tele-TASK Webplattform
Benutzer- und Rechteverwaltung

Rami Eid-Sabbagh, Stephan Müller, Christian Tinnefeld

Konzepte und Methoden der Web-Programmierung 2005
bei Prof. Dr. sc. nat. Meinel
Hasso-Plattner-Institut für Softwaresystemtechnik

{rami.eidsabbagh.stephan.mueller,christian.tinnefeld}@hpi.uni-potsdam.de

Dieses Paper beschreibt die Konzeption, Implementierung und Verwendung einer Benutzer- und Rechteverwaltung für die tele-TASK Webplattform des Lehrstuhls für Internet-Technologien und –Systeme am Hasso Plattner Institut an der Universität Potsdam. Das tele-Task System kann verwendet werden für tele-Teaching, E-Learning, Konferenzen, Firmentraining oder Produkt Präsentationen. Aus dem großen Einsatzbereich des tele-TASK Systems ergeben sich vielfältige Anforderungen an die Benutzer- und Rechteverwaltung, die durch den Entwurf eines speziellen Konzepts zum Rechtemanagement gedeckt werden.

1 Kontext

Das Projekt tele-TASK Webplattform Benutzer- und Rechteverwaltung wurde im Rahmen der Veranstaltung Konzepte und Methoden der Web-Programmierung entworfen und implementiert. Hierbei handelt es sich um eine Teilkomponente der tele-TASK Webplattform des Lehrstuhls für Internet- Technologien und Systeme. Anhand dieses Systems können Vorlesungen aufgenommen und auch live über das Internet angeschaut werden, wobei gleichzeitig die Foliensätze angezeigt werden, die während einer Vorlesung benutzt werden. Weiterhin können die aufgenommenen Vorlesungen auch zu einem späteren Zeitpunkt über das Internet angeschaut werden. Die bestehende Webplattform [1] soll in der Vorlesung „PHP Webprogrammierung“ mit weiteren Funktionalitäten ausgestattet werden. Gedacht ist eine Kommunikationsplattform um die aufgezeichneten und live gesendeten Vorlesungen herum zu schaffen, um das Lernen und den studentischen Austausch zu fördern und zu vereinfachen.

Mögliche Funktionen sind z.B. Chatsessions für offene Fragen, die Möglichkeit während einer Vorlesung oder Übung zu diskutieren, Notizfunktionen mit Zeitstempel anzulegen oder die Einrichtung eines Forums. Die tele-TASK Webplattform hat einen großen Benutzerkreis. Um diesen einzubeziehen und die Einhaltung von Sicherheitsaspekten in Bezug auf Zugriffsrechte zu gewährleisten, muss eine Benutzerverwaltung implementiert werden, mit der man die Zugriffsrechte verschiedener Benutzer und Benutzergruppen verwalten und definieren kann.

1.1 Struktur der Ausarbeitung

Diese Ausarbeitung hat folgende Struktur: in Kapitel 2 wird das Konzept erläutert, auf dem das Projekt basiert. Kapitel 3 beschreibt die Implementierung des Projekts durch PHP Code und MySQL Datenbank, wobei Kapitel 3.2 detailliert auf Klassen und Methoden eingeht und Kapitel 3.3 die Datenbankstruktur beschreibt. In Kapitel 4 wird die Verwendung des Systems beschrieben, einmal aus Sicht des Administrators und einmal aus der Sicht eines anderen Teilprojekts, die quasi auf die hier implementierten Funktionen zugreifen wollen. Kapitel 8 enthält einen Index.

2 Konzept

2.1 Anforderungen

Die Benutzerverwaltung soll flexibel, erweiterbar und anpassbar sein, sowie die Erstellung fein granularer Rechte und deren Überprüfung ermöglichen. Für jeden Datentyp (Contenttype, siehe Abschnitt 2.3) sollen mögliche Aktionen definiert werden können, die auf ihm ausgeführt werden können. Jedem Benutzer oder jeder Benutzergruppe soll die individuelle Erlaubnis oder das Verbot erteilt werden können, diese Aktionen auszuführen. Für spezielle Datentypen sollen jederzeit neue Aktionen erstellt werden können, damit das System erweiterbar bleibt und leicht neuen Anforderungen angepasst werden kann. Über die Möglichkeit Benutzern Rollen zuzuordnen soll die Bedienbarkeit der Benutzerverwaltung einfach gehalten werden, so dass der Administrator der den Benutzern ihre Rechte zuteilt, nicht für jeden neuen Benutzer alle Rechte neu setzen muss.

Die Benutzerverwaltung soll personenbezogene Information wie z.B. Username, Name und Emailadresse speichern. Die Benutzerverwaltung soll Methoden zur Abfrage von Benutzerrechten, Benutzerrollen und Benutzerinformationen bereitstellen.

Um mit den nötigen anderen Teilkomponenten der tele-TASK Webplattform zu interagieren, soll ein Interface für die anderen Module bereitgestellt werden. Dies soll durch die Erstellung eines globalen Userobjekts zur Nutzung der Funktionen durch andere Gruppen geschehen.

Als Datenbank soll mindestens mySQL 4.1 verwendet werden und die Datenbankanbindung soll über ADODB laufen. Als Programmiersprache soll PHP 5.0 verwendet werden. Die Benutzerpasswörter sollen durch einen beliebig austauschbaren Algorithmus verschlüsselt werden. Optional ist die Erstellung eines HTML basierten Administrationsmenus wünschenswert.

2.2 Evaluierung bestehender Benutzerverwaltungen

Für die Entwicklung eines Lösungsmodells wurden bestehende Benutzerverwaltungen analysiert. Die Zugriffskontrolle der Unixsysteme, das Benutzer-Gruppe-Andere-Modell, reduziert alle möglichen Aktionen auf schreiben, lesen und ausführen. Das Modell ist einfach zu handhaben und auch für große Benutzerkreise geeignet. Der Ersteller einer Datei hat Standardmäßig alle drei Rechte. Er kann bestimmen welche Aktionen Zugehörige einer bestimmten Gruppe ausführen dürfen und was der Rest der Welt, die Anderen, mit seiner Datei machen darf. Jedoch ist dies immer nur auf die drei Aktionen lesen, schreiben und ausführen beschränkt. Zwischenstufen gibt es hier nicht. Diese sind jedoch erwünscht. Z.B. soll ein User bestimmte Dateien lesen dürfen, aber nicht kopieren oder nicht auf Ihnen schreiben, aber Kommentare hinzufügen dürfen.

Bei Microsoft wird jedem Betriebssystemobjekt (Dateien, Dienste, Prozesse usw.) eine Sicherheitsbeschreibung (Security Descriptor) beigefügt. Der Security Descriptor kann eine Access Control List enthalten. Diese beschreibt welche Aktionen Nutzer oder Benutzergruppen auf dem entsprechenden Betriebssystemobjekt auszuführen dürfen. Mögliche Aktionen sind Ändern, Lesen, Schreiben, Ausführen und das Auflisten von Ordnerinhalten. Man kann beliebig viele Gruppen erschaffen und jeden Nutzer belieb vielen Gruppen zuordnen. Es gilt, dass ein explizites Verbot stärker ist als eine explizite Erlaubnis. Auch dieses Modell ist auf die Aktionen Ändern, Lesen, Ausführen, Schreiben und Auflisten limitiert und konnte somit nicht als Vorlage für unseren Entwurf dienen. Eine weitere Möglichkeit besteht darin, dass man alle Nutzern, die verfügbaren Objekte, sowie gültige Aktionen in einer großen Matrix erfasst. Dies Modell bietet eine maximale Granularität, allerdings ist die Bedienbarkeit, insbesondere bei einer größeren Anzahl von Nutzern äußerst unkomfortabel.

Um den gestellten Anforderungen gerecht zu werden, haben wir ein eigenes Konzept entworfen, welches in Kapitel 2.3 erläutert wird. Wir haben uns bewusst gegen die Standardvarianten der Benutzerverwaltung entschieden, da diese meist die Anforderung der Anpassbarkeit nicht erfüllen oder nicht fein genug einstellbar sind.

2.3 Konzept im Detail

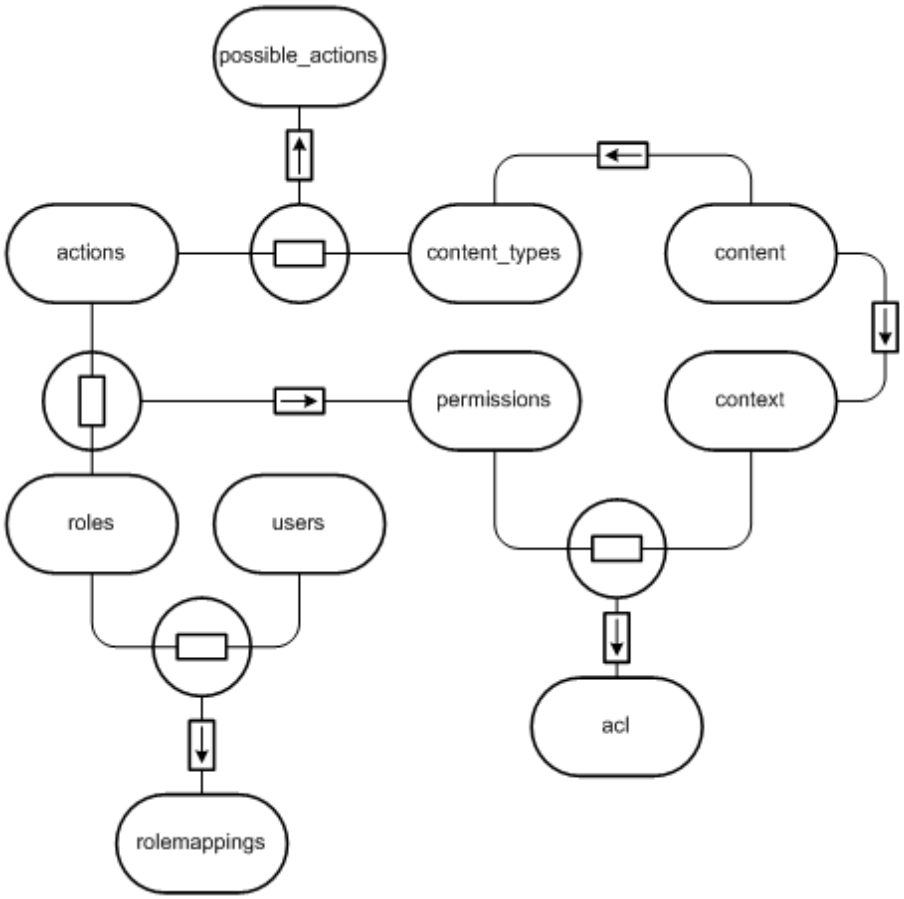

Abbildung 1 : E/R Diagramm des Konzepts

In unserem Konzept kann man Benutzer erstellen und Ihnen Rollen zuordnen. Es ist möglich Rollen (roles) zu definieren. Diese Rollen können z.B. Administrator, Gast oder Student sein, aber auch Rollen wie Student_LehrveranstaltungXY oder Forumsadministrator. Neben diesen Rollen ist es möglich beliebige Aktionen wie z.b. Lesen, Schreiben, Veröffentlichen, Abspielen oder Sperren zu definieren (actions). Die Verknüpfung von Rollen und Aktionen resultiert in Berechtigungen (permissions). In Permissions wird definiert welche Rolle prinzipiell welche Aktionen ausführen darf. Folglich ist eine Permission eine zweistellige Verknüpfung von Rolle und Aktion. Diese Verknüpfung wird nun auf Context angewandt in der Access Control List (acl). Ein Context kann z.B. eine Lehrveranstaltung sein oder ein bestimmter Bereich im Forum.

Das bedeutet, erst wenn die Rolle Assistent prinzipiell über die Aktionen Schreiben und Lesen verfügt und diese Berechtigungen in der Access Control List der Lehrveranstaltung Mathematik II zugeordnet werden, kann ein Benutzer, der der Rolle Assistent zugeordnet ist, im Context Mathematik II lesen und schreiben. Eine Permission bezeichnet nur die prinzipielle Möglichkeit, das eine Rolle bestimmte Aktionen ausführen darf, in welchem konkreten Context diese dann tatsächlich ausgeführt werden dürfen, wird durch Zuordnung von Permission und Context in der Access Control List festgelegt.

Einen Context wiederum kann man sich als einen Container vorstellen, dem weitere Elemente (Content) zugeordnet sind. Unter Content versteht man einzelne Objekte, die einen eindeutigen Contenttyp haben(content_types). So ist in einem möglichem Szenario Mathematik II ein Context, dem acht verschiedene Objekte zugeordnet sind (Content) und bei diesen acht Objekten handelt es sich um vier Powerpointfolien (Contenttyp) und vier Adobe PDF Dateien (Contenttyp). Des weiteren kann man zuordnen, welche Aktionen auf welchen Contenttypen zulässig sind (possible_actions). So kann man definieren, dass die Aktion abspielen auf dem Contenttyp Video angewandt werden darf, allerdings nicht auf dem Contenttypen Powerpointfolie.

In Anbetracht unseres Konzepts wird folgende Erstellung von Rollen empfohlen: es sollten Grundtypen von Rollen erstellt werden wie z.B. Gast, Student, Dozent. Diesen Rollen ordnet man dann die gewünschten Actions zu und definiert in der ACL wo diese angewandt werden dürfen. Sind in einem bestimmten Context besondere Rechte erforderlich, z.B. soll ein Student im Gegensatz zu anderen Studenten in einem Context gleichzeitig auch Schreiben dürfen, wird dies über die Erstellung einer neuen Rolle realisiert. Es wird beispielsweise die Rolle „Mathematik II – Tutor“ erstellt, die dann die passenden Rechte erhält. Diese Rolle wird dann nur dem Benutzer zugeordnet, der bereits über die Rolle Student verfügt und die erweiterten Rechte erhalten soll.

Dadurch hat man eine flache Hierarchie von Rollen, da ohne direkte Vererbung gearbeitet wird. Es ist möglich auf einer fein granularen Ebene Rechte zu erstellen und gleichzeitig ist die Bedienung und Verwaltung benutzerfreundlich, da ein Großteil der Rechte über allgemeine Rollen verwaltet werden kann.

Zur Verdeutlichung des Konzepts wird nachfolgend die Abfrage zur Überprüfung von Rechten erläutert.

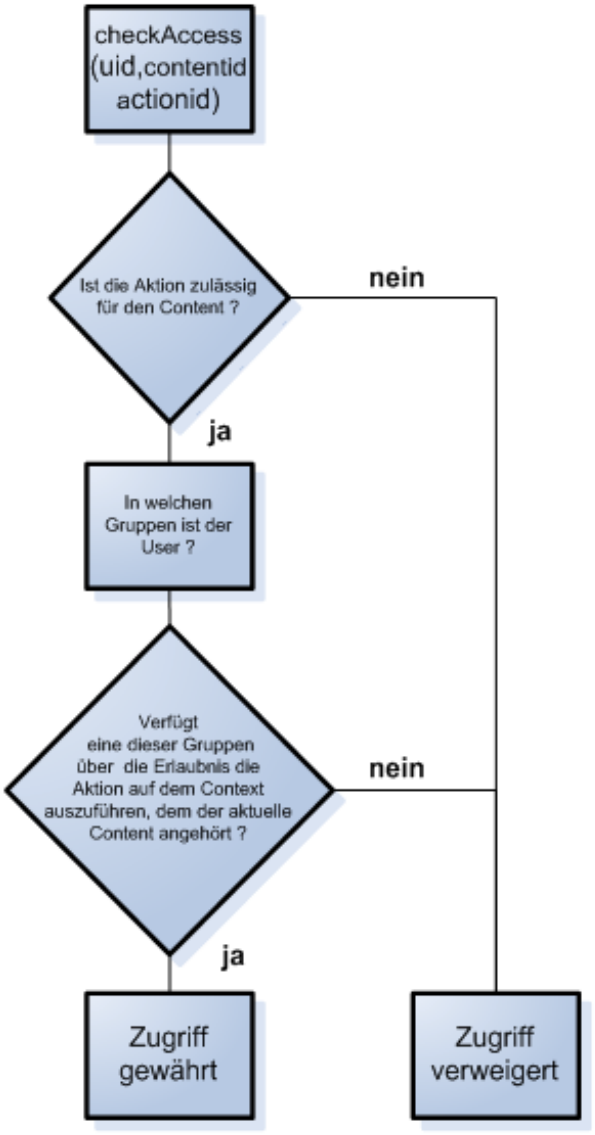

Abbildung 2 : Ablaufdiagramm der Rechteüberprüfung

Bei der Überprüfung, ob ein bestimmter Benutzer (uid) eine bestimmte Aktion (actionid) auf einem bestimmten Content (contentid) ausführen darf, wird zunächst überprüft, ob die gewünschte Aktion prinzipiell zulässig ist für den gewünschten Content. Ist dies nicht der Fall, wird der Zugriff sofort verweigert und die Abfrage ist beendet. Die Überprüfung, ob die Aktion zulässig für den Content ist, erfolgt aus Gründen der Performance zuerst. Danach wird kontrolliert, welchen Gruppen der Benutzer zugeordnet ist. Verfügt eine dieser Gruppen über die Erlaubnis diese Aktion auf dem Context auszuführen, dem der aktuelle Content angehört, wird der Zugriff gewährt, ansonsten wird er verweigert.

3 Implementierung

Bei der Implementierung ging es um die Abbildung des Konzepts auf Software und Datenbank. Die Datenbankstruktur ergibt sich 1:1 aus dem E/R Diagram in Abbildung 1. D.h. für jede Entität wurde eine Tabelle angelegt, was zur Folge hat, dass es die Tabellen acl, actions, content, content_types, context, permissions, possible_actions, roles und users gibt. Die genauen Attribute der einzelnen Datenbanken werden in Kapitel 3.3 vorgestellt. Die Beziehungen zwischen den einzelnen Tabellen werden über die auszuführen Methoden gewährleistet.

3.1 Klassenbeziehungen und Aufbaustruktur

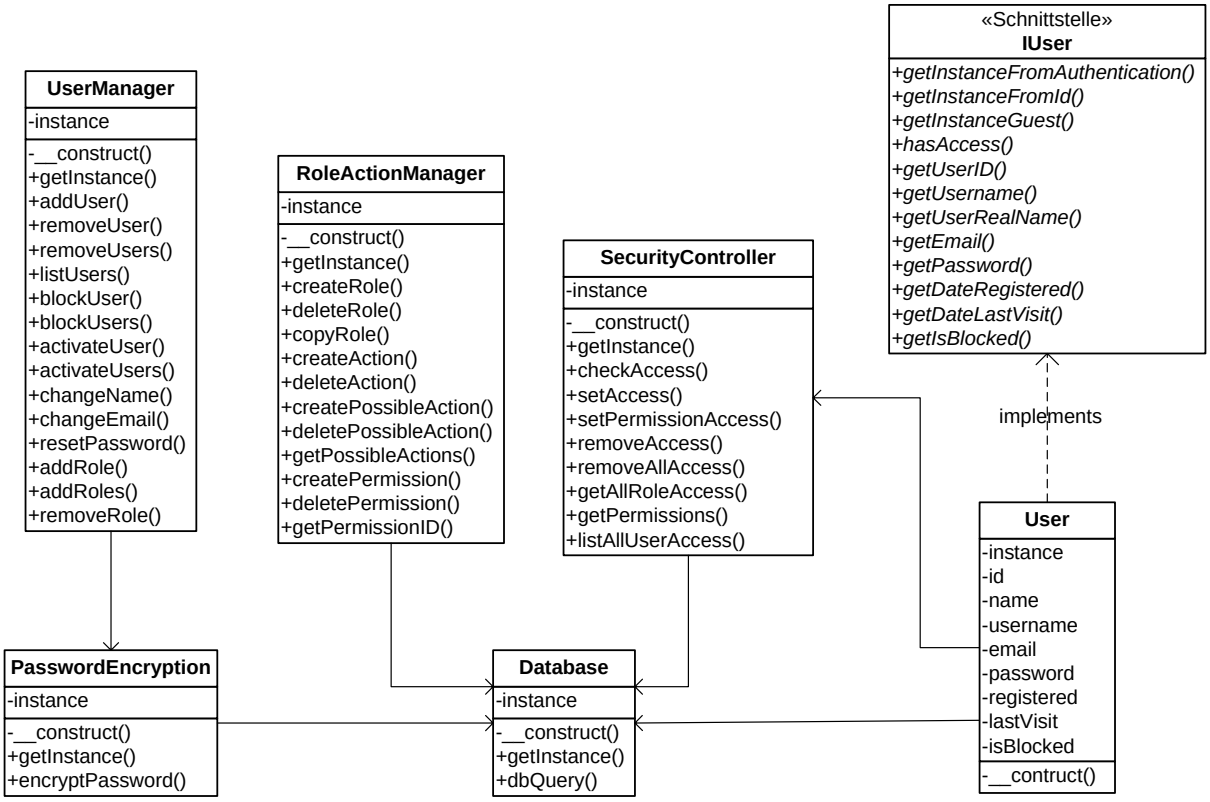

Abbildung 3 : Klassendiagrammm

3.2 Die einzelnen Klassen

3.2.1 Database Klasse

Die Klasse Database stellt die Verbindung zur Datenbank her und führt die Anfragen an die Datenbank aus. Der Konstruktor der Klasse ist private und verhindert somit, dass die Klasse direkt exemplarisiert werden kann. Sie hat zwei Methoden, getInstance() und dbQuery($string, $array).

- static public getInstance()

  - Diese Methode erstellt ein Exemplar der Klasse Database durch den Aufruf new Database(). Über diese Methode können sich andere Klassen ein Objekt der Klasse Database holen und die Methode dbQuery($string, $array) der Klasse Database ausführen, um Datenbankanfragen auszuüben.
- dbQuery($string, $array)
  - Übergabeparameter sind ein String und ein Array. Der String beinhaltet den Sql Anfrage Text. Das Array die nötigen Variablen um die Sql Anfrage zu vervollständigen. Hier wurde bewußt der Anfragetext von den Variablen getrennt um Mißbrauch durch Sql-Statement Injektions zu verhindern, da die User die Möglichkeit haben werden gewisse personenbezogenen Daten selber zu ändern. Die Sql Datenbanken übernehmen somit die Prüfung der vom User eingegebenen Daten und schützen sich vor bösartigen Angriffen.
  - Die Methode verfüght über eine Try und catch Block.
  - Im try-Block wird ein neues Datenbankverbindungsobjekt erstellt durch den Aufruf der Methode newAdoConnection('mysql'). Der Übergabeparameter `mysql` gibt den Typ der Datenbank an. In diesem Fall handelt es sich um eine mySQL Datenbank. Die Verbindung zur Datenbank wird durch den Aufruf der Methode Connect("localhost", "root", "", "teletaskdb") aufgebaut. Wobei die Übergabeparameter die Daten für Hostadresse (hier:localhost), Benutzername für die Datenbank (hier:root), Passwort (hier:kein Passwort), und Name der Datenbank (hier:teletaskdb) bedeuten. Bei Integration in die tele-TASK Webplattform müssen diese Parameter entsprechen geändert werden. Die Datenbankanfrage wird durch den Aufruf der Methode Execute($string, $array) ausgeführt. Das Ergebnis der Anfrage wird in der Variable $result gespeichert und ist auch der Rückgabewert der Methode dbQuery(). Misslingt die Ausführung des try-Blocks fängt der catch-Block dies ab und gibt ein false zurück. Somit weiss der Rufer der Methode, dass sein Anfrage gescheitert ist.
    Bei Bedarf kann durch das Unkommentieren der Kommandos
    ```
    //var_dump($e);
    //adodb_backtrace($e->gettrace());
    ```
    eine detailierte Fehlermeldung auf dem Bildschirm ausgegeben werden. .

Da die Benutzerverwaltung hauptsächlich über Datenbankanfragen funktioniert wird diese Klasse sehr häufig von den anderen Klassen benutzt.

3.2.2 UserManager Klasse

Die Klasse UserManager stellt die nötigen Methoden zur Verfügung, um nutzerbezogene Aktionen auszuführen, wie z.B. das Hinzufügen eines neuen Benutzers, das Setzen persönlicher Daten wie Name, Emailadresse, Paßwort usw. Die Klasse hat zehn Methoden addUser($name, $username, $useremail, $psw, $role), removeUser($username), blockUser($username), activateUser($username), changeName($name, $username), changeEmail($email, $username), addRole($role, $username), removeRole($role, $username), changePassword($username, $oldpwd, $newpwd), und resetPassword($username, $newpwd). Diese Klasse wird hauptsächlich vom Administrator über ein Graphical User Interface benutzt werden.

- static public getInstance()

  - Durch den Aufruf dieser Methode wird ein Exemplar der Klasse UserManager erstellt. Über diese Methode können sich andere Klassen ein Objekt der Klasse UserManager holen und somit die Methoden der Klasse UserManager ausführen.
- addUser($name, $username, $useremail, $psw, $role)
  - Zunächst wird überprüft, ob die Angabe von Vor- und Nachname erfolgt ist, ob ein Benutzername mit fünf oder mehr Zeichen angegeben worden ist, ob ein Passwort mit fünf oder mehr Zeichen eingegeben worden ist und ob eine gültige Email Adresse (mit @ Zeichen) eingetragen worden ist. Wenn diese Bedingungen nicht erfüllt sind, wird die Anfrage sofort abgebrochen und false als Ergebnis zurück geliefert. Wenn bei dem Versuch die Daten in die Datenbank einzutragen festgestellt wird, dass der Benutzername bereits vergeben ist, resultiert die Datenbankanfrage false und die Methode liefert ebenfalls false zurück.
  - Übergabeparameter sind fünf strings die persönliche Daten, sowie Passwort und Rolle des hinzuzufügenden Benutzers beinhalten. Die Parameter $name, $username, $useremail, $psw werden benutzt, um einen neuen Datenbankeintrag zu erstellen. Hierfür wird die Methode dbQuery($query, $array) aufgerufen und ihr die oben genannten Variablen in einem Array übergeben. Zu erwähnen ist auch, dass hier das Blockflag auf null gesetzt wird. D.h. der neu eingetragene User wird aktiviert.
  - Um dem User eine Rolle zuzuordnen wird zuerst die UserID des Benutzers anhand seines Namens wiederum durch eine Datenbankanfrage ermittelt. Die UserID wird benötigt um dem User eine Rolle zuzuordnen. Um dies zu tun wird eine weitere Datenbank Anfrage erstellt, die userid und role als Übergabeparameter hat. Auch diese Anfrage führt zu einem neuen Eintrag in der Datenbank.
  - Der Rückgabewert der Methode addUser() ist das Ergebnis der letzten Datenbankanfrage, also der Zuordnung des Benutzers zu einer Rolle. Es kann auch ein false als Ergebnis der Fehlerbehandlung zurückgegeben werden. Nur wenn alle Einträge erfolgreich sind, wird true als Ergbnis dieser Methode zurückgegeben.
- removeUser($username)
  - Übergabeparameter ist ein String, der Name des Benutzers der aus der Datenbank gelöscht werden soll. Dies geschieht durch eine Datenbankanfrage über die Methode dbQuery($query, $array), wobei der Datenbankeintrags des Benutzers aus der Datenbank gelöscht. Der Rückgabewert der Methode ist das Ergebnis der Datenbankanfrage.
- removeUsers($usernames)
  - Übergabeparameter ist ein Array, das mehrere zu löschende Usernamen beinhaltet. Die Methode hat dieselbe Funktion wie removeUser(), nur beziegt sie sich auf mehrere User. Sie wurde definiert, um dem Amdinistrator die Arbeit zu erleichtern.
- listUsers()
  - Diese Methode durchläuft einmal die ganze users Tabelle und listet alle eingetragenen User auf. Sie gibt ein Array mit allen Usernamen zurück.

- blockUser($username)
  - Übergabeparameter ist ein String, der Username des Benutzers der geblockt werden soll. Dies geschieht durch eine Datenbankanfrage über die Methode dbQuery($query, $array), wobei Datenbankeintrag des Benutzer aufgefrischt wird und sein Blockflag auf 1 gesetzt wird. Der Rückgabewert der Methode ist das Ergebnis der Datenbankanfrage.
- blockUser($usernames)
  - Übernimmt dieselbe Funktion wie die Methode blockUser(), nur auf mehere User bezogen. Sie wurde ebenfalls definiert, um Administratortätigkeiten zu erleichtern.
- activateUser($username)
  - Übergabeparameter ist ein String, der Username des Benutzers der aktiviert werden soll. Dies geschieht durch eine Datenbankanfrage über die Methode dbQuery($query, $array), wobei Datenbankeintrag des Benutzer aufgefrischt wird und das Blockflag auf 0 gesetzt wird. Der Rückgabewert der Methode ist das Ergebnis der Datenbankanfrage.
- activateUsers($usernames)
  - Diese Methode übernimmt dieselbe Funktion wie die Methode activateUser(), nur bezieht sie sich auf mehrere User. Übergabeparameter ist ein Array, das alle Usernamen von den Usern beinhaltet, die aktiviert werden sollen. Auch sie wurde definiert, um Administratortätigkeiten zu erleichtern.
- changeName($name, $username)
  - Übergabeparameter sind zwei Strings, Name und Username. Auch hier wird zunächst die Benutzereingabe auf Richtigkeit überprüft und bei Nichterfüllen wird die Methode sofort mit false als Rückgabe beendet. So muss der Username mindestens 5 Zeichen enthalten. Name beinhaltet den neuen Namen des Benutzers. Anhand des Usernamens wird in der Datenbank der richtige Datenbankeintrag gefunden und der alte Namen gegen den neuen Namen ausgetauscht. Dies wird durch eine Datenbankanfrage über die Methode dbQuery($query, $array) gewährleistet. Die Änderung des Namens geschieht durch ein eine Sql-Update Statement. Der Rückgabewert der Methode ist das Ergebnis der Datenbankanfrage.
- changeEmail($email, $username)
  - Übergabeparameter sind zwei Strings, Email und Username. Auch hier wird zunächst die Benutzereingabe auf Richtigkeit überprüft und bei Nichterfüllen wird die Methode sofort mit false als Rückgabe beendet. So muss die Email Adresse ein „@“- Zeichen enthalten. Email beinhaltet die neue Emailadresse des Benutzers. Anhand des Usernamens wird in der Datenbank der richtige Datenbankeintrag gefunden. Die Änderung des Namens geschieht durch ein Sql-Update Statement, mit Hilfe von der Datenbankanfrage über die Methode dbQuery($query, $array). Der Rückgabewert der Methode ist das Ergebnis der Datenbankanfrage.
- addRole($role, $username)
  - Übergabeparameter sind Role und Username. Role ist ein Integer und beinhaltet einen Pointer auf die entsprechende Rolle. Um dem User eine neue Rolle zuzuordnen wird zuerst die UserID des Benutzers anhand seines Namens

durch eine Datenbankanfrage ermittelt. Die UserID wird benötigt, um dem User in der nächsten Datenbankanfrage eine Rolle zuzuordnen. Ihre Übergabeparameter sind userid und role. Der Rückgabewert der Methode addRole() ist das Ergebnis der letzten Datenbankanfrage, also der Zuordnung des Benutzers zu einer Rolle.

- addRoles($roles, $username)
  - Diese Methode übernimmt dieselbe Funktion wie die Methode addRole(), jedoch wird als Übergabeparameter ein Array übergeben, das alle Rollen, denen der User zugeordnet werden soll, beinhaltet. Somit kann ein User anstatt einer, gleichzeitig mehreren Rollen zugeordnet werden.
- removeRole($role, $username)
  - Übergabeparameter sind Role und Username. Role ist ein Integer und beinhaltet einen Pointer auf die entsprechende Rolle. Um die Beziehung eines User zu einer Rolle zu beenden, müssen die entsprechenden Datenbankanfragen ausgeführt werden, d.h. die Datenbankeinträge löschen, die die Beziehung darstellen. Zuerst wird wieder die UserID des Benutzers anhand seines Namens durch eine Datenbankanfrage ermittelt. Die UserID wird benötigt, um die Beziehung zwischen dem User und der Rolle zu löschen. Die Datenbankanfrage läuft über die Methode dbQuery($query, $array). Diese Anfrage führt zum Löschen eines Eintrags in der Datenbank.
  - Der Rückgabewert der Methode removeRole() ist das Ergebnis der letzten Datenbankanfrage, die den entsprechen Eintrag aus der Tabelle „rolemappings“ der Datenbank löscht.
- removeRoles($roles, $username)
  - Diese Methode übernimmt die gleiche Funktion wie die Methode removeRole(), jedoch wird ihr als Übergabeparameter ein Array übergeben, das alle Rollen, von denen der User entfernt werden soll, beinhaltet. Somit kann ein User anstatt aus einer, gleichzeitig aus mehreren Rollen entfernt werden. Dies erleichtert Verwaltungsaufgaben.
- getRoles($userid)
  - Diese Methode hat einen Integer als Übergabeparameter. Sie ruft alle Rolle ab, denen ein User, repräsentiert durch die userid, zugeordnet ist. Sie gibt ein Array mit den entsprechenden Rollen zurück.
- resetPassword($username, $newpwd)
  - Übergabeparameter sind Username und neues Paßwort. Durch ein Sql.Update-Statement wird das alte Paßwort durch das neue Paßwort ersetzt. Im Gegensatz zur Methode changePassword($oldpwd, $newpwd) wird hier nicht nach dem alten Paßwort gefragt. Das Zurücksetzen des Paßworts geschieht über eine Datenbankanfrage durch die Methode dbQuery($query, $array). Rückgabewert ist das Ergebnis der Datenbankanfrage.

Der Aufbau der Methoden der Klasse UserManager ist sehr ähnlich. Er besteht hauptsächlich aus Datenbankanfragen, die über die Methode dbQuery($query, $array) der Klasse Datenbank gelöst werden. Die Anfragen bestehen immer aus dem SQL Statement, ein String der in der Variable $query gespeichert wird und einem array das ein oder mehrere Übergabeparameter beinhaltet.

3.2.3 RoleActionManager Klasse

Die Klasse RoleActionManager ist für die Erstellung und das Löschen neuer Rollen, Aktionen oder auch möglichen Aktionen zuständig. Sie ist zur Verwaltung von Rollen, Aktionen und möglichen Aktionen da. Sie hat zwölf Methoden der RoleActionManager{} wird vom Administrator über ein Graphical User Interface benutzt werden. Der Konstruktor der Klasse ist private und verhindert somit, dass die Klasse direkt exemplarisiert werden kann. Dies ist ein Resultat der Verwendung des Singleton Patterns, welches sicherstellt, dass es zu einem Zeitpunkt systemweit maximal ein Exemplar der Klasse gibt.

- static public getInstance()
  - getInstance() exemplarisiert die Klasse RoleActionManager durch den Aufruf new RoleActionManager(). Über diese Methode können sich andere Klassen ein Objekt holen und auf die Methoden der Klasse RoleActionManager zugreifen.
- createRole($rolename)
  - Übergabeparameter ist rolename. Er gibt den Namen der neu zu erstellenden Rolle an. Er wird der Methode dbQuery($query, $array) für die Datenbankanfrage übergeben. Ein neuer Datenbankeintrag mit dem übergebenen Rollennamen wird in der Datenbanktabelle roles erstellt. Die Methode createRole($rolename) gibt das Ergebnis der Datenbankanfrage zurück. Ist ein Rollenname schon vorhanden, kann dieser aufgrund des UNIQUE constraints in der Datenbank nicht erneut angelegt werden und die Methode gibt somit false zurück.
- deleteRole($rolename)
  - Übergabeparameter ist rolename. Anhand des Rollennamens wird der entsprechende Datenbankeintrag in der Tabelle roles gefunden und gelöscht. Die Datenbankanfrage läuft über die Methode dbQuery($query, $array). Rolename wird der Anfrage über das Array $array übergeben. Rückgabewert der Methode deleteRole($rolename) ist das Ergebnis der Datenbankanfrage.
- copyRole($originalrolename, $newrolename)
  - Die Methode copyRole() kopiert eine Rolle mitsamt ihrer Beziehungen und Rechte. Es wird ein neuer Eintrag in der Datenbanktabelle roles gemacht, sowie in die dazugehörigen Tabellen acl und permissions, um die Zugriffsrechte der neuen Rolle zu definieren. Die Kopie der Rolle nimmt den mit newrolename übergebenen Namen an. Ist ein Rollenname schon vorhanden, kann dieser aufgrund des UNIQUE constraints in der Datenbank nicht erneut angelegt werden und die Methode gibt somit false zurück.
- createAction($action)
  - Übergabeparameter ist action. Er gibt den Namen der neu zu erstellenden Aktion an. Der Aktionsname wird der Methode dbQuery($query, $array) für die Datenbankanfrage übergeben. ergeben. Ein neuer Datenbankeintrag mit besagtem Aktionsnamen wird in der Datenbanktabelle actions erstellt. Die Methode createAction($action) gibt das Ergebnis der Datenbankanfrage zurück. Ist ein Actionname schon vorhanden, kann dieser aufgrund des UNIQUE constraints in der Datenbank nicht erneut angelegt werden und die Methode gibt somit false zurück.
- deleteAction($action)

  - Übergabeparameter ist action. Anhand des Aktionsnamens wird der entsprechende Datenbankeintrag in der Tabelle actions gefunden und gelöscht. Die Datenbankanfrage läuft über die Methode dbQuery($query, $array). Der zu löschende Aktionsname wird der Anfrage über das Array $array übergeben. Rückgabewert der Methode deleteRole($rolename) ist das Ergebnis der Datenbankanfrage.
- createPossibleAction($action, $contenttype)
  - Übergabeparameter sind action und contenttype. Action gibt den Namen der neuen Action an und contenttype definiert den content auf dem die Aktion ausgeführt werden kann. Beide Parameter werden dbQuery($query, $array) in einem array für die Datenbankanfrage übergeben. Ein neuer Datenbankeintrag mit besagtem Aktionsnamen und contenttype wird in der Datenbanktabelle possible_actions erstellt. Die Methode createPossibleAction($action, $contenttype) gibt das Ergebnis der Datenbankanfrage zurück.
- deletePossibleAction($action, $contenttype)
  - Übergabeparameter sind action und contenttype. Die Datenbankanfrage läuft über die Methode dbQuery($query, $array). Anhand beider Parameter wird der entsprechende Datenbankeintrag aus der Tabelle possible_actions gefunden und gelöscht. Die Methode deletePossibleAction($action, $contenttype) gibt das Ergebnis der Datenbankanfrage zurück.
- getPossibleActions($contentid)
  - Diese Methode gibt die Namen aller möglichen Aktionen an, die prinzipiell auf einem Content ausgeführt werden dürfen. Zurückgegeben wird ein Array mit den entsprechenden Aktionen.
- createPermission($roleid, $actionid)
  - Mit dieser Methode wird eine Permission erstellt. Eine Permission ist eine Beziehung zwischen roleid und action darstellt. Es wird ein Datenbankeintrag in der Tabelle permissions getätigt.
- deletePermission($roleid, $actionid)
  - Mit dieser Methode werden die Datenbankeinträge in den Tabellen acl und permissions gelöscht. Dass in der Tabelle acl gegebenenfalls auch Einträge gelöscht werden müssen liegt an der Beschaffenheit der Datenbank. Das Fremdschlüsselprinzip wird nicht unterstützt und so muss man der Programmierer selbst auf die constraints achten. Auf die Tabelle acl wird zugegriffen da eine Permission schon einem oder mehreren Kontext zugeordnet sein kann und diese Zuordnung in diesem Moment auch gelöscht werden muss.
- getPermissionId($roleid, $actionid)
  - Die Methode erfragt die Id einer Permission mit Hilfe der Übergabeparameter roleid und actionid. Rückgabewert ist ein Integer, die Id der Permission. Die Datenbankanfrage greift auf die Tabelle permissions zu in der die role und actionpaare eingetragen sind.

3.2.4 SecurityController

Die Klasse SecurityController verwaltet alle Aufgaben bezüglich der Zugriffsrechte. Sie stellt Methoden zur Erstellung und Beseitigung von Zugriffsrechten für Rollen in Bezug auf Aktion und Kontext zur Verfügung. Der SecurityController wird von dem User Objekt gerufen, um zu prüfen ob ein Benutzer die Erlaubnis hat eine bestimmte Aktion auszuführen. Über den SecurityController wird überprüft, welche Permissions auf welchem Context gegeben sind. Die Klasse hat einen Konstruktor und acht Methoden. Der Konstruktor ist private und verhindert somit, dass die Klasse direkt exemplarisiert werden kann. Dies ist ein Resultat der Verwendung des Singleton Patterns, welches sicherstellt, dass es zu einem Zeitpunkt systemweit maximal ein Exemplar der Klasse gibt.

- static public getInstance()
  - getInstance() exemplarisiert die Klasse SecurityController durch den Aufruf new SecurityController(). Über diese Methode können sich andere Klassen ein Objekt holen und auf die Methoden der Klasse SecurityController zugreifen.
- checkAccess($userid, $contentid, $actionid)
  - (siehe auch 2.3) Übergabeparameter sind userid, contentid und actionid. Diese drei Parameter werden für die Datenbankanfragen verwendet. Um herauszubekommen, ob ein Benutzer Zugriff über eine bestimmte Aktion auf einen Content hat, werden drei Datenbankanfragen ausgeführt. Die Datenbankanfrage wurde absichtlich in drei Teile unterteilt, um große Joins und den daraus resultierenden hohen Rechenaufwand zu vermeiden. Die erste Datenbankanfrage greift auf die Tabellen possible_actions und content zu. Wobei geschaut wird, ob die Aktion auf dem Content überhaupt definiert ist. Ist dies nicht der Fall, werden die zweite Abfrage und dritte Abfrage nicht mehr ausgeführt und die Methode checkAccess() gibt false zurück. Ist die Aktion auf dem Content definiert werden die Rollen, denen der User angehört, abgefragt. Diese werden in einem Array gespeichert, über das in einer foreach-Schleife iteriert wird. In jedem Schleifenschritt wird eine Datenbankanfrage ausgeführt. Bei dieser Abfrage wird geprüft, ob eine der Rollen, die der Benutzer inne hat, Zugriff auf den übergebenen Content hat. Ist dies der Fall gibt die Methode checkAccess() true zurück und dem Benutzer wird der Zugriff gewährt. Ist dies nicht der Fall wird gibt die Methode checkAccess() false zurück. Dies bedeutet, dass dem Benutzer der Zugriff verwährt wird.
- setAccesss($roleid, $contextid, $actionid)
  - Übergabeparameter sind roleid, contextid und actionid. Anhand der Parameter wird ein neuer SecurityContext geschaffen. Eine Aktion wird mit einer Rolle, repräsentiert durch die roleid und mit einem Kontext, repräsentiert durch die contextid in Beziehung gebracht. Bei dieser Datenbankanfrage wird ein neuer Eintrag in die Tabellen acl und permissions gemacht.
- setPermissionAccess($permissionid, $contextid)
  - Mit permissionid und contextid als Übergabeparameter wird ein Datenbankeintrag in der Tabelle acl gemacht. Rückgabewert ist ein Boolean.
- removeAccess($roleid, $contextid, $actionid)
  - Übergabeparameter sind roleid, contextid und actionid. Um den richtigen Eintrag aus der acl Tabelle der Datenbank zu löschen, wird der Eintrag im SQL-Statement anhand der drei Variablen roleid, contextid und actionid

gesucht. Die Datenbankanfrage geschieht durch den Aufruf der Methode dbQuery($query, $array). Rückgabewert ist das Ergebnis der Datenbankanfrage.

- removeAllAccess($roleid, $contextid)
  - Diese Methode löscht alle Zugriffsrechte einer Rolle auf einem bestimmten Context. Übergabeparameter sind roleid und contextid. Um dies zu erreichen werden zwei Datenbankanfragen ausgeführt. Die erste ruft alle Einträge mit der entsprechenden roleid aus der Tabelle permissions ab. Mit Hilfe dieser Liste werden dann in einer Schleife die dazugehörigen Einträge aus der Tabelle acl entfernt.
- getPermissions($contextid)
  - Die Methode getPermissions($contextid) listet alle Aktionen auf, die in dem übergegebenen Kontext ausgeführt werden können. Sie greift auf die Tabellen roles, acl, actions und permissions zu. Als Ergebnis gibt sie alle Rollen aus und darüber hinaus welche Aktionen die jeweilige Rolle auf dem Context ausführen darf.
- listAllUserAccess($userid)
  - Diese Methode ist auch für die graphische Ausgabe bestimmt. Sie listet alle Aktionen auf, die ein Benutzer auf den im System definierten Kontexten ausführen darf. Sie greift auf die Tabellen context, permissions, rolemappings und actions zu. Diese Methode wurde in Bezug auf die von uns entworfene Administrationsoberfläche implementiert und sollte nur in der Absicht benutzt werden eine graphische Ausgabe zu erzeugen.
- listAllRoleAccess($roleid)
  - Auch diese Methode ist für die graphische Ausgabe bestimmt. Sie listet alle Aktionen auf, die die als Parameter übergebene Rolle auf die im System definierten Kontexte ausführen darf. Sie greift auf die Tabellen context, permissions und actions zu. Diese Methode ist abgestimmt auf die von uns entworfene Administrationsoberfläche und sollte nur in der Absicht benutzt werden eine graphische Ausgabe zu machen.

3.2.5 IUser Interface

Das Interface IUser gibt folgende Methoden vor:

- static getInstanceFromId($id)
- static getInstanceFromAuthentication($username, $password)
- static getInstanceGuest()
- hasAccesss()
- getUserID()
- getUserName()
- getUserRealName()
- getEmail()
- getPassword()
- getDateRegistered()
- getDateLastVisit()
- getIsBlocked()
- changePassword($oldpwd, $newpwd)

- changeEmail($email)

Diese Methoden wurden von der Projektgruppe „Page Controller“ gewünscht, um den Security Controller nutzen zu können.

3.2.6 User Klasse

Die Klasse User implementiert das Interface IUser. Es gibt vierzehn Methoden vor, die implementiert werden müssen. Das User Objekt kann durch drei Methodenaufrufe exemplarisiert werden, getInstanceFromId($id), getInstanceFromAuthentication($username, $password) und getInstanceGuest(). Über das User Objekt können auf die unterschiedlichen Methoden zugegriffen werden.

Die Klasse User hat einem Konstruktor und implementiert die vierzehn Methoden vom IUser Interface.

Die Konstruktormethode _construct bekommt einen Integer id als Parameter übergeben. Anhand dieser ID erstellt sie ein neues User Objekt und setzt ID, Name, Emailadresse, Password, das Datum des letzten Besuchs des Users und das Blockflag. Die Konstruktormethode gibt ein User Objekt zurück. Folgende Methoden können gerufen werden.

- static getInstanceFromId($id)
  - Übergabeparameter ist ein Integer id. Mit dem Aufruf new User($id) wird ein neues User Objekt erstellt, das auch den Rückgabewert der Methode darstellt.
- static getInstanceFromAuthentication($username, $password)
  - Übergabeparameter sind zwei Strings username und password. Username wird benutzt, um mit einer Datenbankanfrage das verschlüsselte Paßwort des Users aus der Datenbank zu erhalten. Daraufhin wird das aus der Datenbank erfragte verschlüsselte Paßwort mit dem durch den Übergabeparameter übermittelte Paßwort verglichen, welches zuvor durch die Methode encryptPassword($password) verschlüsselt wurde. Sind die Paßwörter gleich wird ein neues User Objekt mit den Daten des Benutzers erstellt und zurückgegeben. Bei ungleichen Paßwörtern wird ein User Objekt mit Guest Daten erstellt, das dann den Rückgabewert bildet.
- static getInstanceGuest()
  - Diese Methode wird benutzt, um nicht registrierte Benutzer als Gast anzumelden. Sie erstellt ein User Objekt mit dem Aufruf new User(0). 0 als Übergabeparameter referenziert auf die Guest Einstellungen in der Datenbanktabelle wo ein spezieller User-Eintrag mit der Id 0 vorhanden ist. Entsprechend der Attribute des Gast-Eintrags in der Datenbank wird ein Gast-Objekt erzeugt. Die Methode getInstanceGuest() gibt ein User Objekt zurück.
- hasAccesss(`$contentid`, `$actionid`)
  - Übergabeparameter sind zwei Integer contentid und actionid. Zuerst wird ein SecurityController Objekt angefordert, um die Methode checkAccess(self::$id, $contentid, $actionid) des SecurityControllers aufzurufen. CheckAccess liefert ein Boolean zurück. Dieser Rückgabewert wird weitergereicht und ist auch der Rückgabewert der Methode hasAccess(`$contentid`, `$actionid`). Diese

Methode sollte immer dann gerufen werden, wenn ein Zugriff auf ein Objekt stattfindet.

- getUserID()
  - Hat keinen Übergabeparameter. Rückgabewert ist die ID des aktuellen User Objektes.
- getUserName()
  - Hat keinen Übergabeparameter. Rückgabewert ist die der Benutzername mit dem der Nutzer sich beim Login in die tele-TASK Webplattform anmeldet. Der UserName wird aus dem aktuellen User Objekt ausgelesen.
- getUserRealName()
  - Hat keinen Übergabeparameter. Rückgabewert ist die der echte Name des Nutzers. Der Name des Nutzers wird aus dem aktuellen User Objekt ausgelesen.
- getEmail()
  - Hat keinen Übergabeparameter. Rückgabewert ist die Emailadresse des Nutzers. Die Emailadresse des Nutzers wird aus dem aktuellen User Objekt ausgelesen.
- getPassword()
  - Diese Methode gibt das verschlüsselte Paßwort des Benutzers zurück.
- getDateRegistered()
  - Diese Methode gibt das Registrierungsdatum des aktuell angemeldeten Benutzers zurück.
- getDateLastVisit()
  - Anhand dieser Methode wird das Datum des letzten Login des Users erfragt und zurückgegeben.
- getIsBlocked()
  - Mit der Methode isBlocked() wird der Status des Users geprüft. Sie gibt einen Boolean wert zurück.
- changePassword($oldpwd, $newpwd)
  - Übergabeparameter sind zwei Strings, das alte und das neue Paßwort. Zuerst wird das alte Paßwort aus der Datenbank abgerufen. Daraufhin wird das aus der Datenbank erfragte verschlüsselte Paßwort mit dem durch den Übergabeparameter übermittelte Paßwort verglichen, das davor durch die Methode encryptPassword($oldpwd) verschlüsselt wurde. Stimmen die alten Passwörter überein wird durch ein Sql.Update-Statement das alte Paßwort durch das neue Paßwort ersetzt. Rückgabewert ist ein Boolean, true bei erfolgreicher Änderung des Paßworts, andernfalls wird false zurückgegeben.
- changeEmail($email)
  - Übergabeparameter ist ein String, der die neue Emailadresse beinhaltet. Über eine SQL-UPDATE Anfrage wird die neue Emailadresse in die Tabelle users der Datenbank übertragen. Rückgabewert der Methode ist ein Boolean.

Die Funktionen des User Objekts können zum Beispiel zum Erstellen personifizierten Benutzerumgebungen genutzt werden.

3.2.7 PasswordEncryption Klasse

Die Klasse PasswordEncryption besteht aus einem Konstruktor und zwei Methoden getInstance() und encryptPassword($password_cleartext). Der Konstruktor ist private und verhindert somit entsprechend dem Singleton Pattern, dass die Klasse direkt exemplarisiert werden kann.

- static getInstance()
  - getInstance() exemplarisiert die Klasse PasswordEncryption durch den Aufruf new PasswordEncryption(). Über diese Methode können andere Klassen auf die PasswordEncryption Klasse zugreifen.
- encryptPassword($password_cleartext)
  - Übergabeparameter ist ein String. Er beinhaltet das Paßwort in Klartext. Über den Methodenaufruf md5($password_cleartext) wird das Paßwort verschlüsselt. Der Rückgabewert der Methode ist das verschlüsselte Passwort.

3.3 Verwendete Datenbankstruktur

Es wurde bei der Datenbank bewusst der Typ MyISAM [2] gewählt, da dieser eine Volltextsuche über den Datenbankinhalt bietet. MyISAM nterstützt das Fremdschlüsselprinzip im Gegensatz zu InnoDB Datenbanken nicht. Dies bringt den Nachteil mit sich, dass bei der Implementierung die Abhängigkeiten verschiedener Tabellen vom Programmierer selbst berücksichtigt werden müssen.

So ist es beispielsweise erforderlich, dass bei der Ausführung der Methode deletePermission der Klasse RoleActionManager nicht nur die betroffenen Einträge in der Tabelle permissions gelöscht werden müssen, sondern darüber hinaus auch die Einträge in der Tabelle acl, die die zu löschenden Permissions verwenden. Würde dies nicht erfolgen, so wäre es z.B. möglich, dass ein Benutzer prinzipiell im System eine bestimmte Aktion nicht mehr ausführen darf, diese Aktion aber im Kontext der Tabelle acl als gültig erfasst ist.

Darüber hinaus entspricht die Struktur der Datenbank dem E/R Konzept, d.h. für jede Entität im E/R Diagram gibt es eine Tabelle in der Datenbank. Im Kapitel 6 findet sich das Datenbankschema.

3.3.8 Tabelle acl

Die Tabelle acl besteht aus den Attributen permission und context. Beide Attribute sind Integerwerte. Permission und Context bilden zusammen den Primärschlüssel.

Auf diese Tabelle greifen die Methode checkAccess(), die Methode setAccess() und die Methode removeAccess() der Klasse SecurityController zu.

3.3.9 Tabelle actions

Die Tabelle actions besteht aus zwei Attributen id und name. Id ist ein Integer und bildet den Primärschlüssel der Tabelle. Name ist ein uniques Characterfeld und beschreibt den Namen der eingetragenen Aktion wie z.B. schreiben oder lesen. Dadurch, dass name unique

ist, wird vorgebeugt, dass zwei Aktionen den gleichen Namen tragen und so verwechselt werden können.

Auf die Tabelle actions greifen die Methoden createAction() und deleteAction() der Klasse RoleActionManager zu.

Bei Veränderung der Tabelle sollte die Content Management Gruppe mindestens die Attribute id, context, content_type und name zur Verfügung stellen, mit id als Primärschlüssel. Name sollte unique sein, um die Unterscheidbarkeit der Aktionen zu gewährleisten.

3.3.10 Tabelle content

Die Tabelle content besteht aus fünf Attributen id, context, content_type, name und description. Id, context und content_type sind Integerwerte. Id bildet den Primärschlüssel der Tabelle. Name und description sind Characterfelder. Name ist der Name des gewünschten content und description gibt eine Beschreibung des contents. Name ist unique gesetzt, um sicher zu stellen, dass keine zwei contents mit dem gleichen Namen erstellt werden.

Auf diese Tabelle greift die Methode checkAccess() der Klasse SecurityController zu.
Die Content Management Gruppe muß mindestens die Attribute id, context, und content_type zur Verfügung stellen, wobei id Primärschlüssel sein sollte.

3.3.11 Tabelle content_types

Die Tabelle content_types besteht aus zwei Attributen id und name. Id ist ein Integer und bildet den Primärschlüssel der Tabelle. Name ist ein Characterfeld. Name ist der Name des content_types.

Auf diese Tabelle greift die Methode checkAccess() der Klasse SecurityController zu.
Die Content Management Gruppe muss mindestens das Attribut id zur Verfügung stellen, das auch den Primärschlüssel der Tabelle darstellen sollte.

3.3.12 Tabelle context

Die Tabelle context besteht aus drei Attributen id, name und description. Id ist ein Integer und bildet den Primärschlüssel der Tabelle. Name und descripion sind Characterfelder. Name ist der Name des contexts. Description gibt eine nähere Beschreibung des contexts.

Auf diese Tabelle greift die Methode checkAccess() der Klasse SecurityController zu.
Diese Tabelle muss mindestens die Attribute id und name zur Verfügung stellen. Id sollte Primärschlüssel sein und name sollte unique sein, um Verwechslungen vorzubeugen.

3.3.13 Tabelle permissions

Die Tabelle permissions besteht aus drei Attributen id, role und action. Id, role und action sind Integer, wobei id den Primärschlüssel der Tabelle bildet.

Auf diese Tabelle greifen die Methoden checkAccess(), setAccess(), removeAccess() der Klasse SecurityController zu.

3.3.14 Tabelle possible_actions

Die Tabelle possible_actions besteht aus zwei Attributen action und content_type. Action und content_type sind Integer. Sie bilden zusammen den Primärschlüssel der Tabelle.

Auf diese Tabelle greifen die Methoden checkAccess() der Klasse SecurityController und die Methoden createPossibleActions() und deletePossibleActions() der Klasse RoleActionManager zu.

Diese Tabelle muss mindestens die Attribute id und name zur Verfügung stellen. Id sollte Primärschlüssel sein und name sollte unique sein, um Verwechslungen vorzubeugen.

3.3.15 Tabelle rolemappings

Die Tabelle rolemappings besteht aus zwei Attributen user und role. User und role sind Integer. Sie bilden zusammen den Primärschlüssel der Tabelle.

Auf diese Tabelle greifen die Methoden checkAccess() der Klasse SecurityController und die Methoden addUser(), addRole() und removeRole() der Klasse UserManager zu.

3.3.16 Tabelle roles

Die Tabelle roles besteht aus zwei Attributen id und name. Id ist ein Integer und bildet den Primärschlüssel der Tabelle. Name ist ein Characterfeld und ist der Name der entsprechenden Rolle. Das Feld name ist unique, um sicherzustellen, dass keine zwei Rollen, den gleichen Namen tragen, somit nicht unterscheidbar sind.

Auf diese Tabelle greifen die Methoden createRole() und deleteRole() der Klasse RoleActionManager zu.

3.3.17 Tabelle users

Die Tabelle users besteht aus acht Attributen id, name, username, email, password, dateRegistered, dateLastVisited und Block. Id ist ein Integerwert und bildet den Primärschlüssel der Tabelle. Name, username, email, password sind Characterfelder. Im Namensfeld ist der Name des entsprechenden eingetragenen Nutzers zu finden, username ist der Username des Nutzers. Das Feld username ist unique gesetzt, um die Unterscheidbarkeit der verschiedenen User zu gewährleisten. So können keine zwei User den gleichen Namen annehmen. Im Feld email ist die Emailadresse des Nutzers zu finden. DateRegistered und dateLastVisited sind Datumswerte und geben das Registrierungsdatum und das Datum des letzen Besuchs der tele-TASK Webplattform der jeweiligen Nutzer an. Block ist ein Booleanwert und ist ein Eintrag der entweder 0 oder 1 gesetzt ist.

Auf diese Tabelle greifen die Methoden addUser(), removeUser(), blockUser(), activateUser(), changeName(), changeEmail(), addRole(), removeRole(), changePassword

und resetPassword der Klasse UserManager und der Konstruktor _construct der Klasse User zu.

3.4 Fehlerbehandlung

Die Fehlerbehandlung in unserem Projekt haben wir auf zwei Ebenen realisiert. Zum einen wird durch Constraints in der Datenbank sichergestellt, dass keine ungewünschten Daten abgespeichert werden. Dies geschieht beispielsweise durch das Constraint UNIQUE. So haben wir in der Datenbanktabelle users sichergestellt, dass keine doppelten Benutzernamen vergeben werden können. Weiterhin wurden folgende Attribute in der Datenbank auf UNIQUE gesetzt: der Actionname, der Rollenname und der Name von möglichen Aktionen.

Darüber hinaus werden Eingaben, die direkt vom Benutzer unseres Admin-Interfaces kommen, explizit überprüft. Hier wurden in den entsprechenden Methoden die zur Übernahme von Benutzereingaben benutzt werden die Eingaben auf Korrektheit überprüft: beispielsweise muss eine Email-Adresse ein „@“ Zeichen enthalten, Benutzername und Password mindestens fünf Zeichen enthalten sowie ein Name angegeben werden.

Beim Anlegen eines neuen Benutzers werden mit folgender Abfrage die Benutzereingaben überprüft:

```
if (strlen($name)<1 || strlen($username)<5 || strlen($psw)<5 ||
      !strstr($useremail,'@')){
    return false;
}
```

Listing 1 : Auszug aus Methode addUser(…)

Bei folgenden weiteren Funktionen wrude Überprüfung von Korrektheit der Benutzereingaben getätigt:

- createAction
- createRole
- copyRole
- changeEmail
- changePassword

4 Verwendung der Benutzer- und Rechteverwaltung

In diesem Kapitel wird die praktische Verwendung der Benutzer- und Rechteverwaltung erläutert. In Kapitel 4.1 werden die administrativen Aufgaben anhand eines selbst implementierten HTML-Adminmenus erläutert. In Kapitel 4.2 wird skizziert wie man mit dem System fein granulare Rechte erstellen kann. In Kapitel 4.3 wird die Verwendung des Userobjekts gezeigt, über das andere Teilgruppen die für sie relevanten Funktionen der Benutzer- und Rechteverwaltung nutzen können.

4.1 Administrative Aufgaben

Im HTML-Adminmenu können Methoden aus User-Manager, RoleActionManager und Security-Controller genutzt werden. Die Menge der über das Adminmenu benutzbaren Funktionen ist nur eine Untermenge der Methoden, die prinzipiell über die o.g. Klassen zur Verfügung gestellt werden. Allerdings ist diese Menge ausreichend um alle notwenigen Administrationsarbeiten durchzuführen.

Analog zu den Klassen ist das Menu in User-Manager, Security-Controller und RoleActionManager unterteilt.

Abbildung 4 : Struktur des Admin-Menus

Darunter gibt es jeweils direkte Links, die zu den entsprechenden Methoden im jeweiligen Untermenu führen.

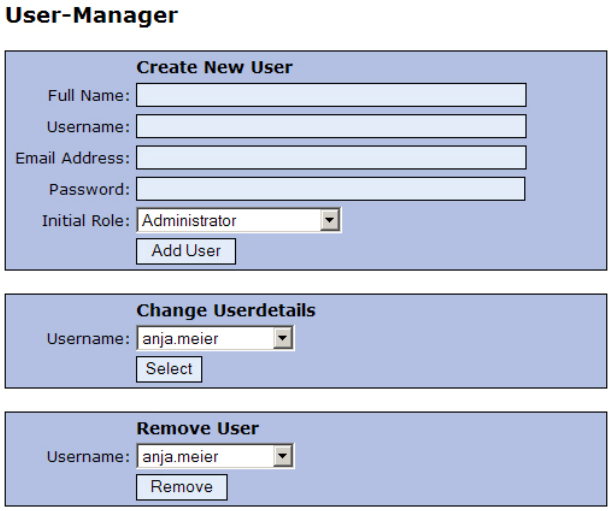

Abbildung 5 : Struktur des User-Managers

In Abbildung 5 wird die Struktur des User-Managers gezeigt. Analog zur verwendeten addRole Methode können hier neue User erstellt werden. Neben der Angabe der Benutzerinformationen kann hier eine initiale Rolle zugeordnet werden. Diese initiale Rolle kann per Drop-Down Menu von den existierenden Rollen ausgewählt werden. Über Remove User kann ein Benutzer entfernt werden, über Change Userdetails gelangt man zu einem

Untermenu, in dem man benutzerspezifische Einstellungen ändern kann.

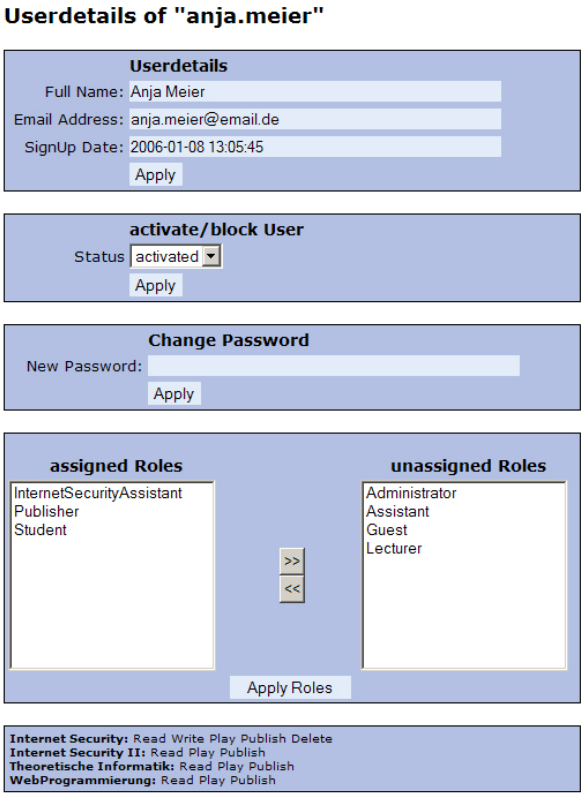

Abbildung 6 : User-Detail Ansicht

Bei den Userdetails kann man sich zu einem Nutzer die individuellen Informationen anzeigen lassen. Man kann einen Nutzer blockieren oder sein Passwort ändern, sowie per SwipeBox verschiedene Rollen zuordnen bzw. bereits zugeordnete Rollen wieder entfernen. Darüber hinaus werden auf der Seite alle gültigen Aktionen angezeigt, die ein User auf einem bestimmten Context ausführen darf.

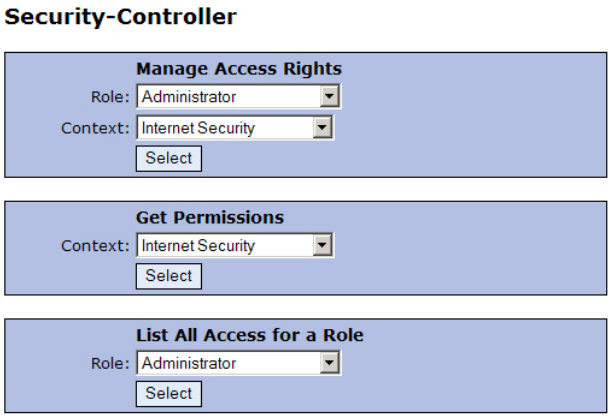

Abbildung 7 : Security Controller

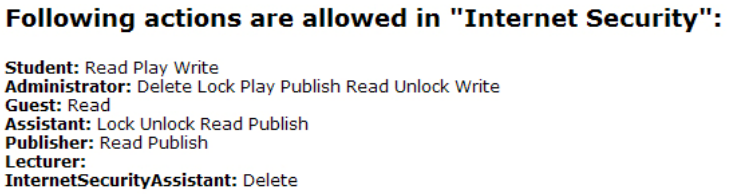

Abbildung 8 : Ausgabe von Get Permissions

In Abbildung 7 wird der Menu Punkt Security Controller gezeigt. Dort hat man die Möglichkeit mit Manage Access Rights zu bestimmen, welche Aktionen eine bestimmte Rolle auf einem bestimmten Context ausführen darf. Des weiteren kann man sich über getPermissions anzeigen lassen, welche Rollen welche Aktionen auf einem bestimmten Context ausführen dürfen. Mit List All Access for a Role bekommt man gezeigt, welche Aktionen eine bestimmte Rolle auf welchem Context ausführen darf.

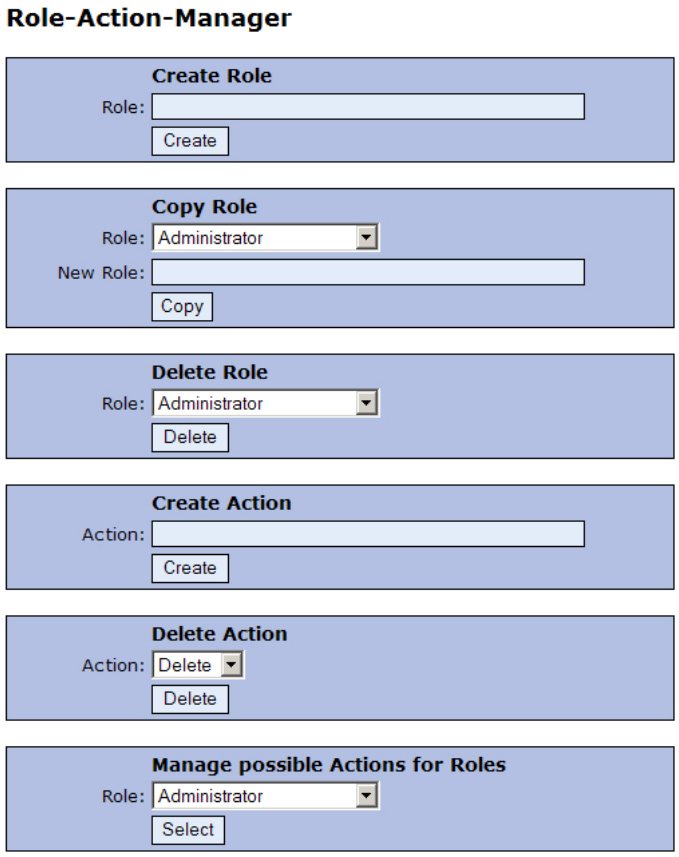

Abbildung 9 : Role-Action-Manager

Im Role Action Manager ist es möglich Rollen zu erstellen, zu kopieren und zu löschen. Aktionen können erstellt und gelöscht werden. Mit Manage possible Actions für Roles kann bestimmt werden, welche Aktionen ein bestimmter Benutzer prinzipiell ausführen darf.

4.2 Individuelle Zuordnung von Rechten

Nehmen wir an, wir haben eine Rolle Student und diese Rolle darf z.B. auf der

Lehrveranstaltung Mathematik II Lesen und Videos anschauen. Nun soll aber ein bestimmter Nutzer gleichzeitig Tutor in Mathematik II sein und deswegen die Rechte veröffentlichen und schreiben haben.

Diese spezielle Anforderung kann dadurch erzielt werden, dass man eine neue Rolle erstellt, die z.B. MathematikII_Tutor genannt wird und der man die Aktionen Veröffentlichen und Schreiben für den Kontext Mathematik II zuordnet. Dann ordnet man dem gewünschten Benutzer neben seiner Rolle Student die Rolle MathematikII_Tutor zu.

4.3 Verwendung des User Objekts

Möchte eine andere Gruppe auf die Funktionen der Benutzer- und Rechteverwaltung zugreifen, so geschieht das über das User Objekt. Das User Objekt kann über das Interface IUser angesprochen werden und wird durch die Klasse User realisiert.

Es stehen drei verschiedene Konstruktoren zur Verfügung: bei getInstanceFromAuthentification(…) werden Benutzername und Passwort übergeben. Dieser Konstruktor ist sinnvoll im Zusammenhang mit einem Anmeldevorgang am System. Des weiteren kann dem Konstruktor getInstanceFromID(…) auch nur die UserID übergeben werden. Scheitert die Exemplarisierung durch zuerst genannte Konstruktoren, z.B. weil ein User geblockt ist, Benutzername und Passwort falsch sind oder weil die UserID nicht vorhanden ist, wird automatisch das Gast User Objekt exemplarisiert durch getInstanceGuest() ohne Angabe von Parametern.

Ist ein User Objekt erstellt worden, werden die internen Variablen mit den Werten des aktuellen Benutzers belegt, auf die dann mittels get Methoden zugegriffen werden kann (z.B. User.getUsername()). Einige Variablen können auch über das User Objekt und damit u.a. auch den Nutzer selbst gesetzt werden, wie z.B. die Email-Adresse oder Vor- und Nachname. Auch eine Änderung des Passwortes unter Angabe des aktuellen Passworts ist möglich.

5 Zusammenfassung

Zusammenfassend lässt sich sagen, dass das hier vorgestellte System einen sehr guten Kompromiss aus der Möglichkeit einer fein-granularen Rechteerstellung und einem überschaubaren Verwaltungsaufwand beschreibt.

Abschließend möchten wir erwähnen, dass uns die Erstellung des Projekts sehr viel Spaß gemacht hat, auch wenn es teilweise zu hitzigen, aber durchaus interessanten und produktiven Diskussionen gekommen ist.

5.1 Offene Aufgaben für die Content Management Gruppe

In unserem Projekt haben wir einige Aufgaben ausgeführt, die eigentlich in die Zuständigkeit der Content Management Gruppe gehören. Da wir aber auf das Vorhandensein einiger Tabellen angewiesen sind, haben wir diese erstellt. Daher nachfolgend eine Liste mit Dingen, die die CM Gruppe überprüfen sollte.

Die Content Management Gruppe sollte Klassen zum Auffüllen, Anzeigen und Löschen der Inhalte der Tabellen actions, content, content_type, context und possible_actions schreiben. Ihre Aufgabe ist es content, content_types und contexts, actions und

possible_actions zu definieren und dem Administrator die Möglichkeit zu geben die Standard Inhalte später den Anforderungen entsprechend zu erweitern.

Weiterhin kann sie die Tabellen erweitern und verändern, solange die oben genannten Mindestanforderungen eingehalten werden..

- Die Tabelle actions muss die vier Attribute id, context, content_type und name haben. Id muss Primärschlüssel sein und name unique.
- Die Tabelle content muss mindestens die Attribute id, context, und content_type zur Verfügung stellen. Id muss Primärschlüssel sein.
- Die Tabelle content_types muss das Attribut id haben, das auch den Primärschlüssel der Tabelle darstellen muss.
- Die Tabelle context muss die Attribute id und name besitzen. Id muss Primärschlüssel sein und name unique.
- Die Tabelle possible_actions muss die Attribute id und name haben. Auch hier muss Id Primärschlüssel sein und name unique.

6 Appendix – DB Schema

acl

Feld	Typ	Null	Standard
permission	int(4)	Ja	0
context	int(4)	Ja	0

action

Feld	Typ	Null	Standard
id	int(4)	Ja	*NULL*
name	varchar(25)	Ja	

content

Feld	Typ	Null	Standard
id	int(10)	Ja	*NULL*
context	int(10)	Ja	0
content_type	int(11)	Ja	
name	varchar(50)	Ja	
description	varchar(200)	Ja	

content_types

Feld	Typ	Null	Standard
id	int(11)	Ja	*NULL*
name	varchar(50)	Ja	

context

Feld	Typ	Null	Standard
id	int(10)	Ja	*NULL*
name	varchar(50)	Ja	
description	varchar(200)	Ja	

permissions

Feld	Typ	Null	Standard
id	int(10)	Ja	*NULL*
role	int(4)	Ja	0
action	int(4)	Ja	0

possible_actions

Feld	Typ	Null	Standard
action	int(11)	Ja	
content_type	int(11)	Ja	

rolemappings

Feld	Typ	Null	Standard
user	int(11)	Ja	
role	int(11)	Ja	

roles

Feld	Typ	Null	Standard
id	int(4)	Ja	*NULL*
name	varchar(25)	Ja	

users

Feld	Typ	Null	Standard
id	int(11)	Ja	*NULL*
name	varchar(50)	Ja	
username	varchar(25)	Ja	
email	varchar(100)	Ja	
password	varchar(100)	Ja	
dateRegistered	datetime	Ja	0000-00-00 00:00:00
dateLastVisit	datetime	Ja	0000-00-00 00:00:00
block	(1)	Ja	

7 Referenzen

[1] tele-TASK Webplattform: http://www.tele-task.de
[2] Christoph Meinel, Harald Sack (Universität Jena): „WWW - Kommunikation, Internetworking, Web-Technologien“, Springer Verlag (2004)
[3] MyISAM Dokumentation: http://dev.mysql.com/doc/refman/4.0/de/myisam.html
[4] mySQL Dokumentation: http://www.mysql.de
[5] PHP Dokumentation: http//www.php.net

8 Index