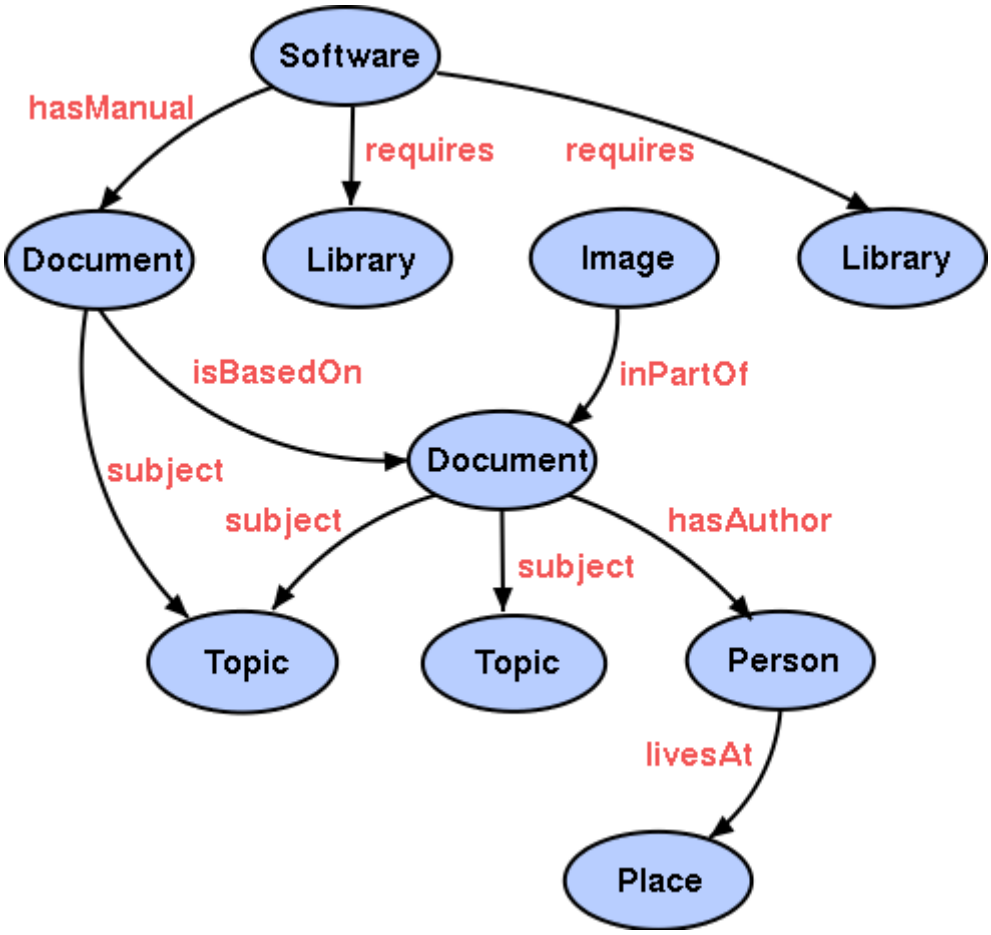
Ontologie-Editor



1 Revision History

Version	Datum	Autor	Bemerkung
0.1	17.09.10	Moos	Initiale Struktur und erster Teil Requirements Engineering, technisches Design
0.2	05.10.10	Moos	Erweiterung RE, Unterteilung in Use cases und NFR
0.3	15.10.10	Moos	Titelblatt, GUI Entwurf
0.4	15.10.10	Lahn	ERM Entwurf
0.5	10.11.10	Moos	Relationales Modell eingefügt, kleine Anpassungen der Usecases
0.6	28.12.10	Moos	Sektion Applikationsaufbau
0.7	15.01.11	Lahn	Final Review (Update RDM und UI-Beschreibung, etc.)

Inhaltsverzeichnis

2 Requirements Engineering

2.1 – Use-Cases

Die Grundlage für das Requirements Engineering stellt das abgegebene Dokument „Aufgabenstellung Semesterarbeit "Cooperative Modeling" dar.

Die Implementierungsdetails darunter sind per Definition frei. Alle gemachten Entscheide sollten der Übersicht halber hier kurz erwähnt werden.

2.1.1 Verwaltung von Begriffen

Es müssen Begriffe verwaltet werden können. Diese Funktionalität ist eine der grundlegenden Bausteine der Applikation.

2.1.2 Verwaltung von Beziehungstypen

Es müssen verschiedene Beziehungstypen erfasst werden können. Dies ist den Admin-Accounts vorbehalten.

2.1.3 Erfassen und Ändern von Beziehungen

Die definierten Begriffe werden untereinander verknüpft. Dies geschieht über die gerichteten Begriffstypen aus 2.1.2.

2.1.4 Locking der einzelnen Einträge

Wenn ein Benutzer einen entsprechenden Eintrag ändern will und die Maske aufruft, werden der Eintrag und alle Beziehungen damit gesperrt. Diese Sperre wird aufgehoben, wenn dieselbe Session eine andere Seite öffnet resp. die Änderungen abschliesst.

Für den Fall, dass der Benutzer seinen Browser ohne Logout schliesst, wird die Sperre nach 2 Minuten wieder aufgehoben.

Die anderen Benutzer sehen ein gesperrtes Item mit einer anderen Farbe und bekommen eine Warnung wenn sie dieses ändern wollen.

2.1.5 History

Es soll eine History geführt werden, in welcher die Änderungen von Begriffen und Relationen kurz kommentiert werden muss. Dies dient der Absprache mit den anderen Teammitgliedern und soll ein separates Tool (Forum / Chat) unnötig machen.

Die History-Funktion wird auch der Fokus dieser Arbeit darstellen. Es ist sicher, dass die Änderungen an einem Objekt textuell nachvollziehbar sind. Wir werden uns auch darauf konzentrieren, das Datenbankmodell so zu erstellen, dass zu einem späteren Zeitpunkt ein „read-only rollback“ gemacht werden kann. Die gesamte Ontologie könnte damit auf den früheren Stand gebracht werden.

Eine weitere Ausbaumöglichkeit wäre noch, RSS Feeds zu generieren. Diese könnten Benutzer über Änderungen an ihren Objekten informieren.

2.1.6 Accounts und Rollen

Accounts müssen verwaltet werden können. Zudem werden Benutzer in eigentliche User und Admins unterteilt, wobei Admins die gesamte Userverwaltung erledigen.

Zudem ist den Admins vorbehalten, neue Beziehungstypen zu erfassen.

2.2 Nicht-funktionale Anforderungen

2.2.1 Darstellung

Die Darstellung der Ontologien und deren Beziehungen soll möglichst anwenderfreundlich sein. Eine Möglichkeit stellt die Benutzung von dynamisch generierten Vektorgrafiken dar. Der Aufwand dafür muss noch abgeklärt werden.

Es soll nicht zu viel Zeit in dieses Problem investiert werden.

2.2.2 Realisierte Darstellung

Zur Anwenderfreundlichkeit siehe Punkt 4.1.6. Aufgrund der Wahl des Schwerpunktes und der zur Verfügung stehenden Zeit haben wir uns gegen die Generierung von Vektorgrafiken entschieden. Die Ontologie wird stattdessen in kontextbezogener tabellarischer Form abgebildet.

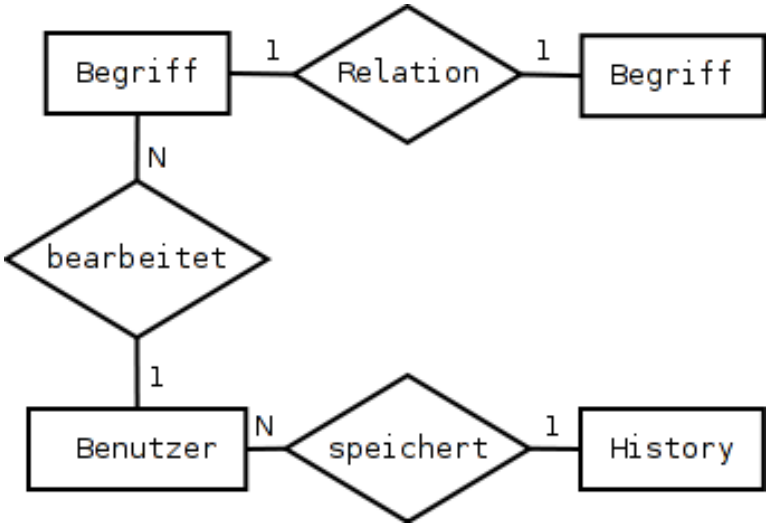
2.2.3 Mehrbenutzerfähigkeit

Das Tool muss von mehreren Benutzern gleichzeitig bearbeitet werden können. Dabei darf es nicht zu inkonsistenten Daten kommen.

Dieses Ziel wurde nur teilweise erreicht, siehe Punkt 4.1.6.

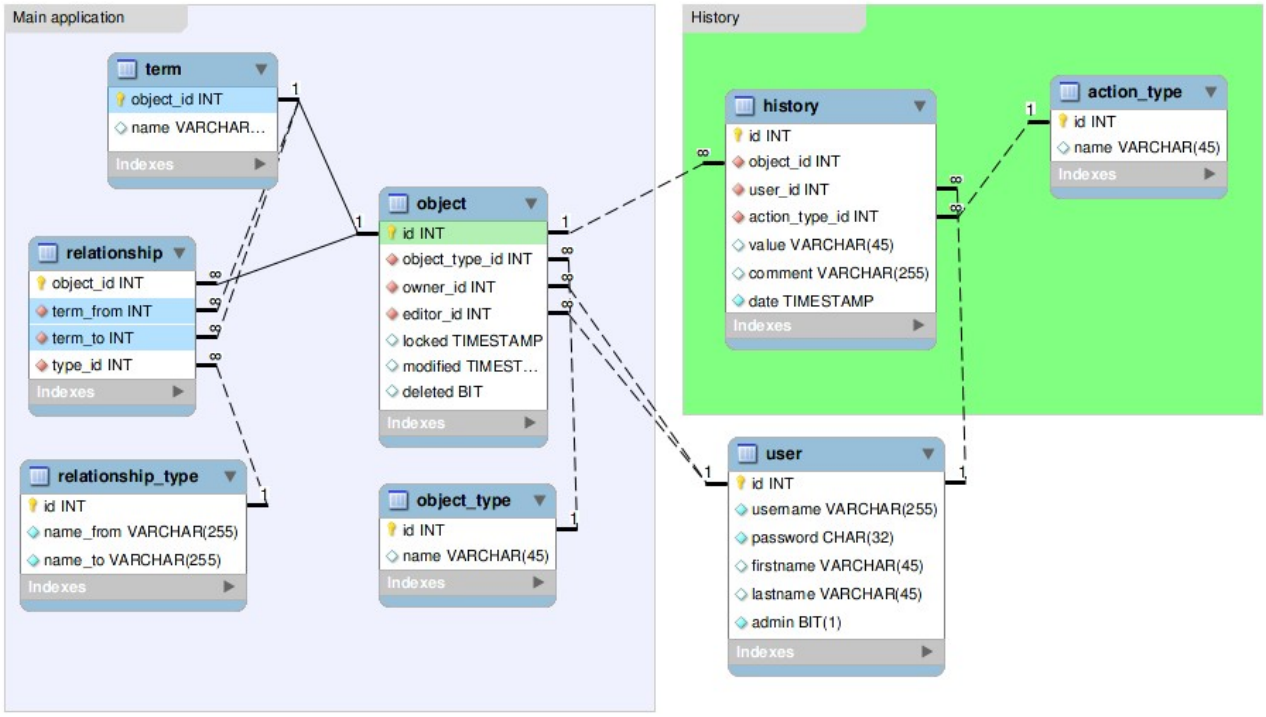
3 Technisches Design

3.1 Entity-Relationship-Modell



3.2 Relationales Datenmodell

3.2.1 Diagramm



3.2.2 Erläuterung

Das relationale Modell setzt sich aus dem Verwaltungsteil für die Ontologie, sowie dem History-Teil zusammen.

Der Ontologie-Teil ist um die zentrale object-Tabelle aufgebaut. Ihre Struktur stellt eine Generalisierung von term bzw. relationship_type dar. Der Hauptgrund für diese Entscheidung ist die Vereinfachung des History-Teils. So muss dieser nur eine History eines einzigen Objekts führen.

Die Tabelle Relationship speichert die eigentliche Beziehung zwischen zwei Begriffen.

Der History-Teil ist relativ selbst erklärend, die Haupttabelle zeichnet alle gemachten Änderungen auf. Zudem wird eine Tabelle action geführt, welche alle möglichen Aktionen (create, rename, delete etc. führen soll). Dies könnte später das angesprochene read-only feedback ermöglichen.

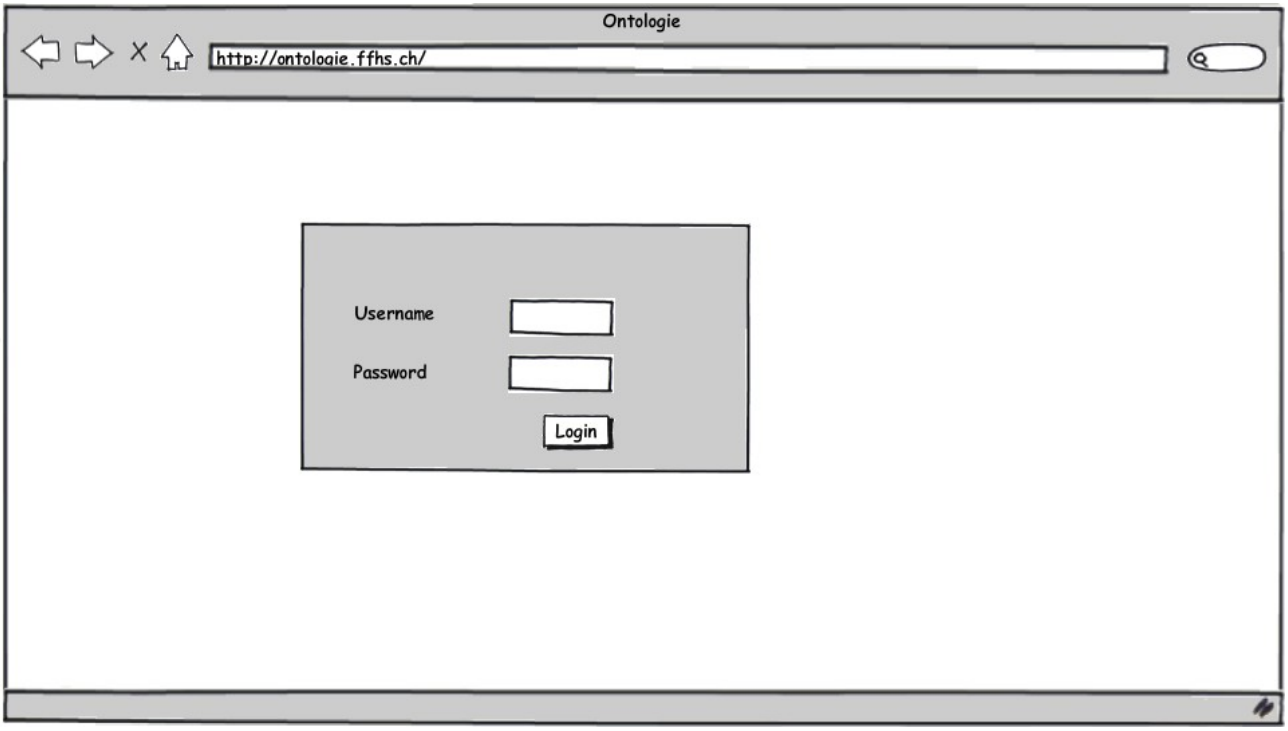
4 GUI-Entwurf

Der GUI-Entwurf soll kurz darstellen, wie die Applikation aussehen könnte. Sie dient vor allem als Grundlage, dem Kunden einen ersten Eindruck liefern zu können.

Als Tool haben wir uns für Balsamiq Mockups entschieden, welches schnelle und simple Entwürfe ermöglicht.

4.1 Ablauf

4.1.1 Login

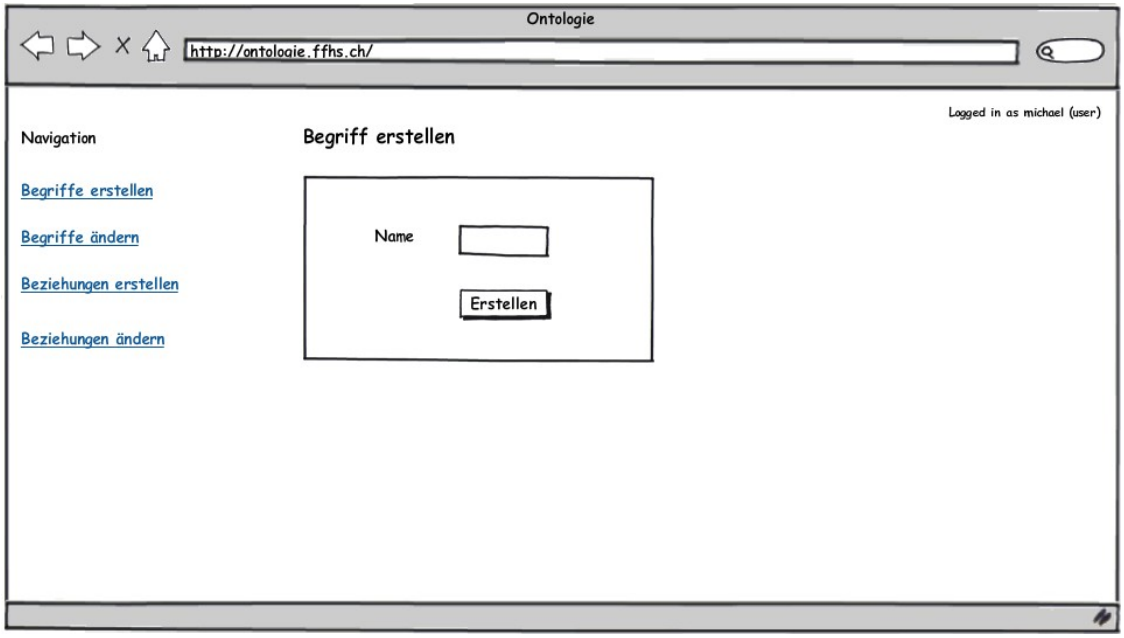


In einem ersten Schritt kann sich der Benutzer einloggen.

Danach erscheint die eigentliche Applikation, links die Navigation.

4.1.2 Begriff hinzufügen

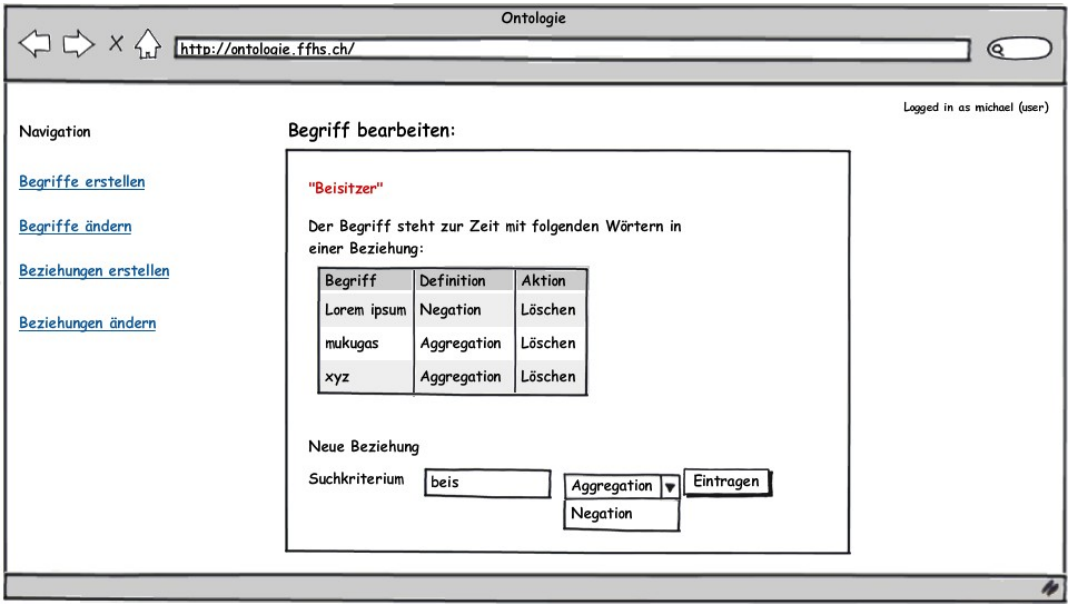
Die erste Funktion nach dem Login (im Benutzerbereich) dient dazu, Begriffe hinzuzufügen.



Hierbei muss lediglich der Name des neuen Begriffs eingegeben werden. Im nächsten Fenster können Beziehungen zu anderen Wörtern aufgebaut werden.

4.1.3 Begriff bearbeiten

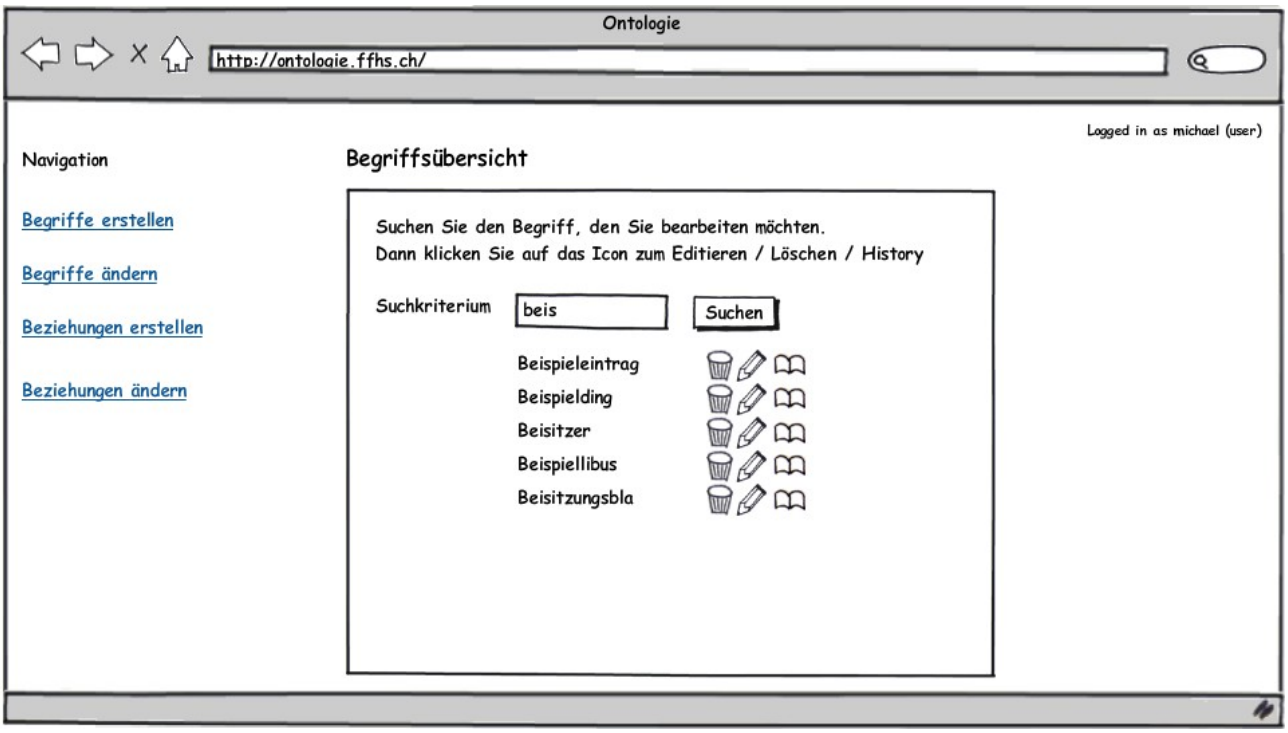
In diesem Formular erscheinen bereits bestehende Beziehungen zu anderen Begriffen – unten können neue hinzugefügt werden.



4.1.4 Begriffsübersicht

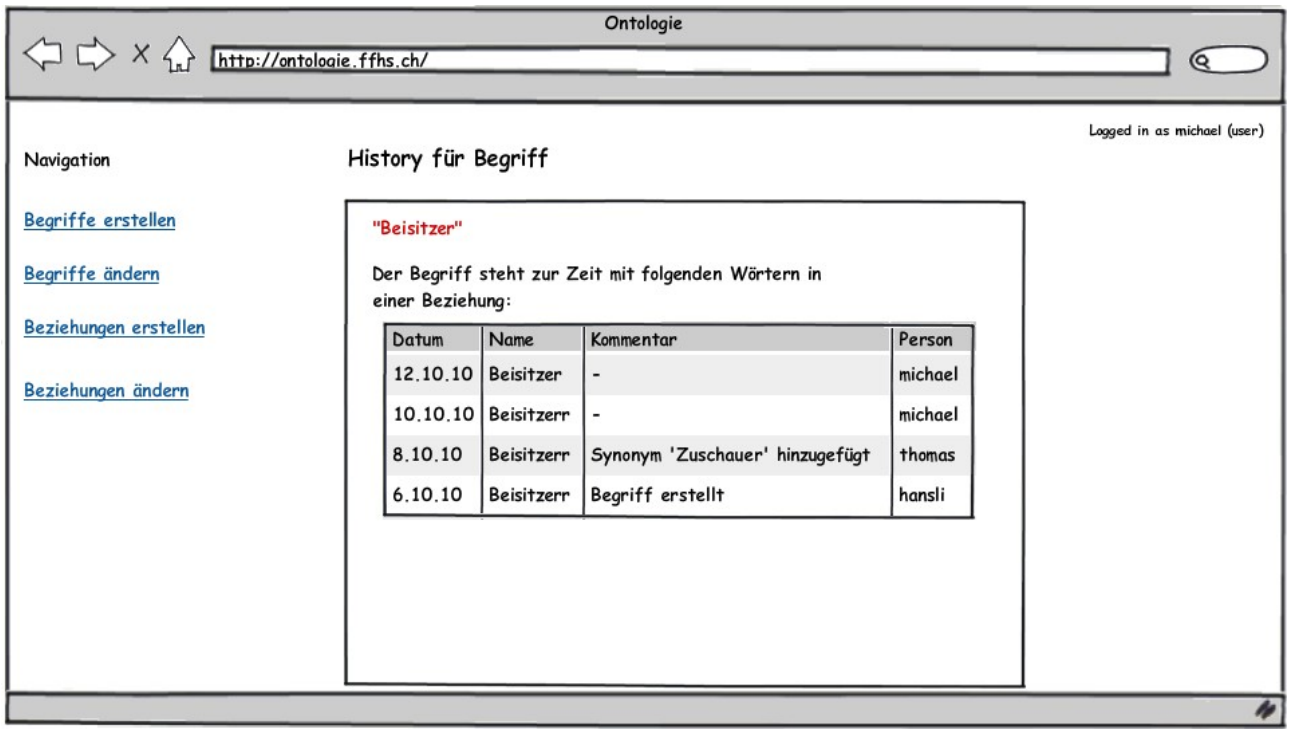
Wenn Begriffe geändert werden sollen, wird zuerst in einer Übersicht der entsprechende Eintrag ausgewählt. Dieser kann hier direkt gelöscht werden. Über die Editierfunktion kommt man auf die Bearbeitungs-Funktion zurück (siehe Hinzufügen).

Die letzte hier verfügbare Funktion ist die History. Diese wird im folgenden Abschnitt beschrieben.



4.1.5 History

Die History dient dazu, Änderungen zu kommentieren. So wird jede Änderung an einem Begriff systematisch geloggt. Es ist auf jeden Fall möglich, Kommentare zu erfassen, damit andere Mitarbeiter wissen, warum etwas geändert wurde (analog bsp. SVN-Commits).



Die Beziehungen funktionieren analog zu den Begriffen. Es wird deshalb in diesem Rahmen darauf verzichtet, diese noch explizit aufzuführen.

4.1.6 Realisierte Applikation

Die realisierte Applikation unterscheidet sich vom Entwurf in wichtigen Punkten:

- Benutzerfreundlichkeit
  - Die jeweiligen Funktionen Erstellen und Ändern/Löschen wurden in jeweils einem Menüpunkt zusammengefasst. Somit ist es möglich, schon beim Hinzufügen eines Objekts (Begriff oder
  - Es ist möglich, direkt von einem Begriff zu den verwendeten Beziehungen und von einer Beziehung direkt zu den in Beziehung stehenden Begriffen zu navigieren.
  - Die von einem Begriff verwendeten Beziehungen können auch beim Bearbeiten eines Begriffs gelöscht werden.
  - Es werden Icons verwendet, um die Affordance der jeweiligen Steuerelemente/Links deutlich zu machen.
  - Das Löschen von Objekten erfordert eine Bestätigung.
- Funktionalität
  - Das Löschen von Begriffen im Benutzer-Bereich wurde nicht implementiert. Dies einerseits aus Zeitgründen, andererseits halten wir es für sinnvoller, diese Funktion später nur dem Admin zur Verfügung steht.
  - Auf eine separate Anzeige der History (und einen separaten Link) wurde verzichtet; die History wird beim Bearbeiten eines Begriffes oder einer Beziehung unterhalb des Eingabeformulars angezeigt.
  - Die History („Änderungsprotokoll“) wird derzeit nur für die Aktionen „Hinzufügen“ „Umbenennen“ (nur Begriffe) und „Ändern“ (nur Beziehungen) geführt. Objekte werden nach Bestätigung aus der Datenbank gelöscht, dies wird nicht protokolliert. In einer nächsten Version sollten Objekte nur noch auf gelöscht gesetzt werden (mit evtl. Wiederherstellungsfunktion), der Löschvorgang protokolliert und das Lösch-Flag bei der Anzeige berücksichtigt werden.
  - Das geplante Locking konnte aus Zeitgründen nicht mehr funktionell implementiert werden. Dies muss vor dem Einsatz in einer Mehrbenutzerumgebung nachgeholt werden (die entsprechenden Klassenattribute sind vorhanden).

5 Applikationsaufbau

5.1 Technologien

Die Applikation läuft auf Basis MySQL 5.1 mit Tomcat 6.0.28-9 als Webserver für die JavaServer Pages (JSP). Als „Model-2“-Framework wird Struts 2 verwendet, der Persistenz-Layer wird mit Hibernate realisiert.

Für das Template-System wird Sitemesh verwendet. Dieses rendert die entsprechenden Seiten ins Design und verknüpft CSS etc.

Im nachfolgenden Absatz werden die einzelnen Konfigurationen näher beschrieben.

5.2 Struktur

Die Applikation ist nach dem „Model-2“-Prinzip aufgebaut.

5.2.1 Views

Die JSP-Views liegen im WebContent-ordner. Diese werden mit Hilfe von Sitemesh gerendert. Die Konfigurationsdatei dafür liegt unter WebContent/WEB-INF/decorators.xml.

Dieses legt fest, dass das Template aus dem Folder WebContent/decorators/mainTemplate.jsp für alle Views verwendet wird.

Die JSP-Views sind unterteilt in den /admin/ und /user/ Bereich. Entsprechend abhängig von diesem Pfad wird auch die Navigation für /admin/ bzw. /user/ geladen.

5.2.2 Struts-Framework

Das Struts-Config File liegt im /src/struts.xml. Diese Konfiguration ist zentral und definiert die verwendeten Actions und Views für eine Aktion. Sie ist ebenfalls in ein user- und admin-Package unterteilt, weil es zwischen den jeweiligen Funktionalitäten nur minimale Überschneidungen gibt und die Auftrennung die Applikation übersichtlicher macht.

Alle Struts-Actions liegen im Java Package `ch.ffhs.webE.action`.

5.2.3 Hibernate-Persistenz

Die Hibernate-Konfiguration befindet sich in /src/hibernate.cfg.xml. Dort werden die Verbindungsparameter für die Datenbankschicht definiert. Ebenso wird auf die Domain-Klassen verwiesen, welche entsprechend im Package *ch.ffhs.webE.domain* zu finden sind. Diese sind mit den Hibernate-Tools von JBoss grundlegend erstellt und dann manuell verfeinert worden.

Alle für den Hibernate-Zugriff verwendeten Data Access Objects (DAOs) liegen im Package ch.ffhs.webE.dao.