingesetzt werden.

Die endgültige Verarbeitung des Textes findet in einer Schleife statt, die über jeden entstandenen Split iteriert. Bei jedem Schleifendurchlauf wird anhand der booleschen Markierung überprüft, ob es sich beim aktuellen Split um ein Tag oder nicht-Tag handelt. Falls es sich bei einem Split um ein Tag handelt, wird vorerst untersucht, ob es sich um ein einleitendes oder schließendes Tag handelt.

Falls es sich um ein einleitendes Tag handelt, wird dieses als offen markiert. Sobald als Pendant ein schließendes Tag auftaucht, wird die Markierung negiert:

```
...
if($tagname == 'note') {
  $tagwords_note_tag = TRUE;
}
...
```

und

```
...
if ($tagname[0] == '/') {
   $tagname = substr($tagname, 1);
   if($tagname == 'note') {
      $tagwords_note_tag = FALSE;
   }
...
```

Das Beispiel zeigt zunächst ein Tag namens `note`, das ein einleitendes Tag darstellt. In diesem Fall wird die Markierung `$tagwords_note_tag` auf `TRUE` gesetzt. Sobald ein schließendes `note`-Tag auftaucht, wird die Variable `$tagwords_note_tag` negiert. `pb`-Tags werden jedoch gesondert behandelt. Da jedes Wort eine Seite zugewiesen bekommt, wird bei einem einleitenden `pb`-Tag die Seitennummer aus der Zeichenkette, in Form von `pXXX`, extrahiert:

```
if($tagname == 'pb') {
  $tagwords_pb_tag = substr($value, 6, 15);
  $tagwords_pb_tag = str_replace('"', '', $tagwords_pb_tag);
}
```

In der if-Anweisung wird geprüft, ob das gerade verarbeitete Tag ein `pb`-Tag ist. Trifft diese Bedingung zu, so wird mit Hilfe der Funktion `substr()` zunächst der Zeichenkette des `pb`-Tags die Seitennummer entnommen. `str_replace()` beseitigt im Anschluss

überflüssige Anführungszeichen.

Sobald es sich bei einem Split um ein nicht-Tag beziehungsweise einen Text handelt, wird dieser Text wiederum mit der Funktion

```
teisearch_index_split($value)
```

gespalten. Daraus ergeben sich einzelne Wörter, die in einer weiteren inneren Schleife abgearbeitet werden und in einem Array, das die folgende Struktur aufweist, gespeichert werden:

```
$results = array(
  'numeric_word_id' => array('' => array(),),
  'page' => array('' => array(),),
  'searchtags' => array(
    'note' => array('' => array()),
    'quotesinnotes' => array('' => array(),),
    'quote' => array('' => array()),
    'orig' => array('' => array()),
    'reg' => array('' => array()),
    'corr' => array('' => array()),
    'sic' => array('' => array()),
    )
);
```

Wie aus der Struktur ersichtlich ist, können für jedes Wort die Seitennummer, ein eindeutiger Identifikator und Tags, falls gesetzt, zugeordnet werden. Das folgende Beispiel zeigt, wie die Speicherung eines Wortes stattfindet, das von einem `orig`-Tag umschlossen wird.

Die Zugehörigkeit dieses Wortes zu einem Tag, die Seitennummer sowie ein Identifikator werden gesetzt und im `$results`-Array gespeichert:

```
...
$results['page'][0][$word] = $tagwords_pb_tag;
if($tagwords_orig_tag) {
  $results['searchtags']['orig'][0][$word] ='orig';
  $results['numeric_word_id'][0][$word] = 7000;
}
...
```

Eine besondere Beachtung erfahren an dieser Stelle diejenigen Wörter, die mehrfach auf einer Seite vorkommen. Sobald ein Wort als Duplikat identifiziert wird, wird diesem ein Wert von `0.0001` zum bereits gesetzten Identifikator hinzu addiert. In dieser Modulversion ist die Erfassung aller Wörter die länger als zwei Zeichen sind nicht unbedingt notwendig. Der Index wurde jedoch von Anfang an so konzipiert, dass alle Wörter erfasst werden. Obwohl diese Eigenschaft des Index zum gegebenen Zeitpunkt nicht genutzt wird, ist es durchaus denkbar, dass dies in einer späteren Version beispielsweise für den Abgleich der Anzahl der erzeugten Textausschnitte gegenüber der Anzahl aller Suchtreffer dienen kann.

Der folgende Quelltextauszug zeigt, wie die Erfassung von Duplikaten abgewickelt wird:

```
...
if(isset($results['numeric_word_id'][0][$word])) {
  $duplicate_words[$word] += 1;
  $results['page'][$duplicate_words[$word]][$word] =
            $tagwords_pb_tag;
  ...
  if($tagwords_orig_tag) {
    $results['searchtags']['orig'][$duplicate_words[$word]]
        [$word] ='orig';
    $results['numeric_word_id'][$duplicate_words[$word]]
        [$word] = $duplicate_words[$word] + 0.0001;
  }
...
```

Zunächst wird im Array `$results` im Index des Identifikators an der Stelle `0` für das jeweilige Wort geprüft, ob dieses bereits im Array enthalten ist. Falls dies der Fall sein sollte, handelt es sich um ein Duplikat. Für dieses Wort wird der Zähler inkrementiert, das der Zählung jedes einzelnen Duplikates dient. Auf diese Art ist die genaue Anzahl jedes Wortes auf einer Seite bekannt. Im Anschluss wird dem Wort eine Seitennummer zugewiesen. Wenn das Wort zusätzlich von bestimmten Tags umschlossen wird, im Auszug des Quelltextes beispielsweise von `orig`-Tags, so wird diese Information im Array erfasst.

Des Weiteren wird zum bereits gesetzten Identifikator der Wert `0.0001` hinzu addiert. Hierzu ein Beispiel für das Wort `que`, das auf einer Seite mehrfach vorkommt:

1 Vorkommen: `numeric_word_id = 1`
2 Vorkommen: `numeric_word_id = numeric_word_id + 0.0001`
3 Vorkommen: `numeric_word_id = numeric_word_id + 0.0001`
...

Die auf diese Weise im Array `$results` erfassten Wörter werden anschließend in die Datenbanktabelle `teisearch_index` geschrieben:

```
db_merge('teisearch_index')
        ->key(array(
          'word' => $word,
          'nodenr' => $nodenr,
          'searchtags' => $searchtag,
          'page' => $results['page'][$i][$word],
          'numeric_word_id' => $numeric_word_id,
        ))
->execute();
```

An dieser Stelle kristallisierte sich in der Testphase des Indexiervorgangs heraus, dass beim Schreiben in die Datenbank in bestimmten Einzelfällen Ausnahmen oder so genannte Exceptions verursacht wurden. Aus einer Mitteilung ging hervor, dass während des Schreibens in die Tabelle `teisearch_index` Duplikate aufgetreten sind. Es erwies sich, dass die Ausnahme durch folgendes, wie auf der MySQL Referenzseite beschrieben, verursacht wurde:

> *„For example, if “e” and “é” have the same sort value in a given collation, they compare as equal.“*[34]

Bei der Ausnahme handelte es sich um die Zeichenketten `put` und `pût`.[18] Diese wurden bei der Überprüfung auf Duplikate nicht als Duplikate erkannt, da sie unterschiedliche Zeichenketten darstellen. Die restlichen Werte, bis auf die Zeichenkette selbst, waren für die Seitenzahl, Node-Nummer etc. identisch. Die MySQL-Datenbank macht jedoch, wie aus dem Zitat hervorgeht, keine Unterscheidung bei Zeichen, die in der gleichen Kollation vorliegen, hier zwischen `u` und `û`. Die Zeichenketten `put` und `pût` sind in diesen Fällen entbehrlich, solange mindestens eine davon in der Datenbanktabelle für die jeweilige Seite erfasst wurde. Der Grund liegt darin, dass das Erfassen beider Zeichenketten keinen Mehrwert bietet, da bereits eine der beiden Zeichenketten im Index für die gegebene Seitennummer existiert, bevor die Ausnahme ausgelöst wird. Aufgrund der Entbehrlichkeit dieser Zeichenketten wurde die Lösung des Problems, statt der Wahl einer anderen Kollation oder sonstige datenbankbezogene Änderungen, durch das Abfangen dieser Ausnahme umgesetzt, wobei nach dem Abfangen nichts weiter mit den Duplikaten geschieht.

An dieser Stelle ist die Erfassung beziehungsweise die Indexierung einer einzelnen Node für die `teisearch_index`-Tabelle abgeschlossen.

4.3.2 Indexierung einzelner Seiten ohne Sonderzeichen

Im Gegensatz zur Indexierung einzelner Wörter ist das Indexieren einzelner Seiten wesentlich handlicher. Hierbei muss nicht jedem einzelnen Wort ein Identifikator oder Tag zugeordnet werden, da diese Tabelle für das Auffinden ganzer Sätze genutzt wird. Es genügt die Wörter in Kleinbuchstaben zu konvertieren und von Sonderzeichen zu befreien. Jedoch wird die Textversion berücksichtigt. Der grobe Ablauf lässt sich folgendermaßen zusammenfassen:

➔ Es wird pro Node, jeweils für eine Seite und die Textversion eine Zeichenkette erzeugt, die keinerlei Tags und Sonderzeichen beinhaltet. Jeder Zeichenkette wird eine Node-Nummer, Textversion und Seitennummer zugeordnet und anschließend in die Datenbank geschrieben.

Die Funktion für die Indexierung einzelner Seiten, ohne Sonderzeichen, wird in der Funktion `teisearch_index()` aufgerufen durch

```
teisearch_dataset($nodenr, $text);
```

mit der Node-Nummer und dem Node-Text als Parameter. Der Text ist an dieser Stelle bereits von überflüssigen Daten wie `teiHeader` (vgl. [29]) bereinigt, enthält jedoch noch benötigte Tags und sämtliche Sonderzeichen.

In der Funktion `teisearch_dataset()` werden zunächst zwei Arrays mit den

18 Zur Erinnerung: Die Felder für Wörter wurden im Schema als 'varchar' definiert.

folgenden Strukturen definiert:

```
$tags = array(
  'orig' => '',
  'reg' => '',
  'corr' => '',
  'sic' => '',
);
$textversion = array(
  'orig_sic' => 'orig_sic',
  'reg_corr' => 'reg_corr',
);
```

Im Array `$tags` sind die für die Verarbeitung benötigten Tags definiert und im Array `$textversion` werden die beiden Textversionen `orig_sic` und `reg_corr` gesetzt.

Im Anschluss wird der Text, analog zum Abschnitt der Erfassung einzelner Wörter, an denjenigen Textstellen, an denen Tags vorkommen, jeweils aufgetrennt und in Splits zerlegt. Die eigentliche Erfassung der Texte erfolgt in einer zweifach geschachtelten Schleife. Eine äußere Schleife iteriert über die beiden Textversionen, während eine innere Schleife jeden Split bearbeitet:

```
foreach($textversion as $vesion => $tempversion) {
...
  foreach ($split as $value) {
...
  }
}
```

Für jede Schleifeniteration über die beiden Textversionen werden je nach Textversion die Tags aus dem Text entfernt, die für die aktuell bearbeitete Textversion überflüssig sind:

```
if($tempversion == 'orig_sic') {
  $tags = array(
    'orig' => '',
    'sic' => '',
    'pb' => '',
  );

  $temptext = preg_replace('#<reg>(.+?)\<\/reg>#s','',
                $temptext);
  $temptext = preg_replace('#<corr>(.+?)\<\/corr>#s','',
                  $temptext);
} else {
  $tags = array(
    'reg' => '',
    'corr' => '',
    'pb' => '',
  );
```

```
  $temptext = preg_replace('#<orig>(.+?)\<\/orig>#s','',
                  $temptext);
  $temptext = preg_replace('#<sic>(.+?)\<\/sic>#s', '',
                  $temptext);
}
```

Falls die Variable `$tempversion` den Wert `orig_sic` besitzt, werden die Tags `reg` und `corr` mit der Funktion `preg_replace()` mit Hilfe von regulären Ausdrücken entfernt. Dies geschieht gleichermaßen für die Textversion `reg_corr`.

Im Anschluss wird die innere Schleife zum Abarbeiten der einzelnen Splits betreten. Für jeden Split wird analog zur Indexierung einzelner Wörter geprüft, ob es sich dabei um ein Tag oder einen Text handelt. Hier ist als Tag nur ein Seitenumbruch in Form von `pb`-Tags interessant, da dieser die Seitenzahl beinhaltet.

Nachdem alle Splits abgearbeitet wurden, befinden sich die Resultate, die während der Schleifendurchläufe gesammelt wurden, im Array `$accum`, das zu jedem Textstück die Node-Nummer, Seitennummer sowie die Textversion enthält.

Die im Array `'$accum'` gesammelten Werte werden in einer Schleife, die über alle Seiten iteriert, in die Datenbank geschrieben:

```
foreach($accum as $page => $data) {
  if($page !== '') {
    db_insert('teisearch_dataset')
        ->fields(array(
          'page' => $page,
          'textversion' => $tempversion,
          'nodenr' => $nodenr,
          'data' => $data,
        ))
    ->execute();
  }
}
```

wobei die Drupal-Funktion `db_insert()` mit dem Tabellennamen als Parameter `teisearch_dataset` aufgerufen wird. Die einzelnen Felder werden für `page`, `nodenr` und `data` gefüllt, die dem Array `$accum` entnommen werden. `textversion` ist hierbei nicht in `$accum` enthalten. Da die äußere Schleife für beide Textversionen ausgeführt wird, musste dieser Wert nicht explizit im Array `$accum` gespeichert werden.

Sobald alle Nodes und deren beide Textversionen pro Seite erfasst wurden, ist der Indexiervorgang für einzelne Seiten ohne Sonderzeichen beendet.

4.3.3 Indexierung einzelner Seiten

Die Indexierung einzelner Seiten ist beinahe identisch zum vorherigen Abschnitt. Deshalb werden nur wesentliche Unterschiede aufgeführt.

Für diese Tabelle werden, ähnlich wie im Abschnitt zuvor, Texte pro Node-Nummer,

Seitenzahl und Textversion erfasst, jedoch ohne die Texte von Sonderzeichen zu bereinigen. Es müssen lediglich für die jeweils verarbeitete Textversion unnötige Tags entfernt werden.

Demnach besteht der Unterschied darin, dass die erzeugten Splits nicht zusätzlich in Wörter aufgetrennt werden müssen. Nach der Bereinigung von Tags wird pro Split, in dem ein Text einer Seite enthalten ist, in die Datenbanktabelle `teisearch_non_cleaned_up_dataset` geschrieben.

4.3.4 Löschung editierter Nodes

Die Funktion `teisearch_node_update()` leitet das Löschen von Nodes aus den Datenbanktabellen ein, die von einem Administrator editiert wurden. Das soll sicherstellen, dass der aktuelle Inhalt eines Nodes nach dem Löschvorgang mit Hilfe eines `cron`-Laufs, der durch den Administrator ausgelöst werden sollte, stets dem Index bekannt ist. Die Funktion `teisearch_node_update()` implementiert `hook_node_update()` folgender-maßen:

```
function teisearch_node_update($node) {
  db_delete('teisearch_index')
    ->condition('nodenr', ($node->nid))
    ->execute();
  db_delete('teisearch_dataset')
    ->condition('nodenr', ($node->nid))
    ->execute();
  db_delete('teisearch_non_cleaned_up_dataset')
    ->condition('nodenr', ($node->nid))
    ->execute();
  drupal_set_message('TEISearch module: Node update
      detected. Please make a cron run.');
}
```

Diese Funktion wird durch das Auslösen eines Speichervorgangs eines editierten Inhaltes aufgerufen. Hierfür wird der editierte Node an die Funktion:

```
teisearch_node_update()
```

übergeben. Die Funktion `db_delete()` leitet die Löschung der Datensätze für den gegebene Node ein.

4.4 Suche

4.4.1 Der TEISearch-Filter

Die *TEISearchbox* ist ein Suchformular, das als Filter für *TEI-Lite* kodierte Dokumente implementiert wurde.

In Drupal durchlaufen Inhalte, bevor sie dargestellt werden, eine Reihe von Filtern. Bei der Implementierung eines Moduls besteht die Möglichkeit, einen oder mehrere Filter für Textformate zu erzeugen [18]. Mit Hilfe von Filtern werden auch die TEI kodierten Inhalte vor der endgültigen Darstellung bearbeitet.

Filter kommen in einer bestimmten Reihenfolge zum Einsatz. Dazu müssen sie aktiviert werden. Hinzu kommt eine Gewichtung für jeden Filter, die die Reihenfolge zur Ausführung der einzelnen Filter festlegt.

TEI-Lite kodierte Dokumente, die vom *TEIChi-Modul* in Drupal eingebunden werden, besitzen das Textformat `teichinode`, das im `teichi.module` definiert wird. Um diese Textformate mit dem *XSL-Stylesheet*, das aus der Projektarbeit hervorgegangen ist, darzustellen, muss der Filter *XSL filter for TEIChi module* aktiviert sein.

Das `teisearch.module` stellt einen weiteren Filter für diese Inhalte bereit und ist im administrativen Menü in Drupal nach der Aktivierung des *TEISearch-Moduls* zunächst für alle Textformate verfügbar. Dieser Filter ist lediglich für TEI kodierte Dokumente nutzbar, die mit dem dazu entsprechenden *XSL-Stylesheet* transformiert wurden. An dieser Stelle ist die Reihenfolge der Filter, die Veränderungen an TEI kodierten Inhalten vornehmen, von großer Bedeutung. Das *XSL-Stylesheet* enthält eine eindeutige Zeichenkette in Form eines Tags:

```
<form id="teisearch"></form>
```

Nach der Transformation eines TEI kodierten Inhalts, mit Hilfe des *XSL filter for TEIChi module*−Filters, entsteht der folgende Eintrag, der im HTML-Quellcode aufzufinden ist :

```
<form id="teisearch"/>
```

Sobald dieser Eintrag im HTML-Quellcode erscheint und der *TEISearch-Filter* aktiv ist, beginnt das `teisearch.module` mit dem Filtern dieser Inhalte.

In der Funktion `teisearch_filter_info()` wird `hook_filter_info()` implementiert und hat den folgenden Inhalt:

```
function teisearch_filter_info() {
   $filters['teisearch'] = array(
      'title' => t('TEISearch Filter'),
      'process callback' =>
                        '_teisearch_tags_filter_process',
      'tips callback'  => '_teisearch_filter_tips',
      'weight' => 20,
   );
  return $filters;
```

```
}
```

Ein Filtername mit dem Inhalt *TEISearch Filter*, eine Funktion zum Verarbeiten `_teisearch_tags_filter_process`, eine Funktion für Filterdetails sowie eine Gewichtung für den Filter sind in dieser Funktion definiert. Sobald ein Node aufgerufen wird, der einen `teichinode` Inhalt vorweist und der *TEISearch-Filter* aktiv ist, wird diese Funktion aufgerufen.

Die Funktion, die mit der tatsächlichen Filterung der Inhalte eingeleitet wird, ist die `_teisearch_tags_filter_process($text,$filter,$format, $langcode="").` Diese bekommt den zu bearbeitenden Text, der durch den Filter *XSL filter for TEIChi module* vorgefiltert wurde, mit der Variable `$text` sowie das Filter Objekt, Format Objekt und lokalisationsspezifische Parameter übergeben.

Zunächst wird eine CSS-Datei, die bereits während der Aktivierung des Moduls erzeugt wurde, eingebunden. Diese Datei legt Aussehen und Position des zu erzeugenden Formulars fest:

```
drupal_add_css($cssPath, $options);
```

wobei `$cssPath` den Pfad zur Datei repräsentiert und `$options` den Typ des Inhalts, in dem Fall eine CSS-Datei. Der Rückgabewert der Funktion `'_teisearch_tags_ filter_process()'` ist die Ersetzung der `<form id="teisearch"/>`-Zeichenkette, im Text des Nodes, durch ein von Drupal gerendertes Suchformular, das die *TEISearchbox* darstellt:

```
return str_replace('<form id="teisearch"/>',
                            $teisearchRenderedForm, $text);
```

Demnach sorgt der vom Modul bereitgestellte Filter für die Erzeugung des Suchformulars sowie die Hervorhebung der gefundenen Treffer auf einem Node. Die konkrete Implementierung des Suchformulars *TEISearchbox* und des Vorgangs zum Hervorheben von Treffern ist in den folgenden drei Kapiteln beschrieben.

4.4.2 Die ursprüngliche TEISearchbox

Zunächst wurden für das Suchformular, der sogenannten *TEISearchbox*, neben einem Eingabefeld und Knopf zum Bestätigen einer Suchanfrage, ein ausklappbares Menü zugeführt.

Dieses Menü dient der Option, eine Suche auf dem aktuellen Node vorzunehmen und die Suchtreffer direkt, ohne einer Weiterleitung zur Ergebnisseite, anzuzeigen. Die Option, die das ausklappbare Menü bereitstellt, wurde nach einer Rücksprache mit dem Projektinitiator entfernt. Im Quelltext ist diese Option weiterhin vorhanden und kann bei Bedarf reaktiviert werden. Aus dem Grund wird hier näher auf diese Funktionalität eingegangen, da sie in der Umsetzung relativ komplex war.

Die *TEISearchbox* wird mit dem Aufruf der Funktion

```
drupal_render(drupal_get_form('teisearch_searchbox_form'));
```

erzeugt. Die Funktion `drupal_get_form()` bekommt ein Formular als Rückgabewert, das in der Funktion `teisearch_searchbox_form()` erzeugt wird. Das Formular besitzt ein Eingabefeld, eine Dropdown-Liste und einen Such-Knopf. Hier als Beispiel für die Definition des Textfeldes und der Dropdown-Liste:

```
$form['teisearchfield']['searchfield'] = array(
  '#type' => 'textfield',
  '#id' => 'searchfield',
  '#size' => 25,
  '#maxlength' => 40,
  );

$form['teisearchfield']['selectbox'] = array(
  '#type' => 'select',
  '#options' => array(
  1 => variable_get('teisearch_first_select_value', 'This
                  Chapter'),
  2 => variable_get('teisearch_second_select_value', 'All
                  Chapters'),
  ),
);
```

Das Eingabefeld für Suchbegriffe ist `25` Zeichen lang und darf maximal `40` Zeichen aufnehmen. Falls eine Zeichenkette eingegeben wird, die die Länge von `40` Zeichen überschreitet, wird diese auf den definierten Maximalwert gekürzt. Die Dropdown-Liste stellt zwei Optionen bereit, zum einen das Durchsuchen eines aktuell aufgerufenen Nodes und zum anderen das Durchsuchen aller Nodes. Mit der Betätigung des Suchknopfs wird die Funktion:

```
teisearch_searchbox_form_submit()
```

aufgerufen, mit den Parametern `$form` und `&$form_state`. Der Parameter `&$form_state` beinhaltet beispielsweise den Wert, der in das Eingabefeld eingetragen wurde sowie die gesetzte Option der Dropdown-Liste. Im Anschluss beginnt die Auswertung dieser Werte.

Falls beim Bestätigen des Formulars durch den Suchknopf das Eingabefeld leer ist, wird eine Meldung mit dem Hinweis ausgegeben, dass ein Wort für die Suche eingeben werden soll. Falls das eingegebene Wort weniger als drei Zeichen lang ist, wird wiederum ein Hinweis mit einer Meldung produziert, der darauf aufmerksam macht und die weitere Ausführung der Funktion abbricht.

Wenn eine Zeichenkette eingegeben wird, die länger als zwei Zeichen ist, beginnt der Suchvorgang. Zunächst wird geprüft, welche Art von Suche durchgeführt werden soll. Falls in der Dropdown-Liste die Option zum Durchsuchen des aktuellen Kapitels, das dem voreingestellten Wert entspricht, gewählt wurde, untersucht die Funktion, welche Art von Zeichenkette eingetragen wurde. Hierbei wird zwischen einem Satz und einem einzelnen Wort unterschieden. Dazu wird die Zeichenkette auf die folgende Weise ausgewertet:

```
...
$keys = teisearch_trim_search_keys($keys);
$phrase = FALSE;
$phrase = strpos($keys, ',');
$keys = str_replace(',', ' ', $keys);
if($phrase) {
  ...
}
...
```

Die Variable `$keys`, die den übergebenen Suchschlüssel beinhaltet, wird mit der Funktion `teisearch_trim_search_keys()` normiert. Normieren bedeutet, den Suchschlüssel auf die gleich Art wie beim Indexieren zu bearbeiten. Es werden sämtliche Sonderzeichen und überflüssige Leerzeichen entfernt. Falls mehrere Wörter im Suchschlüssel enthalten sind, werden diese durch ein Komma getrennt.

Die Variable `$phrase` hat zunächst den Wert `FALSE`. Mit der Funktion `strpos()` wird das erste Vorkommen von einem Komma in der Variable `$keys` überprüft. Demnach kann die Variable den Wert `FALSE`, eine Position des ersten Vorkommens oder eine leere Zeichenkette enthalten. Die Bedingung `if($phrase)` ist also nur dann Wahr, wenn ein Komma tatsächlich in der Zeichenkette enthalten ist.

Diese Unterscheidung nach Satz und nicht-Satz ist wichtig, denn um einen zusammenhängenden Satz in der Datenbank ermitteln zu können, wird eine andere Tabelle zum Durchsuchen verwendet, als bei einzelnen Wörtern.

Wenn also die Bedingung für einen Satz zutrifft, sieht der Ablauf folgendermaßen aus:

```
$found = teisearch_find_phrase($keys);
if(!empty($found)) {
  ...
} else {
$found = teisearch_find_word($keys);
if(!empty($found)) {
  ...
}
$found = teisearch_find_word_like($keys);
if(!empty($found)) {
  ...
}
```

An dieser Stelle ist zu sehen, dass drei verschiedene Funktionen aufgerufen werden, die jeweils als Parameter den Suchschlüssel bekommen.

Wenn eine dieser Funktionen ein nicht leeres Objekt liefert, das in der Variable `$found` gespeichert ist, wird eine der `if(!empty($found))` Bedingungen erfüllt. Falls alle Funktionen ein leeres Objekt liefern, wird der Hinweis ausgegeben, dass die Suche keine Treffer ergab.

Dem Objekt `$found`, das die Suchtreffer darstellt, wird lediglich der erste Treffer entnommen. Daraus wird ein Link gebildet, der die Node-Nummer des aktuellen Kapitels, die Seite auf dem sich der Treffer befindet und einen Text zum Hervorheben – den unbearbeiteten Suchschlüssel – beinhaltet. Hierzu ein Beispiel für solch einen Link mit

einem Suchschlüssel `que`, der Node-Nummer `15` und der Seite `p69` auf einem lokalen Webserver:

```
http://localhost/drupal/?q=node/15&highlight=que#p69
```

Wie man sieht, wird an den Pfad ein Parameter `highlight` angefügt. Dieser hat den Zweck, den gesuchten Schlüssel zum Hervorheben zu übergeben. Im Anschluss wird man über diesen Link zum gefundenen Treffer umgeleitet.

Kurz zur Erinnerung: das passiert nur dann, wenn in der Dropdown-Liste die Option zur Suche auf dem aktuellen Node gewählt wurde. Falls die Option zur Suche auf allen Nodes gesetzt wurde, erfolgt eine Umleitung zur Trefferseite. Hierzu wiederum ein Beispiel für einen Link dieser Art:

```
http://localhost/drupal/?q=teisearch/keys=que_offset=[10]
```

In solch einem Link können jedoch noch weitere Parameter enthalten sein.[19]

Damit ist vorerst der Suchvorgang für den aktuellen Node abgeschlossen.

4.4.3 Die neue TEISearchbox

Das neue beziehungsweise an die Bedürfnisse des Projektinitiators angepasste Suchformular ähnelt optisch der ursprünglichen *TEISearchbox*. Das Suchformular heißt weiterhin *TEISearchbox*, beinhaltet jedoch einige Änderungen bezüglich der Funktionsweise. Im Gegensatz zur ursprünglichen *TEISearchbox* wurde die Option zur Suche auf dem aktuellen Node abgestellt und bei Suchanfrage erfolgt eine sofortige Weiterleitung zur Ergebnisseite. Des Weiteren hat diese einen Reset-Knopf erhalten, mit dem sich die Suche zurücksetzen lässt. Zurücksetzen bedeutet in diesem Zusammenhang, dass alle hervorgehobenen Treffer in den ursprünglichen Zustand gebracht werden.

Demnach hat die neue *TEISearchbox* zunächst die Aufgabe, beim Betätigen des Suchknopfs einen Link mit dem Suchschlüssel und Offset-Wert als Parameter für die Weiterleitung zu generieren und an diesen umzuleiten. Des Weiteren sorgt der Reset-Knopf für eine Weiterleitung zum aktuellen Node und entfernt dabei aus der URL alle Parameter.

4.4.4 Hervorhebung von Suchtreffern

Die Markierung eines Treffers übernimmt die Funktion des *TEISearch-Filters*. Denn diese verarbeitet, neben der Erzeugung der *TEISearchbox*, auch die Hervorhebungen der Suchtreffer. In der Funktion `_teisearch_tags_filter_process()` wird mit:

```
if(isset($_REQUEST['highlight'])) {
...
```

19 Diese werden im Kap. 4.4.5 näher betrachtet.

```
}
```

wobei mit `$_REQUEST['highlight'])` der Inhalt des assoziativen Arrays für `highlight` ist, geprüft, ob dieser Wert gesetzt ist. Falls dies zutrifft, wird der Filtervorgang zum Hervorheben der Suchtreffer angestoßen.

Eine Besonderheit des Hervorhebens stellt die Umwandlung der Zeichenkette, die hervorgehoben werden soll, dar. Wie bereits beschrieben, kommt der *TEISearch Filter* erst zum Einsatz, nachdem der Filter für `teichinode`-Textformate des *TEIChi-Moduls*, das die XML-Dokumente nach HTML transformiert hat. Dies bedeutet, dass die Zeichenkette, nach der gesucht werden soll, auf genau die gleiche Weise erzeugt werden muss, wie beim Transformieren der XML-Dokumente. Zur Verdeutlichung dient das folgende Beispiel:
Aus der Zeichenkette

...Le lendemain le soleil s'étant levé...

wird nach der Transformation mittels des *TEIChi-Moduls*

...Le lendemain le soleil s'étant levé... (Vgl. [12])

Um aus solch einer Zeichenkette, die in das Suchformular eingetragen wurde, eine identische Zeichenkette zu bilden, dient der Abschnitt aus der Funktion `_teisearch_tags_filter_process()`:

```
$dom = new DomDocument;
$dom->loadXML($transformkeys);
$highlightwords = $dom->saveXML();
```

Die Variable `$dom` dient zunächst der Zwischenspeicherung des transformierten Suchschlüssels. Mit `$dom->saveXML()` wird der Variable `$highlightwords` die transformierte Zeichenkette zugewiesen. Damit entsteht eine Zeichenkette, deren Erzeugung analog zur Transformation des Dokuments durch den Filter des *TEIChi-Moduls* verläuft. Somit ist sichergestellt, dass falls die Suche einen Treffer ergibt, diese Zeichenkette im Text des Dokuments vorkommt und die Hervorhebung der Treffer vorgenommen werden kann.

Eine Ausnahme stellt jedoch das Auftreten von Tags im Text dar. Bei der Indexierung der zu hervorhebenden Texte werden keinerlei Tags erfasst. Dies hat für die Hervorhebung von Suchtreffern unangenehme Nebeneffekte. Der gesuchte Schlüssel wird zwar gefunden und die Weiterleitung zum Treffer funktioniert tadellos, jedoch kann solch ein Treffer in dieser Modulversion nicht markiert werden. Hierzu ein Beispiel:

Der Suchschlüssel lautet:

'assez épais, semblait annoncer'

Der transformierte Suchschlüssel lautet:

'assez épais, semblait annoncer'

Der transformierte Text eines TEI kodierten Dokuments jedoch:

...assez épais, sembloitsemblait annoncer...

Der Versuch, dieses Problem im Rahmen der Arbeit zu lösen scheiterte, da sich herausstellte, dass das zugrundeliegende Problem deutlich komplexer ist als zunächst angenommen.

Das Grundproblem besteht darin, dass bei einem gegebenen Suchschlüssel nicht ohne weiteres festgestellt werden kann, ob dessen Pendant im Text derartige Tags enthält. Das bedeutet, wie im oben gezeigten Beispiel, dass mit den Mitteln, die die Ersetzung von Zeichenketten bereitstellt, die Gefahr besteht, dass eine Schachtelung von Tags zum Beispiel `<span class="choice reg">semblait</span>` zerstört werden kann und ein HTML-Dokument nach einer nachlässigen Ersetzung nicht mehr valide ist.

Demnach müssen die Ersetzungen sehr akkurat vorgenommen werden und keinesfalls die HTML-Struktur beschädigen. Nach dieser Feststellung wurden weitere Versuche das Problem zu beheben eingestellt, da wesentliche Funktionalitäten der Suche noch nicht implementiert waren und der zeitliche Rahmen für die Fertigstellung des Moduls sehr schmal wurde.

Ein Beispiel für eine unglückliche Bearbeitung eines Textes hinsichtlich der Verschachtelung könnte folgendermaßen aussehen, wobei in diesem Beispiel konkret die Zeichenkette `'le costume des habillemens'` hervorgehoben werden soll:

Vor der Ersetzung:

```
...le costume des <span class="choice orig">habillemens
</span>
<span class="choice reg">habillements</span>...
```

Nach der Ersetzung:

```
...<span class='teihighlight'>le costume des habillemens
</span></span>
<span class="choice reg">habillements</span>...
```

Das Beispiel zeigt, wie ein nachlässiges Löschen der Zeichenkette `<span class= "choice orig">` aussehen kann, die ein einleitendes `span`-Tag darstellt, ohne das dazugehörige schließende `span`-Tag zu entfernen, das die Zerstörung der Schachtelung verursacht.

4.4.5 Das Suchformular der Ergebnisseite

Die Ergebnisseite dient der Präsentation von Suchtreffern aller Nodes, die im `teichinode`-Textformat vorliegen. Auf diese Seite wird ein Benutzer umgeleitet, nachdem der Search-Knopf der *TEISearchbox* betätigt wurde und das Eingabefeld einen gültigen Suchschlüssel beinhaltete.

Auf der Ergebnisseite erscheint neben den Suchtreffern – falls die Suche Treffer hervorbrachte – ein Suchformular. Dieses Suchformular bietet neben einem Eingabefeld für Suchanfragen, einen Knopf zum Absenden einer weiteren Suchanfrage.

Des Weiteren ist dieses Formular ausklappbar gestaltet. Beim Aufklappen kommen weitere Suchoptionen zum Vorschein, die zur Verfeinerung der Suche dienen. Hier bietet sich die Möglichkeit, die Suche auf bestimmte Textbereiche einzuschränken und zum Beispiel nur in Anmerkungen des Textautors zu suchen. Die Optionen sind als Kontrollkästchen im Formular aufgeführt, die durch einen Klick ausgewählt werden können.

Im wesentlichen unterscheidet sich das Formular, im Gegensatz zu der *TEISearchbox*, durch das ausklappbaren Menü. Hierzu ein Auszug aus der Funktion `teisearch_search_form()`, die das Suchformular erzeugt:

```
$form['teisearch']['teisearcheckboxes'] = array(
    '#type' => 'fieldset',
    '#title' => t('TEISearch search options'),
    '#collapsible' => TRUE,
    '#collapsed' => TRUE,
    '#name' => 'TEISearch search options',
    '#id' => 'teisearcheckboxes',
    '#attributes' => array('class' =>
                           array('teisearcheckboxes'),
    ),
  );

...

$form['teisearch']['teisearcheckboxes']['checkboxes'] =
array(
  '#type' => 'checkboxes',
  '#options' => array(
    'quotes' => variable_get('teisearch_checkbox_quotes',
                'Search in quotes'),
    'note' => variable_get('teisearch_checkbox_notes',
            'Search in notes'),
  ...
  ),
);
```

Zunächst wird in `$form['teisearch']['teisearcheckboxes']` ein Array mit den Eigenschaften des Formulars definiert. Als Typ des Formulars wird `fieldset` gesetzt und mit `'#collapsible' => TRUE` und `'#collapsed' => TRUE` sichergestellt, dass das Formular ausklappbar ist und erst nach einem Klick aufgeklappt

wird.

In `$form['teisearch']['teisearchcheckboxes']['checkboxes']` wird ein Typ `checkboxes` für das Formular gesetzt. Im Array `#options` werden die einzelnen Kontrollkästchen definiert. Beispielsweise bekommt das TEI-Tag `note` einen voreingestellten Text `Search in notes`, der neben dem Kästchen erscheint, falls im administrativen Menü, auf der Konfigurationsseite des *TEISearch-Moduls*, kein anderer Wert gesetzt wurde.

Nachdem der Suchknopf betätigt wurde, beginnt die Verarbeitung des Formulars in der Funktion `teisearch_search_form_submit()`. Zunächst prüft die Funktion, ob ein oder mehrere Kontrollkästchen aktiviert wurden und sammelt diese Informationen. Anschließend wird geprüft, ob valide Werte in das Suchformular eingetragen wurden. Valide bedeutet, dass eine Zeichenkette die Länge von zwei Zeichen überschreitet. Falls dies nicht der Fall sein sollte, wird ein Hinweis ausgegeben. Nachdem dies verifiziert und als valide bewertet wurde, erfolgt eine Weiterleitung auf die selbe Seite, jedoch mit einem Übergabewert im Pfad, der den Suchschlüssel, Suchtags und ein initiales Offset von `10` enthält:

```
drupal_goto(urldecode('teisearch/'.
                        $keys.searchtags.'_offset=[10]'))
```

wobei `drupal_goto()` eine Weiterleitung an die URL einleitet, die die Funktion `urldecode()` mit dem übergebenen Parametern, dem Schlüssel, den Suchtags und Offset liefert. Der tatsächliche Suchvorgang wird von der Funktion `teisearch_teiview()` eingeleitet, die im nächsten Abschnitt behandelt wird.

Nach einer Rücksprache mit dem Projektinitiator, wurde dem Suchformular ein weiteres Feld hinzugefügt. Dieses bietet die Möglichkeit, dem Benutzer einen informativen Text zu präsentieren. Dieser Text kann im administrativen Bereich von Drupal, auf der Konfiguratientsseite des *TEISearch-Moduls* festgelegt werden und ist folgendermaßen in der Funktion `'teisearch_search_form()'` implementiert:

```
$form['teisearch']['teisearch_infofield'] = array(
  '#type' => 'item',
  '#name' => 'TEISearch Info',
  '#title' => variable_get('teisearch_info_item', 'Please
                edit this text in TEISearch Preferences'),
  '#id' => 'teisearch',
  '#attributes' => array('class' => array('teisearchfield'),
  ),
);
```

4.4.6 Der Suchvorgang

In der Funktion `teisearch_teiview()` wird der Suchvorgang eingeleitet. Der Suchvorgang beginnt nach der Auswertung der übergebenen Parameter. Die Auswertung verläuft folgendermaßen:

```
$keys = urldecode($_REQUEST['q']);
```

Zunächst werden der Variable `$keys` alle übergebenen Parameter, die in `$_REQUEST` enthalten sind, zugewiesen. `$_REQUEST['q']` ist hierbei ein assoziatives Array, dessen Query-Abschnitt im Index `q` enthalten ist. Dort befinden sich die übergebenen Parameter, unter anderem die zu suchende Zeichenkette sowie die Suchoptionen. Die daraus gewonnene Zeichenkette wird mit Hilfe von PHP-Funktionen für Zeichenketten nach Tags, Suchschlüssel und Offset zerlegt und den Variablen `$keys`, `$tags` und `$offset` zugewiesen. Durch den Aufruf der Funktion `teisearch_execute()` beginnt der Suchvorgang:

```
$results = array();
if ($keys) {
  $results = teisearch_execute($keys, $keys_to_highlight,
                                        $tags, $offset);
}
```

Die Variable `$keys_to_highlight` ist in diesem Fall gleich dem Wert der Variable `$keys`. Die Gleichheit der beiden Variablen hat an dieser Stelle den Grund, dass die Variable `$keys` für die Datenbankabfrage noch nicht normiert wurde. Der Rückgabewert dieser Funktion ist ein Array, das alle Funde beinhaltet und im Array `$results` gespeichert wird.

An dieser Stelle wird naiv vorausgesetzt, dass der Benutzer, der den Suchschlüssel in das Suchformular eingibt, auf die Groß- und Kleinschreibung achtet, da die Hervorhebung von Suchtreffern mit der in `$keys_to_highlight` gespeicherten Zeichenkette erfolgt.

Da sich der gesamte Vorgang des Hervorhebens im Laufe der Implementierung als komplex erwies und erst spät als Feature aufgenommen wurde, weist dieser in der aktuellen Version des Moduls geringe Mängel auf, die nicht alle im gegebenen Zeitrahmen behoben werden konnten. Es existieren genau zwei Mängel. Einerseits besteht das bereits oben ausgeführte Problem, dass die Hervorhebung von Zeichenketten, die Tags enthalten, nicht möglich ist. Andererseits ist weiterhin die Problematik der Groß- und Kleinschreibung vorhanden.

In der Funktion `teisearch_execute()` werden zunächst die Suchschlüssel normiert, da die Daten, die während der Indexierung erfasst wurden, Texte ohne Sonderzeichen darstellen. Das geschieht mit Hilfe der Funktion `teisearch_trim_search_keys()`. Diese liefert als Rückgabewert eine bereinigte Zeichenkette, die einzelne Wörter durch Kommata trennt. Die Funktion `strpos($keys, ',')` stellt die erste Position eines Vorkommens von einem Komma fest, falls vorhanden und liefert als Rückgabewert entweder einen Integer-Wert, der die Position des Vorkommens darstellt oder `FALSE`, `0` beziehungsweise eine leere Zeichenkette. Demnach kann auf die folgende Art überprüft werden, ob es sich bei der gegebenen Zeichenkette um einen Satz handelt:

```
$phrase = strpos($keys, ',');
$keys = str_replace(',', ' ', $keys);
if($phrase) {
  …
}
```

`if($phrase)` wird nur dann Wahr, wenn die Variable `$phrase` einen Integer-Wert beinhaltet. Diese Überprüfung ist wichtig, da die Datenbankabfragen für Sätze und einzelne Wörter unterschiedlich gestaltet sind und die abzurufenden Datensätze nicht in der selben Tabelle vorliegen.[20]

Die Suchanfragen finden demnach in Abhängigkeit von der Art der zu suchenden Zeichenketten statt. Wenn ein Satz gegeben ist, wird zunächst die Funktion `teisearch_find_phrase()` ausgeführt, die als Parameter den Suchschlüssel und Tags benötigt.

Falls ein oder mehrere Tags im Suchformular ausgewählt wurden, bricht diese Funktion mit einer leeren Zeichenkette als Rückgabewert ab. Wenn keine Tags gesetzt wurden, ist der Rückgabewert dieser Funktion bei erfolgreicher Suche ein Objekt, das alle erzielten Treffer der Datenbankabfrage aus der `teisearch_dataset`-Tabelle beinhaltet. Liefert die Funktion `teisearch_find_phrase()` ein leeres Objekt, wird mit der Ausführung der Funktion `teisearch_find_words()` fortgefahren, die in der `teisearch_index`-Tabelle jedes einzelne Vorkommen aller Wörter ermittelt und ein Objekt mit den Resultaten zurückgibt.

Wenn es sich bei der gegebenen Zeichenkette um ein einzelnes Wort handelt, wird zunächst die Funktion `teisearch_find_word()` aufgerufen. Hierzu ein Auszug aus dieser Funktion:

```
$tags = str_replace(',', "' OR searchtags = '", " AND
                                (searchtags = '".$tags."' ");
$tags = str_replace("searchtags = 'quote'", "searchtags =
          'quotesinnotes' OR searchtags = 'quote'", $tags);

$tags = str_replace("searchtags = 'note'", "searchtags =
            'note' OR searchtags = 'quotesinnotes'", $tags);
$find = db_query("SELECT word, nodenr, page, searchtags FROM
        {teisearch_index} WHERE word = '$keys' $tags) ORDER
          BY CHAR_LENGTH(page) ASC, page ASC")->fetchAll();
```

Der Auszug zeigt zunächst, wie eine Zeichenkette für Tags gebildet wird, die dem Datenbankabruf hinzugefügt wird. Falls es sich bei der an die Funktion übergebenen Zeichenkette `$tags` um mehrere Tags handelt, sind diese durch Kommata getrennt. Mit Hilfe von `str_replace()` werden Kommata durch `OR` ersetzt. Im Anschluss wird `searchtags = 'quote'` durch `searchtags = 'quotesinnotes' OR searchtags = 'quote'` ersetzt. Das passiert aus dem Grund, da das Tag `quote` zusätzlich in einer Notiz vorkommen kann. Ähnliches wird für das Tag `note` vorgenommen. Als Beispiel sei angenommen, es werde nach dem Wort „comme“ gesucht, für das die beiden Tags `note` und `corr` gesetzt wurden, dann sehe die Zeichenkette, die der Funktion `db_query()` übergeben würde wie folgt aus:

```
SELECT word, nodenr, page, searchtags FROM {teisearch_index}
WHERE word = 'comme' AND (searchtags = 'note' OR searchtags =
'quotesinnotes' OR searchtags = 'corr' ) ORDER BY CHAR_LENGTH
(page) ASC, page ASC
```

20 Für ausführliche Informationen siehe Kap. 4.3

Falls diese ein leeres Resultat liefert, wird die Funktion `teisearch_find_word_like()` aufgerufen. Der Unterschied der beiden Funktionen liegt darin, dass in `teisearch_find_word()` exakt nach dem vorgegebenen Schlüssel gesucht wird und in `teisearch_find_word_like()` nach Wörtern in Art von `%$key%`[21].

Falls keine dieser Funktionen einen Suchtreffer erzielt, wird ein Hinweis ausgegeben, dass die Suche erfolglos war. Wurden Suchtreffer erzielt, beginnen die Vorbereitungen zur Darstellung der Suchtreffer.

Vorerst wird ein Anfangs-Offset gesetzt, das als Markierung für den Anfang der Trefferausgabe dient. Das Ende des Offsets ist gleich dem als Parameter übergebenen Offset-Wert. Ein Zähler für die Suchtreffer ohne Duplikate, `$count_item`, wird definiert und zunächst mit `0` initialisiert. Weiterhin wird ein Array `$duplicate_page` zum Sammeln von Duplikaten, die für die Ausgabe irrelevant sind, definiert.

Als irrelevante Treffer werden diejenigen gewertet, die auf der gleichen Seite mit der gleichen Node-Nummer versehen und somit hinsichtlich dieser Kriterien Duplikate sind. Die Begründung dazu liegt in der Tatsache, dass der Sprung nur zur gewünschten Seite möglich ist, auf dem sich der Suchtreffer befindet, da zum gegebenen Zeitpunkt ausschließlich die Seitennummern als Ankerpunkte dienen, die den Sprung ermöglichen.

Da ein Zähler und ein Array für Duplikate definiert wurde, werden diese in einer Schleife, die über alle Suchtreffer iteriert, hochgezählt beziehungsweise mit Duplikaten gefüllt.

Falls es sich um ein nicht-Duplikat handelt, wird der Zähler für Treffer ohne Duplikate inkrementiert und das Element mit dem Suchtreffer in ein Array `$results` aufgenommen. An dieser Stelle wird wiederum eine Unterscheidung zwischen Sätzen und einzelnen Wörtern gemacht, da das `$results`-Array Wörter zum Hervorheben beinhaltet. Hier nun ein Beispiel für ein einzelnes Element, dessen Suchschlüssel ein Satz darstellt:

```
$results[] = array(
  'link' => url('node/' . $item->nodenr, array('fragment'
      => $item->page, 'query' =>
          array('highlight' =>
          '['.  $keys_to_highlight.']'))),
  'page' => $item->page,
  'title' => $node->title,
  'nodenr' => $item->nodenr,
  'textsnippet' => teisearch_snippets_for_results(
      $keys_to_highlight,$item->nodenr, $item->page),
);
```

Wie man erkennen kann, wird zu jedem Element im Array ein Link, Seite, Titel, Node-Nummer und Textauszug gespeichert. Die Erzeugung eines Textauszugs lässt sich wie folgt als Algorithmus zusammenfassen:

1. prüfe in der Datenbanktabelle `teisearch_non_cleaned_up_dataset`, ob die gesuchte Zeichenkette vohanden ist
2. falls vorhanden, fahre fort mit Erzeugung des Textausschnitts mit Hilfe des Texts, in

21 Beispielsweise im Fall $keys = 'que' → %que% wären auch die Wörter 'quelque', 'quelle', 'unique' etc. Suchtreffer.

dem die gesuchte Zeichenkette vorliegt
3. falls nicht vorhanden, bereinige den Suchschlüssel von Sonderzeichen und beziehe den Text für die Bildung des Textausschnitts aus der Datenbanktabelle `teisearch_dataset`

Das Erzeugen von Textauszügen birgt einige Tücken. Die beschriebenen Texte, die der Erzeugung dienen, werden an bestimmten Stellen abgeschnitten. Dabei kann es passieren, dass nach dem Beschneiden der Zeichenkette, die in UTF-8 kodiert ist, einzelne Zeichen zerstört werden. Um das zu umgehen wird nach jedem erzeugten Textausschnitt geprüft, ob dieser zerstörte Zeichen enthält, die zugleich beseitigt werden.[22]

Wenn alle Elemente der Schleife abgearbeitet wurden, beginnt die Erzeugung einer Navigation für die Ergebnisse, auch Pager genannt, für das aktuelle Offset. Da alle Resultate, die keine Duplikate darstellen, mitgezählt wurden, ist die genaue Anzahl dieser Elemente bekannt. Somit kann ein Pager erstellt werden, der zu Seiten mit jeweils zehn Treffern einen Link bereitstellt.

Dieser Pager wird nur dann angelegt, wenn die Suche insgesamt mehr als zehn Treffer ergab, wobei Duplikate vernachlässigt werden. Der Variable `$pageroffset` wird eine Zeichenkette zugewiesen, die auf folgende Weise gebildet wird:

```
$pageroffset = $first.$prev.$pageroffset.$next.$last;
```

Der Wert von `$pageroffset` kann den folgenden Inhalt haben:

```
<b><a href="/drupal/?
q=teisearch/keys=comme_offset=[10]">1</a></b>
<a href="/drupal/?q=teisearch/keys=comme_offset=[20]">2</a>
<a href="/drupal/?q=teisearch/keys=comme_offset=[30]">3</a>
<a href="/drupal/?q=teisearch/keys=comme_offset=[20]">
<u>next ></u>
</a> <a href="/drupal/?q=teisearch/keys=comme_offset=[30]">
<u>last >>
</u> </a>
```

vorausgesetzt die Suche nach dem Wort `comme` liefert für diesen konkreten Fall zwischen `20` und `30` Suchtreffer. Hier ist das Offset der ersten zehn Treffer fett gedruckt. Dies passiert genau dann, wenn man auf der Ergebnisseite die ersten zehn Treffer betrachtet beziehungsweise sich im Offset der ersten 10 Treffer befindet. Demnach wird immer die Seite mit dem aktuellen Offset durch `<b>...</b>` markiert.

Nach Abarbeitung der Suche und der Bildung des Arrays mit den Resultaten, sieht der Rückgabewert der Funktion `teisearch_execute()` wie folgt aus:

```
return array(
    '#theme' => 'teisearch_results',
    '#results' => $results,
    '#keys' => $keys,
    '#pageroffset' => $pageroffset,
  );
```

22 Dazu kommt die Funktion iconv() zum Einsatz. Für nähere Informationen vgl. [35]

Dieser beinhaltet ein Theme, das in der Funktion `teisearch_theme()` definiert wurde, die Resultate selbst in der Variable `$results`, den Suchschlüssel und den Pager des aktuellen Offsets.

4.4.7 Darstellung der Suchtreffer

Suchtreffer erscheinen unterhalb des Suchformulars auf der Ergebnisseite. Einzelne Suchtreffer werden in Form eines Links und eines nebenstehenden Textauszugs, der die gesuchte Zeichenkette umgibt, angezeigt. Unterhalb der einzelnen Suchtreffer wird, sobald die Suche mehr als zehn Treffer liefert, ein Pager sichtbar. Dies geschieht mit Hilfe der Funktion `teisearch_teiview()`, die die von `teisearch_execute()` gelieferten Werte wie folgt verarbeitet:

```
$build['teisearch_search_form']=drupal_get_form(
        'teisearch_search_form', NULL, $keys);
$build['teisearch_results'] = $results;
```

Das Array `$build` wird für den Index `teisearch_search_form` mit dem von Drupal gerenderten Suchformular mittels `drupal_get_form()` initialisiert. Der Index `teisearch_results` wird mit dem Array `$results`, das die von der Funktion `teisearch_execute()` gelieferten Werte beinhaltet, initialisiert.

Den Rückgabewert der Funktion `teisearch_teiview()` bildet das Array `$build`. Demnach beinhaltet dieses Array das Suchformular und die Suchtreffer für ein bestimmtes Offset.

Die Visualisierung der Resultate erfolgt mit Hilfe der Funktion `teisearch_theme()`, welche die Drupal-Funktion `hook_theme()` implementiert. Diese Funktion hat den folgenden Rückgabewert:

```
return array(
  'teisearch_result' => array(
    'variables' => array('result' => NULL),
    'template' => 'teisearch-result',
  ),
  'teisearch_results' => array(
    'variables' => array('results' => NULL, 'keys' =>
      NULL, 'testsnippets' => NULL, 'pageroffset' => NULL),
      'template' => 'teisearch-results',
  ),
);
```

Im Array `variables` sind Variablen definiert, die von den in `template` definierten Templates verarbeitet werden. Für die Preprozessierung der Variablen sind die Funktionen `template_preprocess_teisearch_results()` sowie `template_preprocess_teisearch_result()` zuständig. Zunächst folgt der Quelltext der

beiden Templates, um zu verdeutlichen wie die Ausgabe aussehen soll und welche Werte benötigt werden:

`teisearch-results.tpl.php:`

```
<?php
// $Id$
?>
<?php if ($teisearch_results) : ?>
  <h2><?php print t('Search results for: '.$keys);?></h2>
  <ul class="teisearch-results">
    <?php print $teisearch_results; ?>
  </ul>
<?php print $pageroffset; ?>
<?php else : ?>
  <h2><?php print t('Your search yielded no results');?
></h2>
<?php endif; ?>
```

Das `teisearch-results.tpl.php`-Template benötigt die Variable `$keys`, deren Wert als Ausgabe des Suchbegriffs dient, die Variable `$teisearch_results`, deren Wert mit Hilfe des `teisearch-results.tpl.php`-Templates erzeugt wird und `$pageroffset`, die den Pager der unter den Suchtreffern angezeigt wird.

`teisearch-result.tpl.php:`

```
<?php
// $Id$
?>
<li>
  <h3 class="page">
    <a href="<?php print $url; ?>"><?php print $page;></a>
    <br></br>
    <span><?php print $textsnippet; ?></span>
  </h3>
</li>
```

Das `teisearch-result.tpl.php`-Template bekommt die Variable `$url` und `$textsnippet` übergeben, wobei `$url` die URL zu den einzelnen Suchtreffern und `$textsnippet` einen Textauszug um den Suchschlüssel herum darstellt.

Die Funktion `template_preprocess_teisearch_results()` ist für die Erzeugung aller Resultate zuständig, wohingegen die Funktion `template_preprocess_teisearch_result()` in einer Schleife für jedes Element des Resultats aufgerufen wird und individuell für jedes Element eine Ausgabe produziert. Mit Hilfe der beiden Templates erscheinen die Suchergebnisse auf der Ergebnisseite.

5 Zusammenfassung und Ausblick

Die Aufgabenstellung bestand darin, ein Modul für Drupal zu entwickeln, das eine verbesserte Suche gegenüber der Standard-Suche von Drupal, für *TEI-Lite* kodierte Dokumente bietet.

Nach der Ausarbeitung einer strukturierten Vorgehensweise für die Implementierung, begann die praktische Umsetzung des Vorhabens. Wie erwartet, ergaben sich im Verlauf der Implementierung Änderungen, die abweichend vom Konzept waren. Dies lag hauptsächlich daran, dass einige Punkte bezüglich der Umsetzung, im Vorfeld nicht bis ins kleinste Detail geplant werden konnten.

Vor dem tatsächlichen Einstieg in die Programmierung des Suchmoduls wurde zunächst ein Testmodul entwickelt, das den Umgang mit der *Drupal-API* näher bringen sollte. Es wurde ein Block implementiert, der ein Eingabefeld mit einem Knopf beinhaltete. Dieser Block sollte anschließend für das tatsächliche Suchformular genutzt werden. Hierbei ergaben sich bereits erste Erkenntnisse, die eine Abweichung zu Vorüberlegungen als Folge hatten. Die Ursache hierfür war die Frage, wann und wo dieser Block angezeigt werden soll. Daraufhin fiel die Entscheidung, das Suchformular mit Hilfe eines Filters zu erzeugen. Der Vorteil hierbei liegt darin, dass das Suchformular nur dann erscheint, wenn ein Node einen vom *XSL-Stylesheet* verarbeiteten Inhalt enthält.

Im Anschluss an dieses Experiment folgte die Implementierung des Such-Moduls, die stufenweise verlief. Zuerst wurden modulspezifische Eigenschaften umgesetzt, die der Administration dienen, beispielsweise die Installationsroutine, benutzerspezifische Einstellmöglichkeiten, etc.

Den nächsten Schritt stellte die Indexierung dar. Hier wurde die Speicherung von zu indexierenden Daten in Form von Worten, Texten, TEI-Tags, etc. umgesetzt. Hierbei traten oftmals Verzögerungen während der Implementierung auf, da die Struktur der Datenbanktabellen immer wieder überarbeitet werden mussten. Diese Veränderungen ergaben sich aus neu gewonnenen Erkenntnissen, die während Testabfragen aus der Datenbank ersichtlich wurden.

Nachdem die Indexierung den im Konzept festgelegten Ansprüchen genügte, begann die Implementierung der Suche. Zunächst wurde ein Suchformular mit Hilfe eines Filters entwickelt, das Suchanfragen verarbeitet. Im nächsten Schritt wurden ein weiteres Suchformular sowie zwei PHP-Templates implementiert, die den Aufbau der Ergebnisseite repräsentieren.

Das Suchformular beinhaltet Kontrollkästchen, mit denen die Suche dem Konzept entsprechend eingeschränkt werden kann. Treffer werden mit Hilfe der PHP-Templates dargestellt. Diese erscheinen als Links zur gefundenen Textstelle, samt einem Textauszug, der die gesuchte Zeichenkette beinhaltet. Folgt man solch einem Link, gelangt man an die Stelle, an der sich der gesuchte Schlüssel befindet, der hervorgehoben dargestellt wird. Das Hervorheben von gefundenen Suchschlüsseln nach einer Umleitung zur gefundenen Textstelle sowie Textauszüge auf der Trefferseite, wurden später als Features aufgenommen.

Neben der Implementierung des Indexiervorgangs, stellten diese beiden Features den schwierigsten Teil der Arbeit dar. Retrospektiv betrachtet lässt sich feststellen, dass die Verarbeitung von allerlei Zeichenketten, die im Modul stattfinden, häufig zu unvorhergesehenen Problemen geführt haben, deren Lösung teilweise zeitraubend war. Häufige Gründe hierfür lagen in zeichensatzspezifischen Eigenschaften oder an nicht berück-

sichtigten Sonderzeichen[23].

Während einer Vorführung eines Freigabekandidaten des Moduls gegenüber dem Projektinitiator, kamen weitere Ideen für die finale Version auf. Die Änderungswünsche bezogen sich dabei auf suchspezifische Eigenschaften. Diese Änderungen konnten zügig realisiert werden.

Nachdem alle im Konzept gesetzten Anforderungen realisiert sowie die gewünschten Änderungen des Projektinitiators umgesetzt wurden, war die finale Version des Moduls fertiggestellt.

Einige Ideen, die während der Entwicklungsphasen aufkamen, konnten auf Grund des gegebenen Zeitrahmens nicht in die finale Version einfließen.

5.1 Ausblick

Nachdem auf Grund des zeitlich begrenzten Rahmens die Programmierarbeiten eingestellt wurden, blieben noch einige Wünsche offen. Einerseits konnte die Erzeugung von Textausschnitten sowie das Hervorheben von gefundenen Zeichenketten im Text nicht bis zur Perfektion vorangetrieben werden. Andererseits entstanden viele neue Ideen während der einzelnen Entwicklungs- und Testphasen.

Des Weiteren wurden verschiedene Seiten aufgesucht, die TEI kodierte Inhalte bereitstellen. Besonders positiv fielen dabei einige Aspekte der Gestaltung der Suche der „*collection of Middle English texts“* der *University Of Michigan* auf. Diese schreibt über die Kollektion:

> *„All texts in the archive are valid SGML documents, tagged in conformance with the TEI Guidelines, and converted to the TEI Lite DTD for wider use.“* [38]

Nach näherer Betrachtung der Suche und deren Funktionalitäten in der Kollektion, sind einige dieser Aspekte in die Aufzählung der Ideen eingeflossen, beispielsweise die Navigation durch die Funde auf einer Seite.

Es folgt eine Liste der entstandenen Wünsche und Ideen, die im Falle einer Weiterentwicklung des *TEISearch-Moduls* denkbar wären:

- Die nach den Weiterleitung zur Fundstelle hervorgehobenen Treffer könnten auf die Seite, auf der sie vorkommen beschränkt werden
- Die hervorgehobenen Treffer könnten, mit Hilfe eines Previous- und Next-Knopf, „*durchklickbar*“ sein
- Nach einer Weiterleitung zum Text könnte die Version des Textes passend zum Suchtreffer umgeschaltet werden
- Zugeklappte Anmerkungen des Autors und des Herausgebers könnten aufgeklappt oder die Zahl, in der sich der gefundene Schlüssel befindet, hervorgehoben werden
- Das Suchformular der Ergebnisseite könnte weitere Optionen anbieten, beispielsweise Einschränkungen für einzelne Kapitel, Bücher etc.

23 Bei den zu indexierenden Dokumenten handelt es sich um französische Texte, die sehr viele Sonderzeichen beinhalten.

- Das Sortieren von Treffern nach bestimmten Kriterien, beispielsweise Buchtitel, Titel eines Kapitels, Datum der Veröffentlichung und Überarbeitung eines Inhalts, Relevanz, etc. wäre denkbar
- Erzeugung aller Textauszüge zu jedem Treffer, der auf einer Seite vorkommt
- Nähere Informationen für einzelne Treffer, beispielsweise wann der Text hochgeladen wurde sowie den letzte Zeitpunkt an dem der Inhalt geändert wurde, den Autor des Textes, etc.
- Präsentation der Anzahl aller Treffer wäre sinnvoll
- Mit Hilfe des Index könnte die Möglichkeit angeboten werden, beispielsweise die Struktur eines Buches, ein Glossar sowie andere derartige strukturierte Ausgaben zu erzeugen.

Abschließend lässt sich konstatieren, dass die Digitalisierung, besonders im Bereich der Literatur, neue Möglichkeiten sowohl für die Speicherung, als auch die Bereitstellung bietet. Durch die Erarbeitung des hier vorgestellten Moduls konnte somit ein Beitrag zur effektiveren Nutzung dieser Medien geleistet werden.

Anhang A – Systemspezifikationen

Für die Implementierung sowie Tests standen zwei Rechner mit folgenden Spezifkationen bereit:

- Ein Laptop
  - Betriebssystem:
    ```
    $cat /etc/issue
    Debian GNU/Linux 5.0 \n \l
    ```
  - Kernel:
    ```
    $uname -a
    Linux deb-lpt 2.6.26-2-686 #1 SMP Mon Jun 21
    05:58:44 UTC 2010 i686 GNU/Linux
    ```
  - Software:
    - Browser:
      I. Opera
      ```
      $opera --version
      Opera 10.61. Build 6430 for Linux.
      Compiled on Aug  9 2010 by gcc 4.3.2 (ABI:
      1002) for GNU libc 2.7.
      ```
      II. Iceweasel
      ```
      $iceweasel --version
      Mozilla Iceweasel 3.0.6, Copyright (c) 1998 -
      2009 mozilla.orga
      ```
      III. Google Chrome
      ```
      $google-chrome --version
      Google Chrome 6.0.472.36 beta
      ```
      IV. Konqueror
      ```
      $konqueror --version
      Qt: 3.3.8b
      KDE: 3.5.10
      Konqueror: 3.5.9
      ```
    - Webserver
      ```
      $apache2 -v
      Server version: Apache/2.2.9 (Debian)
      Server built:   Apr 19 2010 19:57:58
      ```
    - MySQL
      ```
      $mysql --version
      mysql  Ver 14.12 Distrib 5.0.51a, for debian-
      linux-gnu (i486) using readline 5.2
      ```
    - Drupal
      Version: `7.0-alpha6` und `7.0-alpha7`

- Ein Desktop-Rechner
  - Betriebssystem:

```
$cat /etc/issue
Debian GNU/Linux squeeze/sid \n \l
```

- Kernel:

```
$uname -a
Linux deb-wz 2.6.32-5-amd64 #1 SMP Thu Aug 12
13:01:50 UTC 2010 x86_64 GNU/Linux
```

- Software:
  - Browser:
    I. Opera

```
$opera --version
Opera 10.61. Build 6430 for Linux.
Compiled on Aug  9 2010 by gcc 4.3.2 (ABI:
1002) for GNU libc 2.7.
```

    II. Iceweasel

```
$iceweasel –version
Mozilla Iceweasel 3.5.11, Copyright (c) 1998
- 2010 mozilla.org
```

    III. Chrome

```
$google-chrome --version
Google Chrome 6.0.472.51 beta
```

    IV. Konqueror

```
$konqueror --version
Qt: 4.6.3
KDE: 4.4.5 (KDE 4.4.5)
Konqueror: 4.4.5 (KDE 4.4.5)
```

  - Webserver

```
$apache2 -v
Server version: Apache/2.2.16 (Debian)
Server built:   Jul 24 2010 20:57:19
```

  - MySQL

```
$mysql --version
mysql  Ver 14.14 Distrib 5.1.49, for debian-
linux-gnu (x86_64) using readline 6.1
```

  - Drupal
    Version: 7.0-alpha6 und 7.0-alpha7

Literaturverzeichnis

[1] Uwe M. Borghoff, Peter Rödig, Jan Scheffczyk und Lothar Schmitz (eds.). *Long-Term Preservation of Digital Documents. Principles and Practices*, Springer Verlag, Berlin/Heidelberg/New York 2005

[2] John K. van Dyk. *Pro Drupal Development*, Apress, Berkeley, 2. Aufl. 2008

[3] John K. van Dyk. *Das Drupal-Entwicklerhandbuch: Der Praxisleitfaden für Drupal-basierte Webprojekte*, Addison-Wesley, München, 1.Aufl. 2009

[4] Hagen Graf. Drupal. *Community-Websites entwickeln und verwalten mit dem Open Source-CMS*, Addison-Wesley, München u.a. 2006

[5] Jonathan Gennick. *SQL – kurz & gut*, O'Reilly, Beijing u.a. , 2. Aufl. 2007

[6] Carsten Möhrke. *Besser PHP programmieren. Handbuch professioneller PHP-Techniken*, Galileo Press, Bonn, 3. aktualisierte und erweiterte Aufl. 2009

[7] Jacob Redding. *Beginning Drupal*, Wiley Publishing, Inc., Indianapolis 2010

Webseiten (Letzter Zugriff am 1.10.2010)

[8] Wikipedia. François-Joseph Bérardier de Bataut
http://de.wikipedia.org/wiki/Fran%C3%A7ois-Joseph_B%C3%A9rardier_de_Bataut

[9] Wikipedia. Text Encoding Initiative
http://de.wikipedia.org/wiki/Text_Encoding_Initiative

[10] Wikipedia. Drupal
http://en.wikipedia.org/wiki/Drupal

[11] Wikipedia. Text Encoding Initiative
http://en.wikipedia.org/wiki/Text_Encoding_Initiative

[12] Wikipedia. Numeric character reference
http://en.wikipedia.org/wiki/Numeric_character_reference

[13] Wikipedia. Iconv
http://de.wikipedia.org/wiki/Iconv

[14] Christof Schöch. Bérardier de Batauts Essai sur le récit
http://www.christof-schoech.de/essai-sur-le-recit

[15] Drupal's Search Framework: The execution of a search
http://acquia.com/blog/drupals-search-framework-the-execution-a-search

[16] Wesley Tanaka. How the Drupal 6 search indexer works
http://wtanaka.com/drupal/search-cron-6

[17] Drupal-API. API reference
http://api.drupal.org/api/7

[18] Drupal-API. hook_filter_info
http://api.drupal.org/api/function/hook_filter_info/7

[19] Drupal-API. Hooks
http://api.drupal.org/api/group/hooks/7

[20] Drupal. Specifying the allowed formats for user input
http://drupal.org/handbook/modules/filter

[21] Drupal. Blocks
http://drupal.org/handbook/blocks

[22] Drupal. Creating Drupal 7.x modules
http://drupal.org/node/361112

[23] Drupal. Devel
http://drupal.org/project/devel

[24] Drupal. Writing .info files (Drupal 7.x)
http://drupal.org/node/542202

[25] Drupal. Developing for Drupal
http://drupal.org/contributors-guide

[26] Drupal. Introduction to PHP for theming
http://drupal.org/node/348916

[27] Drupal Center. Benutzerhandbuch
http://www.drupalcenter.de/handbuch

[28] Drupal Code Search. Drupal Code Search helps you find Drupal code easier!
http://drupalcodesearch.com/

[29] Text Encoding Initiative. P5: Richtlinien für die Auszeichnung und den Austausch elektronischer Texte
http://www.tei-c.org/release/doc/tei-p5-doc/de/html/ref-teiHeader.html

[30] Text Encoding Initiative. TEI Lite
http://www.tei-c.org/Guidelines/Customization/Lite/

[31] PHP. PHP Manual
http://www.php.net/manual/en/

[32] Computerphilologie Uni-München. TEI in der Praxis
http://computerphilologie.uni-muenchen.de/praxis/teiprax.html

[33] MySQL 5.1 Reference Manual.
http://dev.mysql.com/doc/refman/5.1/en/index.html

[34] MySQL 5.1 Reference Manual. B.5.5.1. Case Sensitivity in String Searches
http://dev.mysql.com/doc/refman/5.1/en/case-sensitivity.html

[35] PHP. PHP Manual, iconv
http://php.net/manual/de/function.iconv.php

[36] PHP. PHP Manual, String Functions
http://php.net/manual/en/ref.strings.php

[37] University of Michigan. Corpus of Middle English Prose and Verse
http://quod.lib.umich.edu/c/cme/

[38] University of Michigan. Library Digital Collections
http://quod.lib.umich.edu/lib/colllist/

[39] University of Chicago. PhiloLogic
http://philologic.uchicago.edu/

[40] Wikipedia. Cron
http://de.wikipedia.org/wiki/Cron