Entwurfsbeschreibung

SWP12-10

Peggy Lucke
Felix
Ha Tran
Paul Röwer
Alexander Richter
Nathanael Philipp

21. Mai 2012

Inhaltsverzeichnis

1 Allgemeines

LGDEditTool ist ein einfaches Bearbeitungswerkzeug für LinkedGeoData, welches den Nutzer befähigen soll, Mappings der OpenStreetMap-Daten zu editieren und auch zu löschen. Es vereinfacht die komplexe Eingabe in die Datenbank von LinkedGeoData und bietet in der Edit-History die Möglichkeit, die Änderungen einzusehen und wieder zurück zu nehmen. Außerdem ist ein leichtgewichtiger Ontologie-Editor integriert, mit dem z.B. Subklassenbeziehungen erstellt werden können. Für den Administrator besteht die Möglichkeit, in einem nur für ihn sichtbaren Tab die Edit-Historie zu löschen. Das Tool läuft in jedem Browser mit JavaScript-Unterstützung. Es ist also betriebssystemunabhängig.

2 Produktübersicht

Das Tool wird im Webbrowser gestartet und besteht im Wesentlichen aus einer Weboberfläche, welche die Datenbank, auf die sie zugreift, überdeckt und die Befehle in Buttons ausführbar macht. Damit der Nutzer einen Eintrag in der Datenbank gezielt suchen kann, gibt es noch ein Eingabefeld für die Suche nach Keys und Values. Die Darstellung der Daten erfolgt in Tabellen. Aufgerufen wird das Tool ohne Installation über eine Webseite.

3 Grundsätzliche Struktur- und Entwurfsprinzipien für das Gesamtsystem

Unser Programm setzt bei der Programmierung die Kenntnisse über folgende Programmiersprachen voraus: Java Server Pages, Cascading Style Sheets, Java, JavaScript sowie SQL.
Die Entwicklung unseres Programmes verläuft nach dem Top-Down-Schema. Dabei wurde zuerst die Oberfläche konstruiert und danach die Funktionalitäten der einzelnen Tabs implementiert. Das HTML-Dokument, welches im Browser des Nutzers die Oberfläche des Tools darstellt, wird dynamisch von einem Tomcat-Server aus dem JSP-Code-Dokument *index.jsp* generiert. Je nach ausgewähltem Tab wird dabei das entsprechende Java-Template ausgeführt, welches gegebenenfalls die notwendigen Datenbankabfragen vornimmt und anschließend den HTML-Code zur Darstellung der Ergebnisse erzeugt. Die Autovervollständigung wird mit dem JQuery-Framework implementiert.

4 Grundsätzliche Struktur- und Entwurfsprinzipien der einzelnen Pakete

4.1 Verwendete Bibliotheken

Wir verwenden für die Verbindung mit der PostgreSQL-Datenbank den JDBC-Treiber (PostgreSQL 9.1, JDBC 3/4 ↑). Für die die Autovervollständigung der Suchfelder die jquery.autocomplete.js (↑).

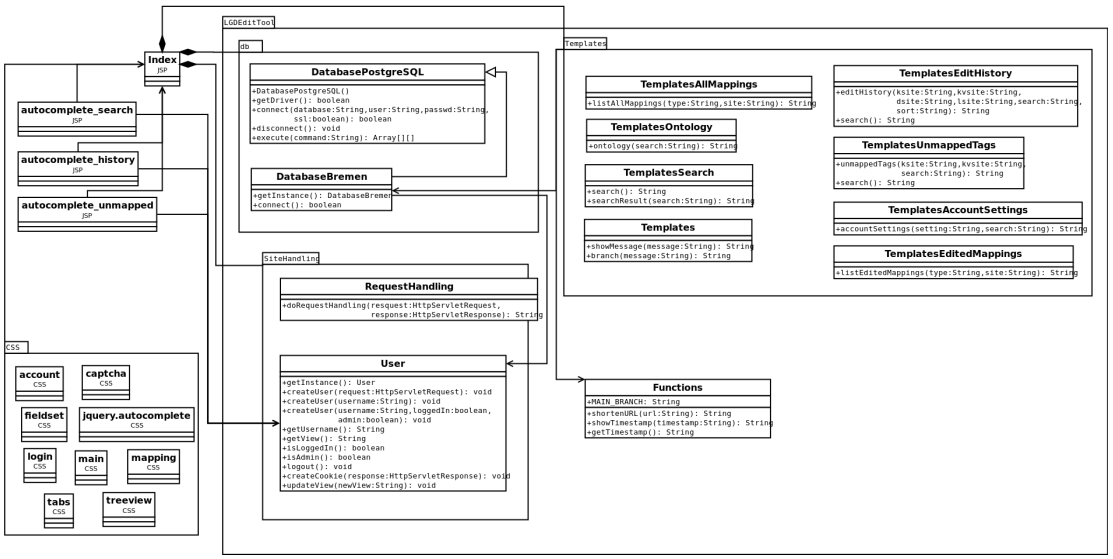

Abbildung 1: statisches Modell

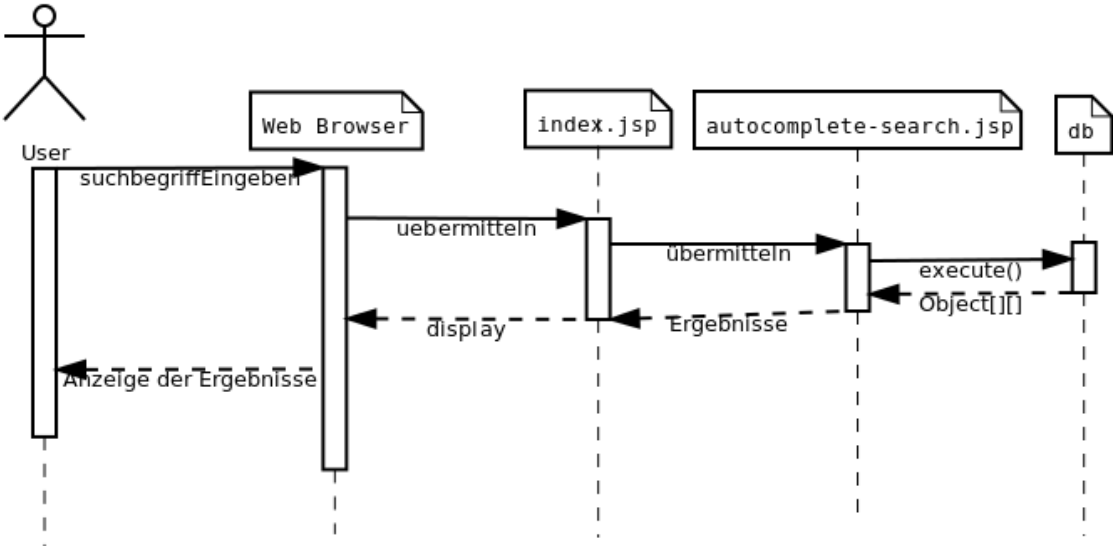

Abbildung 2: Autovervollständigung

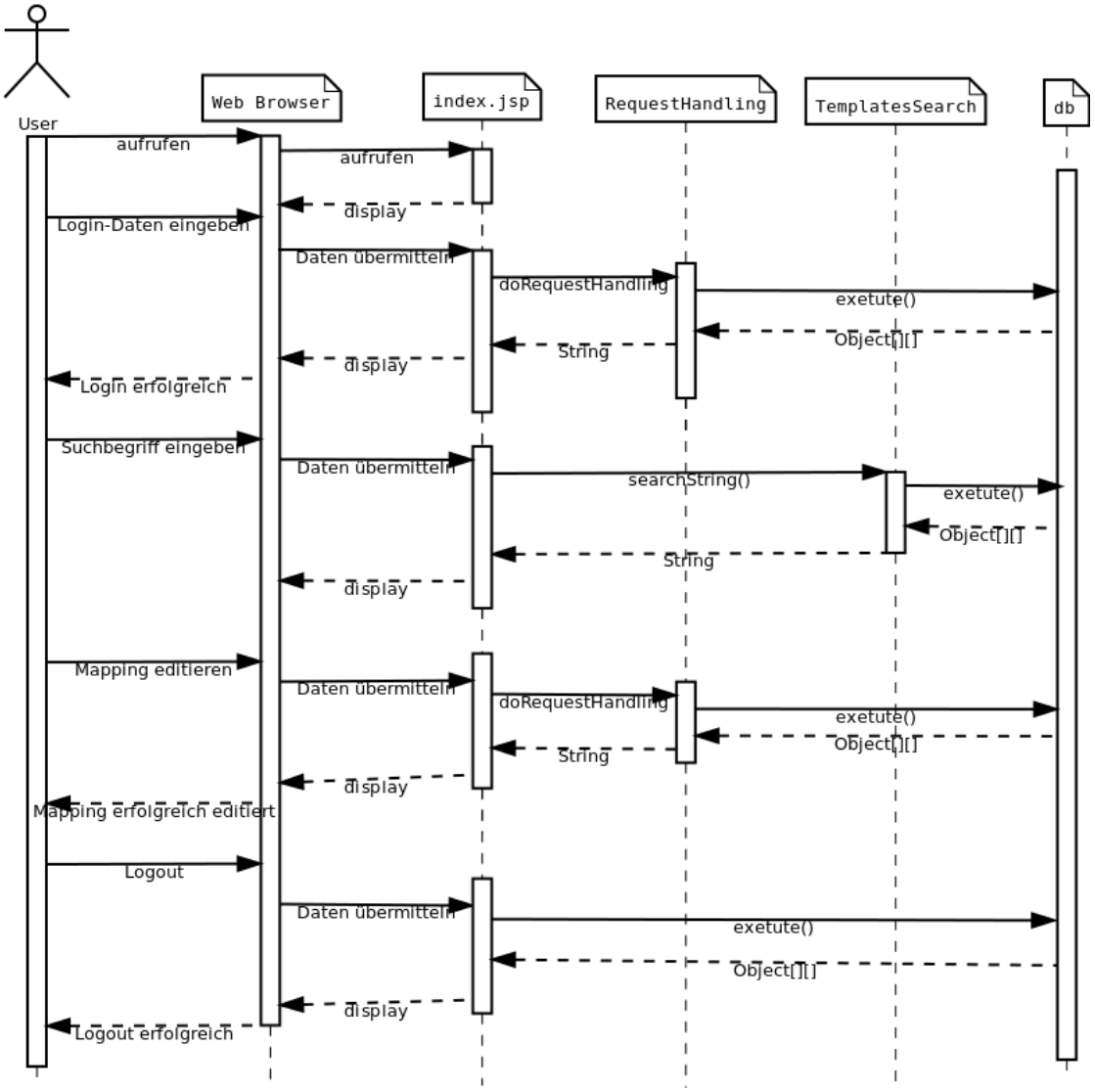

Abbildung 3: Editieren eines Mappings

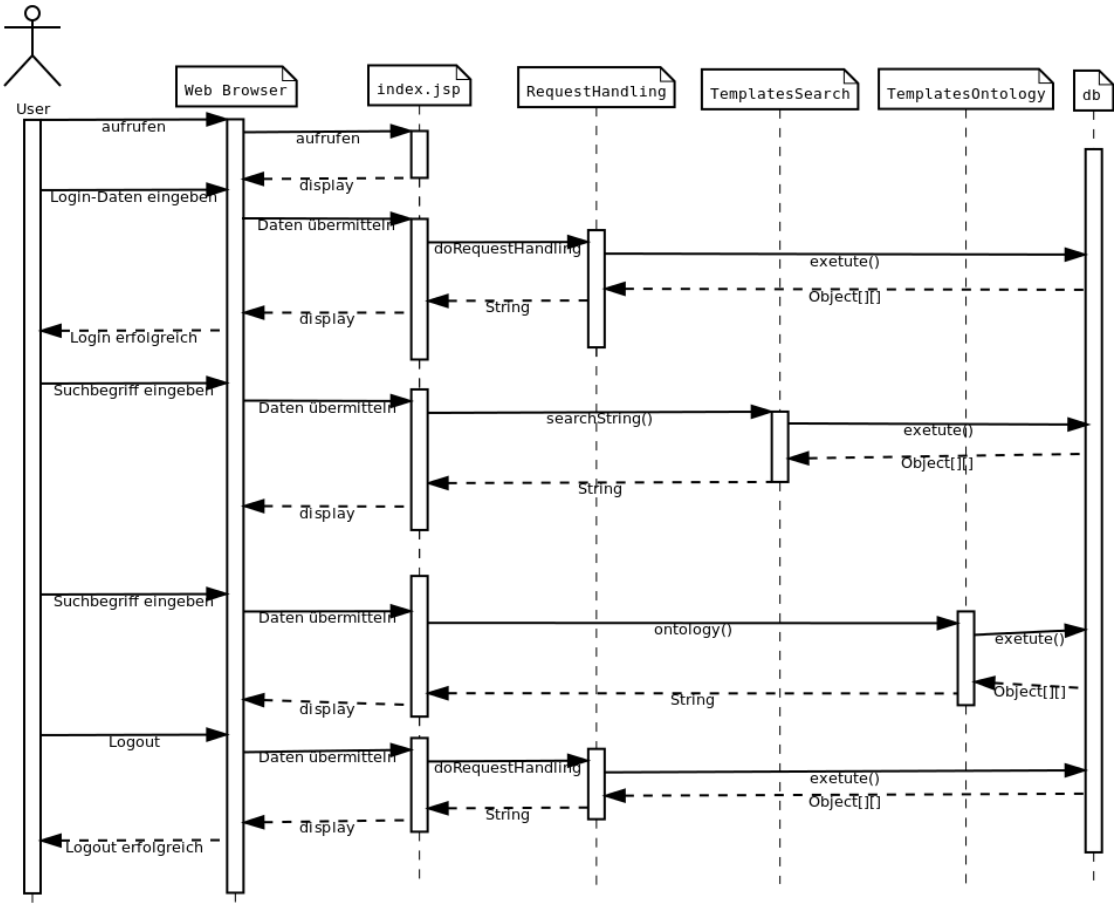

Abbildung 4: Ontologie

4.2 Ordner *css*

In diesem Ordner sind alle Gestaltungsvorgaben, um ein einheitliches Webdesign unseres Tools für alle Webbrowser zu ermöglichen.

4.3 Ordner *js*

Jeder in JavaScript geschriebene Code ist hier vertreten. Die beiden Javascript-Dateien sind nicht selbst geschrieben, sondern wurden übernommen.

4.3.1 *jquery.autocomplete.js & jquery.js*

Enthält den für die Anzeige der Autovervollständigung der Suchfelder erforderlichen Code.

4.4 *autocomplete_search.jsp & autocomplete_history.jsp & autocomplete_unmapped.jsp*

Enthält die Datenbankabfrage für die Autovervollständigung der Suchfelder. Entsprechend der Namen der jsp-Dateien sind diese Dateien für die Autovervollständigung der Suchfelder in den entsprechenden Tabs verantwortlich.

4.5 *index.jsp*

Wird aufgerufen, sobald das Tool im Web angesprochen wird und dient der Verwaltung des kompletten Inhalts der Webseite. Es setzt also die verschiedenen Klassen, jsp-Dateien und JavaScript-Dateien zusammen.

4.6 LGDEditTool-Paket

Ist das Hauptpaket, dem die anderen Pakete, die Javacode enthalten, untergeordnet sind.

4.6.1 Functions

Ist eine Klasse in der die folgenden zwei Felder und drei Methoden enthalten sind.

Attribut	Methode	Funktion
MAIN_BRANCH: String		Speichert den View des Main-Branches für die Autovervollständigung in der Suche. Hier mit lässt sich leicht feststellen, auf welchen View der User gerade arbeitet. Stimmt dieser Wert mit `User.getView()` überein, arbeitet dieser auf dem Main-Branch, ansonsten auf seinem eigenen Branch.
PRIVATE_re-CAPTCHA_KEY: String		Ist für die Kommunikation zwischen dem Server und der Seite erforderlich.
PUBLIC_re-CAPTCHA_KEY: String		Wird im Code eingebunden.
	shortenURL(url: String): String	Kürzt URLs zur besseren Darstellung in den Tabellen.
	showTimestamp (timestamp: String): String	Formatiert den `timestamp` der Historytabellen in ein zur Anzeige besseres Formates.
	getTimestamp(): String	Generiert einen aktuellen Timestamp im Format YYYY-mm-ddTHH:MM:ss und gibt dieses zurück. Dies wird in allen Historytabellen benötigt, um den genauen Zeitpunkt der Änderung zu dokumentieren.

4.6.2 Email-Paket

Enthält die folgenden drei Klassen

Email:

Methode	Funktion
setProperties(host: String, port: String, from: String, password: String, starttls: boolean, ssl: boolean)	Setzt die Werte des Senders.
send(to: String, subject: String, text: String, html: boolean): boolean	Erzeugt einen User.

EmailYahoo:

Methode	Funktion
setProperties(from: String, password: String)	Setzt die Werte des Senders.

EmailLGD:

Methode	Funktion
setProperties()	Setzt die Werte des Senders.
sendPasswordForgotten(to: String, user: String, hash: String):boolean	Sendet eine Email mit einem Link zum erstellen eines neuen Passworts.
sendSignUP(to: String, user: String): boolean	Sendet eine Willkomens-Nachricht.

4.6.3 SiteHandling-Paket

Enthält die folgenden beiden Klassen

RequestHandling:
Enthält die Methode `doRequestHandling(resquest: HttpServletRequest, response: HttpServletResponse): String`, die die Auswertung der Formular (K/KV/Datatype/Literal-Mappings mit den Operationen Edit/Delete/Restore/Commit/Clear/Create) behandelt, dazu werden aus den Formulardaten SQL-Querys generiert. Bei erfolgreicher Abarbeitung eines Formulars wird eine Bestätigung zurück gegeben.

User:
Ist eine Singleton-Klasse, das heißt es gibt nur eine Instanz von dieser Klasse, sie repräsentiert somit genau einen User.

Methode	Funktion
getInstance(): User	Wenn eine Instance besteht, wird diese zurückgegeben, anderenfalls wird eine neue erstellt.
createUser(request: HttpServletRequest): void	Erzeugt einen User.
createUser(username: String): void	Erzeugt einen User.
createUser(username: String, loggedIn: boolean, admin: boolean): void	Erzeugt einen User.
getUsername():String	Gibt Usernamen (Primary Key der lgd_user-Tabelle, also die Email-Adresse) des aktuellen eingeloggten Benutzers zurück, ansonsten einen leeren String.
getView(): String	Gibt den View der Autovervollständigung zurück. Stimmt dieser mit dem Wert von `Functions.MAIN_BRANCH` überein, so arbeitet der User auf dem Main-Branch, andernfalls auf seinem eigen Branch.
isLoggedIn(): boolean	Gibt an ob der User eingeloggt ist oder nicht.
isAdmin(): boolean	Gibt an ob der User Administrator ist oder nicht.
logout(): void	Loggt den User aus.
createCookie(response: HttpServletResponse): void	Erzeugt zwei Cookies, welche zum einen den Username speichert, zum anderen ob der User eingeloggt ist. Beide Werte werden in MD5-Hashwerte umgewandelt.
updateView(newView: String): void	Ändert den View auf den übergebenen wert und speichert diesen in der Datenbank. Dadurch wird der Branch geändert, auf dem der User arbeitet.

4.6.4 Templates-Paket

Für jeden Tab gibt es ein Template. Diese Templates generieren einen String für jeden Tab, der den HTML-Code für dessen Darstellung beinhaltet. Diese werden aus vorherigen Eingaben mit Hilfe von SQL-Querys generiert.

Templates:

Methode	Funktion
showMessage(message:String): String	Dient zur Ausgabe von Nachrichten an den User.
branch(message:String):String	Liefert den Namen des Branches auf dem zur Zeit gearbeitet wird mit einem Link um diesen zu ändern.

TemplatesAllMappings:

Methode	Funktion
listAllMappings(type:String, site:String):String	Generiert HTML-Code, welcher zur Darstellung der Mappings im *All Mappings*-Tab dient.

TemplatesOntology:

Methode	Funktion
ontology(search:String):String	Generiert HTML-Code, welcher zur Darstellung der Mappings im *Ontology*-Tab dient.

TemplatesSearch:

Methode	Funktion
search(): String	Generiert Html-Code für die Inputbox der Suche.
searchResult(search:String): String	Generiert HTML-Code um das Resultat der Suchanfrage darzustellen.

TemplatesEditHistory:

Methode	Funktion
editHistory(ksite: String, kvsite: String, search: String): String	Generiert HTML-Code, welcher zur Darstellung der Mappings im *Edit History*-Tab dient.
search(): String	Generiert HTML-Code für die Inputbox der Suche.

TemplatesUnmappedTags:

Methode	Funktion
unmappedTags(ksite: String, kvsite: String, search: String): String	Generiert HTML-Code, welcher zur Darstellung der Mappings im *Unmapped Tags*-Tab dient.
search(): String	Generiert HTML-Code für die Inputbox der Suche.

TemplatesAccountSettings:

Methode	Funktion
accountSettings(setting: String, search: String): String	Generiert Html-Code, welcher zur Darstellung des *Account Settings*-Tab dient.

TemplatesEditedMappings:

Methode	Funktion
listEditedMappings(type: String, site: String): String	Generiert HTML-Code, welcher zur Darstellung aller geänderten Mappings im Userspace dient.

4.6.5 db-Paket

Enthält eine Klasse für die Verbindung mit einer PostgreSQL-Datenbank sowie eine davon abgeleitete Singleton-Klasse, die alle Werte beinhaltet um eine Verbindung mit Bremen-Datenbank enthält, diese können später dann durch die Werte für die LGD-Datenbank ersetzt werden.

DatabasePostgreSQL:

Methode	Funktion
DatabasePostgreSQL()	Konstruktor, überprüft ob die Treiber vorhanden sind
getDriver(): boolean	Gibt `true` zurück, wenn Treiber existiert, ansonsten `false`.
connect(database: String, user:String, passwd: String, ssl: boolean): boolean	Stellt eine Verbindung zur Datenbank her, gibt `true` zurück, wenn eine neue Verbindung erfolgreich aufgebaut wurde. Andernfalls wird `false` zurück gegeben. Wenn es zu einem Verbindungsfehler kommt wird eine `SQLException` geworfen.
disconnect(): void	Trennt eine bestehende Verbindung mit der Datenbank.
execute(command: String): Object[ ][ ]	Führt die angegebenen SQL-Query aus und gibt die angeforderten Werte zurück. Sollte es zu einem Fehler kommen wird eine `SQLException` geworfen.

DatabaseBremen:

Methode	Funktion
DatabaseBremen()	Konstruktor, überprüft ob die Treiber vorhanden sind
static synchronized getInstance(): DatabaseBremen	Gibt eine aktuelle Instanz der Klasse zurück.
connect(): boolean	Stellt eine Verbindung zur Datenbank her, gibt `true` zurück, wenn eine neue Verbindung erfolgreich aufgebaut wurde. Andernfalls wird `false` zurück gegeben. Wenn es zu einem Verbindungsfehler kommt wird eine `SQLException` geworfen.
createView (username: String)	erstellt einen neuen View für den angegebenen Nutzer
createViewHistory (username: String)	erstellt einen neuen View für die Edit-History
createView Unmapped (username: String)	erstellt einen neuen View für die unmapped Tags
getAdministatorEmailAddresses(): String[][]	Gibt alle Email-Adressen und Usernamen der Administratoren zurück.

5 Beschreibung einzelner Funktionen

5.1 Edit-History

In der Edithistory werden alte Werte der einzelnen Mappings nach einer Änderung gespeichert. Diese Änderungen umfassen eine der fünf folgenden Operationen: Editierung, Löschen, Commit, Restore oder Create. Wird ein Mapping mit eine der vorherigen Änderungen bearbeitet, werden die aktuellen Werte des Mappings in die entsprechende History-Tabelle gespeichert und die neuen Werte in die entsprechende Mapping-Tabelle. Die History-Tabellen verfügen über das Attribut `userspace`, welches angibt zu welchem Branch der entsprechende Eintrag gehört. Alle mit `main` versehenen Einträge gehören zum Main-Branch, alle anderen Einträge enthalten den Usernamen des Eigentümers des entsprechenden Branches.
Weiter werden das Änderungsdatum und ein Kommentar gespeichert. Jeder Eintrag verfügt über ein Feld, `action`, welches angibt, von welcher Änderung die Art war, die zu diesem Eintrag geführt hat. Zuletzt hat jeder Eintrag noch eine `history_id`, welche auf einen vorherigen Eintrag in der Historytabelle dieses Mappings verweist. Diese `history_id` ist leer, wenn für ein Mapping noch keine Änderung stattgefunden hat. Durch diese Speicherung lässt sich die Editierungsgeschichte eines Mappings nachverfolgen. Es existiert noch ein weiteres Feld, welches nur für den Mainbranch relevant ist, in diesem wird gespeichert welcher Nutzer für einen Eintrag verantwortlich ist, im Userbranch stimmt dieses Feld mit dem Feld `userspace` überein.

5.2 Userspace

Alle Mappingtabellen die verwendet werden, wurden um ein Attribut erweitert, das angibt, welcher User der Eigentümer des entsprechenden Eintrages ist. Es handelt sich um einen Fremdschlüssel auf den Benutzernamen. Standardmäßig ist dieses Feld auf `main` gesetzt, dadurch wird angezeigt, dass der Eintrag im Mainbranch liegt. Ändert oder erstellt ein User nun ein Mapping, so wird in der entsprechenden Mappingtabelle ein neuer Eintrag erstellt, der die geänderten Werte enthält. Hat es sich bei der Änderung um eine Löschung gehandelt, so werden die Felder `property`, `object` bzw. `language` auf einen leeren String, bei den `datatypes` auf `deleted` gesetzt. Somit wird gekennzeichnet, dass das entsprechende Mapping gelöscht ist. Darauf muss in den Abfragen geachtet werden.
Möchte man nun zum Beispiel alle K-Mappings eines Userspaces auswählen, so sind folgende Bedingungen nacheinander in der Abfrage anzugeben:

1. Es sind alle Einträge des Mainbranches auszuwählen, welche mit den exakt gleichen Werten für alle Felder im Userbranch vorkommen, dadurch wird sichergestellt, dass gecommitete Einträge aus dem Mainbranch und nicht aus dem Userbranch ausgewählt werden.

2. Es sind alle Einträge aus dem Userbranch auszuwählen, die nicht gelöscht sind. Das heißt wie oben beschrieben, wo `property`, `object`, `language`, `datatype` nicht auf Werte gesetzt sind, die angeben, dass sie gelöscht sind. Wobei bereits ausgewählte Einträge nicht erneut auszuwählen sind.

3. Es sind alle Einträge aus dem Mainbranch auszuwählen, welche nicht im Userbranch vorkommen.

Commitet ein User ein Mapping, so werden die Werte für `property`, `object` oder `property`, `language` oder `datatype` in das Mapping des Mainbranches geschrieben. Die Werte des Mainbranches werden vorher in die History gespeichert und mit der Action `commit` versehen.

6 Datenbank

Wir haben den Datentyp lgd_datatype um den Wert *deleted* erweitert. Dieser Wert zeigt an, das ein Datatype-Mapping gelöscht ist.

6.1 lgd_map_datatype

Spalten	Typ	Bedingung	Funktion
k	text	not null	
datatype	lgd_datatype	not null	
user_id	text		zur Identifikation des Userspaces
last_history_id	integer		Verknüpfung mit Wert vor der letzten Änderung des Mappings

Wir haben die Spalten *user_id* und *last_history_id* zu dieser Tabelle hinzugefügt. Des weiteren ist *k* nicht länger Primary Key dieser Tabelle.

6.2 lgd_map_datatype_history

Wir haben diese Tabelle komplett neu erstellt. Sie dient dazu die Edit-History für Datatype-Mappings zu speichern.

Spalten	Typ	Bedingung	Funktion
id	integer	not null	Primärschlüssel zur Identifikation der Änderung des Mappings
k	text	not null	Key des Mappings
datatype	lgd_datatype	not null	Datentyp des Mappings
user_id	text		Benutzername, welcher die Änderung durchgeführt hat
comment	text		Kommentar des Nutzers zur Änderung
timestamp	text	not null	Zeitpunkt der Änderung
action	text		Art der Änderung (Editieren, Löschen, etc.)
userspace	text		zur Identifikation des Userspaces
history_id	text		Verknüpfung mit Wert vor der letzten Änderung des Mappings

6.3 lgd_map_label

Spalten	Typ	Bedingung
k	text	not null
v	text	not null
language	char 16	not null
label	text	not null

6.4 lgd_map_literal

Spalten	Typ	Bedingung	Funktion
k	text	not null	
property	text	not null	
language	text	not null	
user_id	text		zur Identifikation des Userspaces
last_history_id	integer		Verknüpfung mit Wert vor der letzten Änderung des Mappings

Wir haben die Spalten *user_id* und *last_history_id* zu dieser Tabelle hinzugefügt. Des weiteren ist *k* nicht länger Primary Key dieser Tabelle.

6.5 lgd_map_literal_history

Wir haben diese Tabelle komplett neu erstellt. Sie dient dazu die Edit-History für Literal-Mappings zu speichern.

Spalten	Typ	Bedingung	Funktion
id	integer	not null	Primärschlüssel zur Identifikation der Änderung des Mappings
k	text	not null	Key des Mappings
property	text	not null	Property des Mappings
language	text	not null	Sprachcode
user_id	text		Benutzername, welcher die Änderung durchgeführt hat
comment	text		Kommentar des Nutzers zur Änderung
timestamp	text	not null	Zeitpunkt der Änderung
action	text		Art der Änderung (Editieren, Löschen, etc.)
userspace	text		zur Identifikation des Userspaces
history_id	text		Verknüpfung mit Wert vor der letzten Änderung des Mappings

6.6 lgd_map_resource_k

Spalten	Typ	Bedingung	Funktion
k	text	not null	
property	text	not null	
object	text	not null	
user_id	text		zur Identifikation des Userspaces
last_history_id	integer		Verknüpfung mit Wert vor der letzten Änderung des Mappings

Wir haben die Spalten *user_id* und *last_history_id* zu dieser Tabelle hinzugefügt.

6.7 lgd_map_resource_k_history

Wir haben diese Tabelle komplett neu erstellt. Sie dient dazu die Edit-History für K-Mappings zu speichern.

Spalten	Typ	Bedingung	Funktion
id	integer	not null	Primärschlüssel zur Identifikation der Änderung des Mappings
k	text	not null	Key des Mappings
property	text	not null	Property des Mappings
object	text	not null	Objekt des Mappings
user_id	text		Benutzername, welcher die Änderung durchgeführt hat
comment	text		Kommentar des Nutzers zur Änderung
timestamp	text	not null	Zeitpunkt der Änderung
action	text		Art der Änderung (Editieren, Löschen, etc.)
userspace	text		zur Identifikation des Userspaces
history_id	text		Verknüpfung mit Wert vor der letzten Änderung des Mappings

6.8 lgd_map_resource_kv

Spalten	Typ	Bedingung	Funktion
k	text	not null	
v	text	not null	
property	text	not null	
object	text	not null	
user_id	text		zur Identifikation des Userspaces
last_history_id	integer		Verknüpfung mit Wert vor der letzten Änderung des Mappings

Wir haben die Spalten *user_id* und *last_history_id* zu dieser Tabelle hinzugefügt.

6.9 lgd_map_resource_kv_history

Wir haben diese Tabelle komplett neu erstellt. Sie dient dazu die Edit-History für KV-Mappings zu speichern.

Spalten	Typ	Bedingung	Funktion
id	integer	not null	Primärschlüssel zur Identifikation der Änderung des Mappings
k	text	not null	Key des Mappings
v	text	not null	Value des Mappings
property	text	not null	Property des Mappings
object	text	not null	Objekt des Mappings
user_id	text		Benutzername, welcher die Änderung durchgeführt hat
comment	text		Kommentar des Nutzers zur Änderung
timestamp	text	not null	Zeitpunkt der Änderung
action	text		Art der Änderung (Editieren, Löschen, etc.)
userspace	text		zur Identifikation des Userspaces
history_id	text		Verknüpfung mit Wert vor der letzten Änderung des Mappings

6.10 lgd_user

Wir haben diese Tabelle komplett neu erstellt. Sie dient dazu die Edit-History für KV-Mappings zu speichern.

Spalten	Typ	Bedingung	Funktion
email	text	not null	E-Mailadresse des Nutzers
username	text		Benutzername des Nutzers
password	text		Passwort des Nutzers
admin	boolean	not null	Flag ob Admin oder nicht
view	text		View der Autovervollständigung

6.11 lgd_user_main & lgd_user_main_history & lgd_user_main_unmapped & User-Views

Für eine gute funktionierende Autovervollständigung der einzelnen Tabs werden für den Mainbranch und den Userbranch Views benötigt. Die drei Views `lgd_user_main`, `lgd_user_main_history`, `lgd_user_main_unmapped` gehören zum Mainbranch. Der View `lgd_user_main` ist für das Such-Tab, `lgd_user_main_history` ist für das Edit-History-Tab, `lgd_user_main_unmapped` ist für das Unmapped-Tags-Tab zuständig. Jeder dieser Views besteht aus einem Union-All über die vier verwendeten Mapping-Tabellen.
Für jeden User der sich anmeldet, werden diese drei Views erstellt. Diese heißen dann: `lgd_user_USERNAME`, `lgd_user_USERNAME_history`, `lgd_user_USERNAME_unmapped`.