UNB Components

Dokumentation

UNB Components (Codename: UNB2) ist die Grundlage für eine neue Art Kommunikationsplattform. Es ist eine Art Grundsystem, das Funktionen zum Umgang mit Mitteilungen und deren Klassifizierung bereitstellt, wie sie in Webforen, Wikis, CMS, Fotoalben, Weblogs oder Bug-Trackern angewendet werden. Darauf aufbauend kann eine Oberfläche entwickelt werden, die all diese Fähigkeiten mehr oder weniger spezialisiert zur Verfügung stellt, oder eine bestehende Anwendung um diese Funktionen erweitert werden.

Autor: Yves Goergen
Webseite: http://unclassified.de/go/unb2
Stand: 7. März 2010

Inhalt

1. Worum geht es?

Webforen sind heute weit verbreitet. Wikis und Weblogs sind dann oft nicht mehr weit. Kommt dann noch ein klassisches CMS oder ein Issue Tracker dazu, haben wir so ziemlich alle aktuell bekannten Formen von Kommunikationsplattformen beisammen. Doch das ergibt meist eine Sammlung mehr oder weniger friedlich nebeneinander existierender Programme, die i.d.R. mit ihrer eigenen Bedienoberfläche, Benutzerverwaltung und vielleicht auch noch in unterschiedlichen Programmiersprachen daherkommen. Will man diese Systeme vollständig integrieren, hat man erstmal eine Menge Arbeit vor sich. Später muss man sich noch dazu fragen, ob es so praktisch ist, für jede Art von Informationsverwaltung oder Kommunikation eine separate Anwendung zu verwenden, ist das doch auch mit Einbußen in der Bedienbarkeit und Übersichtlichkeit verbunden.

Das ist die *Ausgangssituation*. Eine logische Lösung für diese Probleme ist meines Erachtens die Entwicklung eines Systems, das die Funktionen der anderen ersetzen kann, indem es Informationen aus unterschiedlichen Formaten vereint und Arbeitsabläufe durch Gemeinsamkeiten in der Bedienung vereinfacht. Wikis enthalten Artikel, die versioniert und mit einem Schlüsselwort adressierbar sind. Webforen enthalten Beiträge, die einen Diskussionsverlauf darstellen und auf die man antworten kann. Weblogs sind im Prinzip ähnlich. Issue Tracker enthalten z.B. Fehlerberichte mit einer Reihe von Attributen und ebenfalls einer Antwortmöglichkeit. All diese Ausprägungen von Software weisen vergleichbare Konzepte auf, die man ebensogut unter ein Dach packen kann. Ein, ich nenne es mal Wikiforum, das Beiträge enthält, die direkt adressierbar sind und so als eigenständige Webseiten verwendet werden können; die versioniert sind, um Änderungen zu verfolgen; die zu Diskussionen zusammengefasst sind und die ohne eine starre Einordnung in einzelne Unterforen frei attributiert werden können. Das wäre dann das *Ziel*.

Der Name „UNB Components“ (Codename: UNB2) verrät die Wurzeln dieses Projekts. Seit April 2003 habe ich das Unclassified NewsBoard 1.x (kurz UNB, zunächst unter anderem Namen) entwickelt und so den Einstieg in die Entwicklung von Webanwendungen und insbesondere Webforen vollzogen. Anfangs hat sich das UNB mehr oder weniger an den Strukturen der damaligen Forenwelt orientiert und sich allein durch eine intuitive Bedienung aus dem Feld abgehoben. Die objektorientierte Architektur war bei anderen Systemen noch wenig verbreitet und auch PHP 4 hat dem enge Grenzen gesetzt. Die Datenbankstruktur hat sich an den Fähigkeiten der MySQL-Versionen vor 4.0 orientiert und ist dementsprechend einfach gehalten. Doch mit der Zeit wachsen die Ideen. Ich habe Foren in der Nähe von Wikis, CMS und anderen Webanwendungen gesehen. Es kam der Wunsch auf, Foren einfach in bestehende Webseiten und Designs zu integrieren. Und schließlich hat sich auch die technische Plattform weiterentwickelt.

UNB2, zunächst als UNB Version 2.0 geplant, sollte deshalb ein kompletter Neuanfang sein. Alle Konzepte wurden von Grund auf hinterfragt, die Ansätze ähnlicher Anwendungstypen miteinander kombiniert und möglichst universell ausgelegt. Dabei wurde schnell klar, dass das daraus resultierende Programmkonzept zu allgemein für eine konkrete Anwendung ist. Aber die Benutzeranforderungen sind ebenso vielfältig. Man kann kaum jemandem eine einheitliche Universallösung anbieten, die in ihrer Komplexität kaum bedienbar wäre. Deshalb wird es auch weiterhin für verschiedene Anforderungen spezialisierte Anwendungen geben – und für diese sollen die UNB Components eine gemeinsame Grundlage sein. Wie der Name schon sagt, handelt es sich dabei um Komponenten, also Bausteine, die gemeinsame Funktionen bereitstellen, die dann unter angepassten Benutzeroberflächen die Kernarbeit verrichten. Dennoch wird auf Basis dieser Komponenten eine „Demo-Anwendung“ entwickelt, die als fertiges Webforum verwendet werden kann.

Aber auch dieses Webforum wird nicht so werden, wie die bekannten Lösungen. Benutzerbeiträge sind anders organisiert, Zugangsregeln funktionieren anders und all das wirkt sich auch auf die Bedienung der Anwendung aus. Eine große Herausforderung hierbei ist es, erfahrene und unerfahrene Benutzer von diesem Konzept zu überzeugen, um die festgefahrenen Strukturen in der Forenwelt aufzubrechen und durch innovative Ideen letztlich die Effizienz und die Benutzerzufriedenheit zu erhöhen.

In dieser Dokumentation werden alle Konzepte der UNB Components vorgestellt und ausführlich beschrieben und Zusammenhänge zur Unterstützung grafisch dargestellt.

2. Programmarchitektur

In diesem Kapitel werden Grundbegriffe erklärt und der generelle Umgang mit dem System beschrieben. Außerdem wird der Aufbau des Programmcodes und der Datenbank dargestellt.

2.1. Konzepte

In diesem Softwaresystem geht es stark vereinfacht dargestellt um die Verwaltung von Mitteilungen. Diese Mitteilungen sind versioniert, über Referenzen verbunden und mit Tags klassifiziert.

2.1.1. Mitteilungen

Eine **Mitteilung** ist ein Beitrag, eine Nachricht oder allgemeiner eine Informationseinheit, die von einem Benutzer mit der Absicht verfasst wird, sie zu veröffentlichen. Eine Mitteilung hat daher immer einen Autor. Da allerdings auch mehrere Autoren an einer Mitteilung, z.B. gemeinsamen Dokumenten, arbeiten können, lässt sich für eine ganze Mitteilung kein einzelner Autor festhalten. Um dennoch einen verantwortlichen Benutzer zuordnen zu können, sprechen wir vom *Besitzer* der Mitteilung. Dieser Besitzer kann im Laufe der Zeit wechseln, wenn sich Verantwortlichkeiten innerhalb einer Organisation ändern. Als Besitzer für neue Mitteilungen wird zunächst der erste Autor festgehalten.

Mitteilungen können außerhalb von Diskussionsverläufen auch als eigenständige Webseiten verwendet werden. Um den Zugriff darauf zu erleichtern und ihnen eine sprechende Überschrift zu geben, lässt sich Mitteilungen ein *Seitenname* zuweisen. Über den sind die Mitteilungen dann wie in einem Wiki direkt aufrufbar.

2.1.2. Referenzen

Um mehrere Mitteilungen in Diskussionen zusammenzufassen und zu ordnen, werden sie mit **Referenzen** verknüpft. Das funktioniert ähnlich wie bei E-Mail-Nachrichten, es genügt aber, eine Referenz auf die unmittelbar beantwortete Mitteilung zu setzen. Da man in einer Mitteilung aber auch auf weitere zusätzliche Mitteilungen Bezug nehmen kann, lassen sich beliebig viele Referenzen angeben. Um die Darstellung der Mitteilungen in einer Baumstruktur zu unterstützen, wird zwischen der *primären Referenz* und den weiteren Referenzen unterschieden. Die primäre Referenz wird direkt für eine Mitteilung gespeichert und hat Auswirkungen auf Moderationseinschränkungen. Weitere Referenzen haben nur informativen Charakter und dienen zur besseren Nachvollziehbarkeit von Angaben aus anderen Mitteilungen.

Was in Webforen bislang Threads waren, sind nun also Referenzen. Eine Mitteilung ohne Referenzen lässt sich so auf den Anfang eines Threads abbilden. Wenn inmitten eines Threads einmal das Thema gewechselt wird, hatte ein Moderator die Möglichkeit, den Thread aufzuteilen. Diese Funktion ist sehr wichtig, da vom ursprünglichen Thema abweichende Mitteilungen oft auftreten. Um aber nicht die ursprünglichen Referenzen der Mitteilung zu verlieren, gibt es das Konzept eines **Diskussionseinstiegspunkts** (kurz DEP). Ein solcher Punkt ist eine spezielle Markierung einer Mitteilung, die aussagt, dass mit dieser Mitteilung ein neues Thema behandelt wird. Die Mitteilung bleibt dabei als Antwort auf ihre primäre Referenz erhalten, stellt gleichzeitig aber einen *Ausgang* aus dem vorherigen Thread dar. Anstatt einen Thread aufzuteilen und die zum neuen Diskussionsverlauf gehörenden Mitteilungen auszuwählen, muss ein Moderator nun nur noch diese DEP-Markierung für eine Mitteilung setzen. Damit alle Antworten automatisch in die neue Diskussion mitwandern, ist es notwendig, dass Antworten immer den richtigen Mitteilungen zugewiesen sind.

2.1.3. Revisionen

Dass Mitteilungen gelegentlich nach ihrer ersten Veröffentlichung nachbearbeitet werden, ist üblich. Benutzer, die bereits die frühere Fassung der Mitteilung gelesen haben, wünschen sich eine Markierung, anhand der sie feststellen können, dass sich der Inhalt verändert hat und sie die Mitteilung ggf. erneut lesen sollten. Gerade bei längeren Texten ist das aber ein mühseliges Unterfangen, erst recht, wenn man keine Unterschiede erkennen kann, da nur ein unscheinbarer Tippfehler verbessert wurde. Nur wenige Benutzer heben ihre Veränderung deutlich hervor und nicht alle Systeme erlauben die Angabe der vorgenommenen Änderungen.

Um jetzt schnell und einfach die Änderungen zu erkennen, werden alle Mitteilungen versioniert. So ist jede frühere **Revision** der Mitteilung einsehbar und mit anderen Revisionen vergleichbar. Mit einer Gegenüberstellung der Änderungen kann man sich das aufmerksame erneute Lesen der Mitteilung sparen und dennoch über den aktuellen Stand informiert bleiben.

Diese Funktion dient ebenfalls der Nachvollziehbarkeit von Änderungen an gemeinsam bearbeiteten Mitteilungen, wie es in Wikis üblich ist. Hierzu kommt noch der Vorteil, dass erkennbar ist, wer wann welche Änderung vorgenommen hat. Einzelne Änderungen können so auch rückgängig gemacht werden, indem ältere Revisionen wiederhergestellt werden. Gegenüber einer reinen Aufzeichnung des Zeitpunkts und Autors der *letzten* Änderung ist das ein großer Vorteil.

Mitteilungsrevisionen sind die eigentlichen Informationsträger von Mitteilungen. Sie enthalten den Textinhalt, den Betreff, Zeitangaben und ggf. auch angehängte Dateien. Jede Mitteilungsrevision hat einen Autor, der unabhängig vom Eigentümer der Mitteilung ist. Außerdem hat jede Revision einen Zeitstempel, der angibt, wann die Revision erstellt wurde. Änderungszeitstempel werden nicht benötigt, da dies durch den Erstellungszeitpunkt einer nachfolgenden Revision festgelegt wird. Dass Revisionen auch angehängte Dateien enthalten hat den Effekt, dass auch Dateien versioniert sind. Außerdem werden die Tags zur Klassifizierung nicht ganzen Mitteilungen, sondern immer einzelnen Revisionen zugewiesen. So können einerseits die Zuweisung von Tags verfolgt werden und andererseits einer neueren Version einer Datei angepasste Attribute wie z.B. EXIF-Daten eines Fotos zugewiesen werden. Letztlich verfügt eine Revision noch über ein Feld, in dem Änderungen zur vorherigen Revision notiert werden können, um die Änderungen in einer Revisionsliste schneller auffindbar zu machen oder einen Überblick über alle Änderungen einer Mitteilung zu geben.

2.1.4. Tags

Um Mitteilungen inhaltlich zu klassifizieren, werden **verwaltete Tags** (etwa: Schlagwörter) verwendet. Diese kombinieren die Eigenschaften klassischer Foren und freier Verschlagwortung durch den Benutzer. Foren sind wie Schubladen – eine Mitteilung kann sich nur in einer Schublade befinden, nie in mehreren gleichzeitig. Tags anstelle von Foren haben aber den entscheidenden Vorteil, dass Mehrfachzuordnungen möglich sind. Während die sogenannten Cross-Postings in Mailing-Listen und Newsgroups nicht so richtig gerne gesehen wurden, kommt es in der Realität doch immer wieder vor, dass eine Mitteilung (bzw. das ganze Thema) eigentlich mehreren Kategorien einer bestehenden Ordnungshierarchie zugeordnet werden müsste. Weblogs machen es uns vor, dort ist die Zuweisung von einer oder mehreren Kategorien zu den Artikeln gang und gäbe, und mit dem Wissen um eine solche Struktur lassen sich auch darauf abgestimmte Kategorie-Einteilungen (hier Tags) finden. Anschaulich könnte man sagen, wir ersetzen die Schubladen durch Aufkleber. Anders als Menschen haben Computer wesentlich weniger Probleme damit, schnell alle Objekte mit demselben Aufkleber zu finden, die über mehrere Schubladen verteilt sind oder gar auf einem einzigen Haufen liegen. Auch diese Änderung kann Einsteigern und sehr kleinen Diskussionsrunden zu Gute kommen, da man für seine neue Mitteilung nicht zuerst eine mehr oder weniger passende Schublade finden muss, sondern auch nach dem Schreiben eine inhaltliche Klassifizierung vornehmen kann, indem man einfach alle zutreffenden Schlagwörter auswählt.

Verwaltet, also durch Moderatoren vorgegeben, werden die Tags oder Schlagwörter übrigens, um den Autoren die Klassifizierung ihrer Mitteilung zu vereinfachen, und die Klassifizierung in geordnete Bahnen zu lenken. Wie freie Tags aussehen, kann man sehr gut in einigen sozialen Netzwerken des „Web 2.0" sehen: mehrere ähnliche Wörter für dieselbe Bedeutung werden verwendet (teilweise falsch geschrieben), wichtige Einordnungen weggelassen oder irrelevante Angaben gemacht. Allein die fehlende Zusammenführung ähnlicher Wörter erschwert eine Suche schon sehr, da man für eine bestimmte Suche relevante Mitteilungen an den unterschiedlichsten Stellen findet. Wenn man überhaupt etwas findet.

Weil auch die Tags einer gewissen hierarchischen Struktur folgen können (vergleichbar mit Organisationseinheiten eines Unternehmens: Abteilungen, Gruppen, Projekte etc., oder Klassifizierungsschemata einer Bibliothek: Nachschlagewerke, Fremdsprachen, Wörterbücher etc.), kann man ihnen zusätzlich noch jeweils ein übergeordnetes Tag zuweisen, das eine Auswahl und Darstellung der Tags ähnlich einer Baumstruktur zulässt, wobei automatisch alle übergeordneten Tags mit selektiert oder das Tag selbst allein in gruppierender

Funktion von einer Auswahl ausgeschlossen werden können. Beim Antworten auf eine Mitteilung werden prinzipiell alle Tags übernommen, Änderungen sind jedoch möglich. So können für einzelne Mitteilungen Tags mit einer speziellen Bedeutung z.B. bzgl. der Zugriffsrechte gesetzt werden. Rückmeldungen über falsch zugewiesene Tags können mit der Moderationsfunktion „Mitteilung melden" erfolgen.

Wie im Kapitel über Mitteilungsrevisionen bereits angemerkt können Tags auch als Attribute verwendet werden. Dabei wird neben der bloßen Zuordnung eines Tags an eine Mitteilungsrevision auch ein zusätzlicher Wert angegeben. Solche Angaben, auch Metadaten genannt, können verwendet werden, um bestimmte Eigenschaften eines Inhalts näher zu beschreiben. Ein Beispiel dafür sind die bereits genannten EXIF-Daten eines Digitalfotos: sie enthalten u.a. Angaben zu Aufnahmeparametern der Kamera wie Belichtungszeit oder die Verwendung eines Blitz. Jedes Tag kann mit dem `UserData`-Flag versehen werden, das die Angabe eines benutzerdefinierten Wertes mit ihrer Zuordnung zu einer Mitteilungsrevision vorsieht. Oft wird man diesen Wert in einem Textfeld eingeben, es bleibt jedoch der Anwendung überlassen, für bestimmte Attribute spezialisierte Eingabefelder (wie z.B. einen Kalender für Datumsangaben) zu verwenden.

Neben diesem gibt es noch weitere Flags, die das Verhalten des Tags beeinflussen. Dazu gehören das `NotSelectable`-Flag, das für in einer Hierarchie übergeordnete Tags gedacht ist, die nur zur Strukturierung verwendet werden, aber selbst nicht zugewiesen werden dürfen, und das `Invisible`-Flag, das ein Tag (oder Attribut) nur für programminterne Zwecke nutzbar macht, es aber vor dem Benutzer verbirgt.

Außerdem haben Tags noch ein Feld, in dem eine spezielle Bedeutung abgelegt werden kann. Diese Bedeutung ist durch die Basisklassen nicht festgelegt und kann von der darauf aufbauenden Anwendung frei genutzt werden. Dadurch können Tags von der Anwendung anhand ihrer Bedeutung, unabhängig vom Namen gefunden werden.

2.2. Zugangssteuerung und Moderation

2.2.1. Zugangssteuerung

Die Zugangssteuerung erfolgt primär über **Schlüssel**. Wenn man eine bestimmte Aktion durchführen möchte, benötigt man dafür ggf. einen passenden Schlüssel. Das Prinzip ist dasselbe wie wenn man eine verschlossene Tür öffnen möchte. Schlüssel sind systemweit eindeutige 32-Bit-Zahlen. Sie sind insofern kein Geheimnis, man kann sie problemlos einsehen. Es ist alleine die Tatsache entscheidend, ob ein Benutzer laut der Datenbank im Besitz eines bestimmten Schlüssels ist oder nicht. Die Eingabe eines Schlüssels durch den Benutzer selbst hilft ihm nicht weiter. *Jede Benutzer-ID ist ein Schlüssel*, somit hat jeder Benutzer seinen persönlichen Schlüssel. Dass diese Schlüssel anstatt fortlaufend als fünfstellige (oder längere) Zufallszahl generiert werden, ist mehr ein optischer Faktor. Wichtig ist nur, dass die niederwertigen Zahlen für feste Schlüssel vorgesehen sind. Dazu gehört der Systemschlüssel, der Administratorschlüssel, der Moderatorschlüssel und einer für vertrauenswürdige Benutzer. Um eine Art Vorlage für neue Benutzer und deren Einstellungen adressieren zu können, gibt es noch einen Vorlagenschlüssel. Diese haben die Zahlenwerte 1 bis 5.

Gruppenzugehörigkeiten oder Gruppenrechte werden durch zusätzliche Schlüssel umgesetzt. Dazu wird ein „virtueller Benutzer" angelegt (denn ein Schlüssel ist nichts anderes als eine Benutzer-ID), dessen Schlüssel andere Benutzer in ihren **Schlüsselbund** aufnehmen. Diese virtuellen Benutzer können sich nicht selbst am System anmelden, da ihnen keine Anmeldedaten wie z.B. ein Anmeldename oder Kennwort zugewiesen werden. Zu diesen virtuellen Benutzern gehören auch die festen Schlüssel. Durch sie kann ein Benutzer Administrations- oder Moderationsaufgaben übernehmen. Der System- sowie der Vorlagenschlüssel sollten nicht einem Benutzer zugeordnet werden, da es keine sinnvolle Kombination darstellt. Durch die Zuordnung des Schlüssels für vertrauenswürdige Benutzer können bestimmte Aktionen erlaubt werden, die anderen Benutzern nicht gestattet sind, z.B. das Veröffentlichen von Mitteilungen ohne auf eine Moderation zu warten.

Eine Mitteilung enthält jeweils eine **Schlüsselliste** für Lese-, Änderungs- und Antwortberechtigungen. Wenn diese Schlüsselliste nicht leer ist, wird die entsprechende Aktion nur einem Benutzer gewährt, der über einen dieser Schlüssel verfügt. Dafür kann der persönliche Schlüssel des Benutzers sowie alle in seinem Schlüsselbund enthaltenen Schlüssel verwendet werden. Die Listen für Lese- und Antwortberechtigung sind norma-

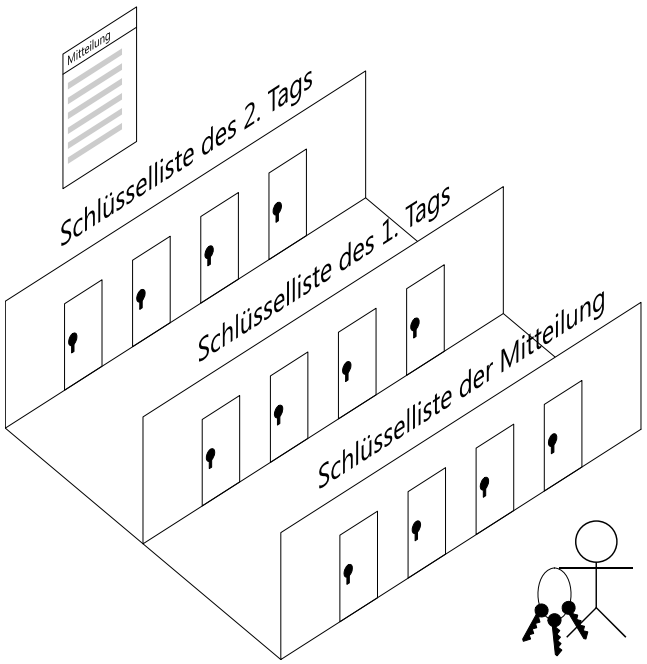
lerweise leer, wenn die Mitteilung frei zugänglich ist und das Antworten darauf nicht eingeschränkt ist. Wird allein der Moderatorenschlüssel auf die Liste zur Antwortberechtigung gesetzt, ist der folgende Themenabschnitt „geschlossen" und es können keine Antworten darauf mehr veröffentlicht werden (außer von Moderatoren). Auf die gleiche Weise kann sich z.B. auch der Autor einer Mitteilung das alleinige Antwortrecht sichern. Die Änderungs-Schlüsselliste enthält normalerweise nur den Schlüssel des Eigentümers (und einzigen Autors) und den festen Schlüssel der Moderatoren, sodass nur diese die Mitteilung bearbeiten dürfen. Sollen Mitteilungen wie in einem CMS oder Wiki üblich von mehreren Benutzern geändert werden können, werden weitere Schlüssel zu dieser Liste hinzugefügt bzw. die Liste geleert.

Ein **Tag** verfügt ebenfalls über zwei Schlüssellisten. Die eine legt die Nutzungsberechtigung des Tags fest. Ist diese Liste nicht leer, darf das Tag nur von Inhabern eines passenden Schlüssels an Mitteilungen zugewiesen werden. Beim Antworten auf eine Mitteilung werden i.d.R. alle in der Ursprungsmitteilung gesetzten Tags als Vorgabe für die Antwort übernommen. Tags, für die man keine Nutzungsberechtigung besitzt (z.B. so etwas wie „Offizielle Ankündigung"), sind davon ausgenommen. Diese Schlüsselliste neuer Tags ist normalerweise leer. Aus den Tag-Nutzungsberechtigungen ergeben sich weitere Einschränkungen auf Mitteilungen, die das Tag verwenden: Diese Mitteilungen dürfen nur von Benutzern verändert werden, die zusätzlich über die Nutzungsberechtigung des Tags verfügen. (So lassen sich wichtige Inhaltsseiten zusätzlich vor unerwünschten Veränderungen schützen.) Die zweite Schlüsselliste wird dafür verwendet, um die Leseberechtigung für Mitteilungen, die das Tag verwenden, einzuschränken. Auch hier gilt: Ist die Liste nicht leer, dürfen nur Benutzer mit passendem Schlüssel die entsprechenden Mitteilungen lesen. (Das entspricht im Wesentlichen den „Forenberechtigungen" anderer Systeme.)

Hier also nochmal eine Übersicht aller Schlüssellisten:

Entität	**Schlüsselliste**	**Code-Name**	**Verwendung**
Benutzer	Schlüsselbund	`AdditionalKeylist`	Gewährt den Zugang zu weiteren eingeschränkten Funktionen
Mitteilung	Leseberechtigung	`ReadAccessKeylist`	Beschränkt das Lesen der Mitteilung
	Änderungsberechtigung	`AlterAccessKeylist`	Beschränkt das Ändern der Mitteilung
	Antwortberechtigung	`ReplyAccessKeylist`	Beschränkt das Antworten auf die Mitteilung
Tag	Leseberechtigung	`ReadAccessKeylist`	Beschränkt das Lesen aller Mitteilungen, die dieses Tag verwenden
	Nutzungsberechtigung	`UseAccessKeylist`	Beschränkt das Zuweisen des Tags an Mitteilungen sowie das Ändern aller Mitteilungen, die dieses Tag verwenden

Die nachfolgende Abbildung veranschaulicht das Zusammenspiel der einzelnen Schlüssel. Der Benutzer, der Zugang zu einer Mitteilung erlangen möchte, besitzt einen Schlüsselbund mit ihm zugewiesenen Schlüsseln, darunter auch sein persönlicher Schlüssel. Die Mitteilung ist bildlich gesprochen durch eine Wand gesichert, durch die mehrere Türen führen. Jede dieser Türen lässt sich mit einem Schlüssel der Schlüsselliste der Mitteilung öffnen, in diesem Fall der Liste für die Leseberechtigung. Es genügt, eine dieser Türen zu öffnen, um Zugang zu erlangen. Hinter dieser Wand wird ggf. für jedes den Zugang einschränkende Tag eine weitere Wand mit mehreren Türen gezogen. Jede dieser Türen lässt sich mit einem Schlüssel der Schlüsselliste des Tags öffnen. Der Benutzer muss also zusätzlich zur Einschränkung durch die Mitteilung selbst auch die Einschränkungen aller Tags passieren, wobei in jeder Wand nur eine Tür geöffnet werden muss. Mitteilungen oder Tags, die keine eigene Schlüsselliste besitzen, verursachen keine Wand an der entsprechenden Stelle.



Schlüsselbund des Benutzers

Es ist die Aufgabe von **Administratoren**, Zugangsberechtigungen zu verwalten. Zunächst müssen sich auch Benutzer mit Administrationsrechten an die geltenden Zugriffsbeschränkungen halten. Sie haben aber die Möglichkeit, Schlüssellisten von beliebigen Objekten zu verändern. Auf diese Weise ist es ihnen möglich, Mitteilungen nur für bestimmte Benutzer zugänglich zu machen oder selbst Zugang zu eingeschränkten Mitteilungen zu erhalten. Üblicherweise darf nur der Eigentümer einer Mitteilung (also i.d.R. der ursprüngliche Autor) den Zugang zu seiner Mitteilung festlegen, was u.a. für private oder Gruppennachrichten verwendet werden kann. Administratoren können außerdem Benutzerkonten bearbeiten, also z.B. das Anmeldekennwort oder den Schlüsselbund ändern.

Die Aufgabe von **Moderatoren** ist die Organisation der Mitteilungen. Dazu gehört das Verwalten von Tags, also das Anlegen und Ändern von Tags, sowie das Zuweisen der Tags an Mitteilungen. (Die Bearbeitung der Schlüssellisten von Tags ist wiederum nur Administratoren gestattet.) Auch das Bearbeiten von Referenzen unter einzelnen Mitteilungen kann von Moderatoren durchgeführt werden, z.B. um die Referenz einer Mitteilung zu entfernen, die fälschlicherweise als Antwort veröffentlicht wurde, oder Mitteilungen als Diskussionseinstiegspunkt zu markieren, was in etwa mit dem Aufteilen eines Threads in anderen Systemen vergleichbar ist. Sofern erforderlich können Moderatoren auch Mitteilungen bearbeiten (ausreichende Änderungsberechtigung auf die Mitteilung vorausgesetzt, siehe Abschnitt 2.2.1, „Schlüsselliste“), um z.B. unerwünschte Inhalte direkt zu entfernen. Weitere Hinweise dazu werden im folgenden Abschnitt beschrieben.

2.2.2. Moderation

Insbesondere die sich in Deutschland abzeichnende Rechtslage, aber auch die Verbreitung von Spam (unerwünschten Werbebotschaften) oder andere Interessen erfordern strenge Prüfverfahren für neue Mitteilungen. Eine neue Mitteilung, oder besser, eine neue Mitteilungsrevision muss genehmigt werden, bevor sie veröffentlicht wird. Davor können nur der Autor und Moderatoren die Mitteilung lesen. Alle anderen Benutzer, angemeldet oder nicht, bekommen an ihrer Stelle nur einen Hinweis auf die noch nicht genehmigte Mitteilung zu sehen. Es ist die Aufgabe von Moderatoren, Mitteilungen zu genehmigen oder ggf. zu sperren.

Die Mitteilungen von vertrauenswürdigen Benutzern werden je nach Konfiguration automatisch genehmigt. Eine Ausnahme davon sind allerdings Antworten auf Mitteilungen, die das `EnforceApproval`-Flag gesetzt haben. Durch dieses Flag müssen alle Antworten auf die jeweilige Mitteilung explizit genehmigt werden. Beim Antworten auf eine Mitteilung wird dieses Flag rekursiv in allen vorherigen, über die primäre Referenz verknüpften Mitteilungen gesucht, jedoch nicht über einen Diskussionseinstiegspunkt hinweg. Mit diesem Flag können besondere moderierte Diskussionen eingerichtet werden, in denen auch Mitteilungen von Be-

nutzern geprüft werden, die das sonst nicht erfordern. Das `EnforceApproval`-Flag kann nur von Moderatoren gesetzt oder entfernt werden, nicht vom Autor einer Mitteilung selbst (außer er *ist* ein Moderator), da derjenige, der das Flag setzt, auch das Interesse hat, die Antworten zu prüfen, und Nicht-Moderatoren ja gar keine Mitteilungen genehmigen können.

Zusätzlich zum Genehmigungsverfahren und ggf. dem Sperren einzelner Mitteilungsrevisionen gibt es die Möglichkeit, komplette Mitteilungen zu sperren oder ganz zu verstecken. Beide Zustände werden, wie das `EnforceApproval`-Flag, als Flag einer Mitteilung gespeichert und sind nur von Moderatoren veränderbar. Das Sperren einer Mitteilung hat hat keinen Einfluss auf nachfolgende Antworten, es kann in der Anwendungsoberfläche aber die Möglichkeit bestehen, automatisch alle Antworten (bis vor den nächsten Diskussionseinstiegspunkt) mit zu sperren. Das Sperren einer Mitteilung kann dem Zweck dienen, unerwünschte veröffentlichte Inhalte vorerst möglichst schnell unzugänglich zu machen, ohne sie gleich zu löschen. So ist die Mitteilung zur späteren Auswertung oder Nachverfolgung weiterhin für Moderatoren verfügbar, bspw. um Ansprüche Dritter zu klären. Die Sperrung einer Mitteilung wirkt sich auf alle Revisionen aus, die ansonsten einzeln gesperrt werden können, aber auch auf zukünftige neue Revisionen. Benutzer, die keine Moderatoren und nicht der Eigentümer der Mitteilung sind, sehen anstelle der Mitteilung nur einen Hinweis auf die Sperrung.

Das Verstecken einer Mitteilung blendet sie vollständig aus, es wird also auch kein Hinweis auf den Zustand der versteckten Mitteilung angezeigt. Da die Mitteilung nicht angezeigt wird, sind auch direkte Antworten darauf nicht mehr einfach zugänglich. Die Antworten können zwar weiterhin über ggf. vorhandene Direkt-Links abgerufen werden, sind aber nicht mehr aus der Antwortenliste der ursprünglichen Mitteilung aufrufbar. Unabhängig davon können natürlich auch alle Antworten mit versteckt werden, wodurch sie auch über Direkt-Links nicht mehr zugänglich wären.

2.2.3. Zusammenfassung

Zusammenfassend lässt sich anhand folgender Übersicht ablesen, aus welchen Gründen eine bestimmte Mitteilung oder Mitteilungsrevision für einen Benutzer nicht verfügbar ist.

Einschränkungen durch **Zugriffsrechte**:

- Zugriff auf die Mitteilung ist durch die Schlüsselliste der Mitteilung beschränkt
- Zugriff auf die Mitteilung ist durch die Schlüsselliste der der „Suchrevision" zugewiesenen Tags beschränkt

Diese Sperren können von keinem Benutzer umgangen werden. Administratoren und der Eigentümer einer Mitteilung haben aber generell die Möglichkeit, Schlüssellisten zu ändern. (Windows-Administratoren wird das vom NTFS bekannt vorkommen.) Dadurch erhalten Sie die Möglichkeit, die Zugriffsrechte so zu ändern, dass sie oder Dritte Zugang zu den gesperrten Inhalten erlangen, sofern keine Moderationsmaßnahmen den Zugang weiterhin verhindern.

Einschränkungen durch **Moderationsmaßnahmen**:

- Mitteilung ist gesperrt oder versteckt
- Mitteilungsrevision wartet auf Genehmigung
- Mitteilungsrevision ist gesperrt

Diese Sperren werden durch einen Moderator automatisch umgangen. Ein Moderator kann also ohne weiteres Eingreifen ungenehmigte oder gesperrte Inhalte sehen und lesen.

2.2.4. Ausnahmen der Zugriffssperren

Der Zugriff auf Mitteilungen, soweit nicht durch die oben genannten Maßnahmen eingeschränkt, wird generell gewährt. Zusätzlich dazu gibt es eine Reihe von genau definierten Ausnahmen, die auf bestimmte Situationen zutreffen. Dafür muss unterschieden werden, welcher Zugriff auf eine Mitteilung beabsichtigt wird. Es ist möglich, die Mitteilung ohne Inhalt abzurufen, wonach immer noch der Eigentümer bzw. Autor, der Zeitstempel und die zugewiesenen Tags verfügbar sind. Darüber hinaus kann man den Betreff und die Revisionszusammenfassung einer Mitteilungsrevision anfordern, was nur in einzelnen Situationen den zusätzlichen

Nutzen hat, dass Mitteilungen anhand des Betreffs besser unterschieden werden können. Letztlich kann man den vollen Inhalt und angehängte Dateidaten anfordern, was zur Darstellung der Mitteilung nötig ist.

Im Fall, dass der Zugang zu einer Mitteilung aufgrund von **Schlüssellisten** verwehrt würde, gilt folgende Zugangsmatrix für den Zugriff auf verschiedene Objekte durch die einzelnen Benutzertypen (Subjekte):

Subj. \ Obj.	Mitteilung	Rev. ohne Inhalt	Rev. mit Betreff	Rev. mit Inhalt
Benutzer	–	–	–	–
Moderator	–	–	–	–
Administrator	✓[1]	✓	✓[2]	–
Eigentümer	✓[1]	✓	✓[2]	–

[1]) Administratoren und der Eigentümer der Mitteilung erhalten die Möglichkeit, den Zugang zur Mitteilung zu ändern.

[2]) Der Betreff dient dem leichteren Umgang mit unzugänglichen Mitteilungen. Da der Nutzerkreis sehr eingeschränkt ist, kann dies nicht zur unautorisierten Veröffentlichung ungenehmigter Informationen genutzt werden.

Im Fall, dass der Zugang zu einer Mitteilung aufgrund von **Moderations-Flags der Mitteilung** verwehrt würde, gilt folgende Zugangsmatrix:

Subj. \ Obj.	Mitteilung allein	Rev. ohne Inhalt	Rev. mit Betreff	Rev. mit Inhalt
Benutzer	✓[3]	✓	–	–
Moderator	✓[4]	✓[4]	✓[4]	✓[4]
Administrator	✓	✓	✓	–
Eigentümer	✓[5]	*(wie Benutzer)*	*(wie Benutzer)*	*(wie Benutzer)*
Autor	*(n.v.)*	✓[5]	✓[5]	✓[5]

[3]) Falls die Mitteilung versteckt wurde, und nicht nur gesperrt, ist durch einen Benutzer kein Zugriff möglich.

[4]) Moderatoren umgehen Moderationssperren automatisch. Es sollte ihnen allerdings ein entsprechender Hinweis angezeigt werden, der über diesen Umstand informiert.

[5]) Dem Eigentümer einer Mitteilung und dem Autor einer Revision wird der Zugang nicht durch Moderationsmaßnahmen verwehrt. Wenn Autor und Eigentümer voneinander abweichen, kann der Zugang u.U. dennoch verwehrt werden.

Im Fall, dass der Zugang zu einer Mitteilungsrevision aufgrund des **Moderationsstatus der Revision** verwehrt würde, gilt folgende Zugangsmatrix:

Subj. \ Obj.	Rev. ohne Inhalt	Rev. mit Betreff	Rev. mit Inhalt
Benutzer	✓	–	–
Moderator	✓[4]	✓[4]	✓[4]
Administrator	✓	✓	–
Autor	✓[5]	✓[5]	✓[5]

Insgesamt betrachtet lässt sich jede Revision ohne Inhalt abrufen, solange der Benutzer über einen ggf. benötigten Schlüssel verfügt und die Mitteilung nicht durch einen Moderator versteckt wurde. Der Betreff und der eigentliche Inhalt der Mitteilung sind dagegen vollständig von den Moderationsmaßnahmen betroffen.

2.3. Web-Interface

Eines der Hauptziele der UNB Components ist es, möglichst einfach in bestehende Webseiten integrierbar zu sein. Dazu gehört es, dass einzelne UNB-Elemente für sich möglichst viel technisch bedingte Programmlogik übernehmen und ihre Verwendung nicht übermäßig viel und komplizierten Programmcode in der Anwendung erfordert. Ein wesentlicher Bestandteil der UNB Components ist daher die automatische Formularverarbeitung. In vielen Fällen reicht es aus, ein einfaches Formular in der HTML-Seite anzulegen, das Eingabefelder mit bestimmten Feldnamen enthält. Nach dem Absenden des Formulars wird durch wenige Codezeilen die

automatische Formularverarbeitung aufgerufen, die dann entsprechend den im Formular übertragenen Feldnamen die gewünschten Aktionen durchführt. Auf diese Weise können Anwendungsformulare sehr flexibel gestaltet werden, z.B. indem einzelne Abschnitte nur bei Bedarf ins Formular aufgenommen werden, ohne dass die Anwendung selbst die entsprechenden Aufrufe des UNB-API implementieren muss.

2.3.1. Formularverarbeitung

Das zentrale Steuerfeld, das der Formularverarbeitung mitteilt, welche Aktionen durchzuführen sind, heißt `action[]`. Diesem Array-Feld können mehrere Aktionsbezeichner zugewiesen werden. So kann in einem Formular z.B. eine neue Mitteilung erstellt und in einem Schritt auch mit Tags und Zugriffslisten verknüpft werden. Zur Ausführung werden die Aktionen nach einer bestimmten Reihenfolge sortiert, so dass ggf. neu zu erstellende Objekte (z.B. Benutzer anlegen) zuerst erstellt werden, bevor sie verändert werden (z.B. Schlüssel hinzufügen). Wenn bei der Durchführung einer Aktion ein Fehler auftritt, werden bisherige Änderungen an der Datenbank rückgängig gemacht und keine weiteren Aktionen durchgeführt.

Die einzelnen Datenfelder haben festgelegte Namen. Die Feldnamen werden für mehrere Aktionen wiederverwendet, das Feld für ein Benutzerkennwort hat z.B. immer den gleichen Namen, ob nun bei der Anmeldung, beim Erstellen eines neuen Benutzers oder beim Ändern des Kennworts. So kann das gleiche Eingabeelement für verschiedene Zwecke verwendet werden.

Die Methode zur Formularverarbeitung liefert ein mehrdimensionales Array zurück, das Angaben über durchgeführte Aktionen und ggf. dabei aufgetretene Fehler enthält. Die Fehlermeldungen können dem Anwender angezeigt werden. Bei Erfolg aller Aktionen sollte normalerweise eine Weiterleitung zu einer neuen Seite erfolgen, um das versehentliche erneute Absenden des letzten Formulars zu vermeiden.

Das Web-Interface kann auch von Anwendungen außerhalb einer Browser-Umgebung verwendet werden, um mit dem UNB-System zu interagieren. Die Anfragen werden dafür per HTTP POST gesendet. Um Antworten auszuwerten, muss ein zusätzlicher XML-Server eingerichtet werden, der die Rückgabe der Formularverarbeitung ins XML-Format übersetzt. Dieser XML-Server besteht nur aus wenigen Zeilen Programmcode, da das UNB-API bereits fast alle Arbeiten übernimmt. Zusätzlich zu den mit der beschriebenen Formularverarbeitung möglichen Änderungsaktionen können über diese XML-Schnittstelle auch Daten abgerufen werden, um sie z.B. mit JavaScript oder in einer anderen Anwendung darzustellen.

Die definierten Aktionen mit den von ihnen ausgewerteten Datenfeldern werden in der Referenz aufgelistet.

2.3.2. Benutzerschnittstelle (UI)

Um die Verwendung der UNB-Funktionen in Webseiten noch einfacher zu gestalten, werden Methoden angeboten, um die HTML-Eingabeelemente zu erzeugen, die von der Formularverarbeitung verwendet werden. Dazu gehören zahlreiche einzeilige und mehrzeilige Textfelder, Auswahllisten und Optionsfelder. Die Eingabefelder werden je nach Verwendung bereits mit den Daten des zu bearbeitenden Objekts vorbelegt und behalten ihren Wert auch nach dem Absenden, falls das Formular aufgrund eines Verarbeitungsfehlers erneut angezeigt werden muss. Manche Felder enthalten zusätzliche Funktionalität, um dem Anwender ihre Verwendung zu erklären oder zu vereinfachen.

Das Einbinden eines Eingabefelds in einer Webseite beschränkt sich in den meisten Fällen auf das Einfügen eines PHP-Methodenaufrufs mit wenigen Parametern, z.B. dem Objekt, das in einem Formular bearbeitet werden soll.

Alle Methoden zur Anzeige von Eingabefeldern sind in einer eigenen PHP-Klasse `UnbUi` zusammengefasst und können nach der UNB-Initialisierung nahezu ohne vorbereitende Aufrufe verwendet werden. Die einzelnen Methoden sind in der Code-Referenz aufgelistet. Darunter sind neben den reinen Eingabeelementen auch Methoden zur allgemeinen Anzeige von Fehler- oder Erfolgsmeldungen, zur Seitenweiterleitung oder zur allgemeinen Formularsteuerung.

2.4. Klassenübersicht

Der Programmcode der UNB Components ist vollständig objektorientiert aufgebaut. Es gibt keine globalen Funktionen und alle Klassennamen beginnen mit einem einheitlichen Präfix. So sind Kollisionen mit Funktionen oder Klassen anderer Anwendungen praktisch ausgeschlossen. Deshalb können die UNB-Klassen problemlos in bestehende Anwendungen eingebunden werden. Alle Session-Funktionen werden überwiegend automatisch durchgeführt, was das Einbinden und Initialisieren der UNB-Klassen auf wenige Codezeilen beschränkt.

Manche der Klassen enthalten Konstanten, mit denen sich ihr Verhalten beeinflussen lässt. Dabei handelt es sich nur um Einstellungen, die ohnehin einen tiefgehenden Eingriff in die Anwendung oder eine Veränderung von Dateien oder Verzeichnissen erfordern.

Die UNB-Klassen lassen sich in folgende Kategorien aufteilen.

2.4.1. Datenbankverbindung und Konfiguration

`UnbDatabase`

Die Datenbankverbindungsklasse stellt eine einfache Abstraktion von Abfragen bereit, mit deren Methoden auf einfache Weise SQL-Anweisungen ausgeführt und Datensätze oder einzelne Werte mit einer SQL-Abfrage gelesen werden können.

Mit den Konstanten `DisplayQueryErrors` und `DisplayQueryInfo` lässt sich das Verhalten bzgl. der Anzeige aller ausgeführten SQL-Anweisungen oder nur solcher, bei der Fehler aufgetreten sind, einstellen. Diese Einstellungen sind für die Entwicklungsumgebung und zur Fehlersuche gedacht und sollten im Produktivbetrieb wirklich deaktiviert sein.

`UnbConfiguration`

Die Konfigurationsklasse enthält statische Methoden zum Lesen und Ändern von UNB-Einstellungen. Die Einstellungen sind in einer Datei vorgegeben und können durch veränderbare, in der Datenbank abgelegte Einstellungen erweitert werden. Zu den nicht in der Datenbank abgelegten Angaben gehört auch die Konfiguration der Datenbankverbindung. Eine Auflistung aller Konfigurationseinstellungen befindet sich in Kapitel 5.2.

2.4.2. Entity-Klassen

`UnbMessage`

Die Mitteilungsklasse repräsentiert eine einzelne Mitteilung im UNB-System und stellt Methoden zum Auffinden und Verändern einzelner Mitteilungen bereit.

`UnbMessageRevision`

Die Mitteilungsrevisionsklasse repräsentiert eine einzelne Mitteilungsrevision im UNB-System und stellt Methoden zum Auffinden und Verändern einzelner Mitteilungsrevisionen bereit.

`UnbTag`

Die Tagklasse repräsentiert ein einzelnes Tag im UNB-System und stellt Methoden zum Auffinden und Verändern einzelner Tags bereit.

`UnbUser`

Die Benutzerklasse repräsentiert einen einzelnen Benutzer im UNB-System und stellt Methoden zum Auffinden und Verändern einzelner Benutzer bereit.

Die Konstante `KeyLength` gibt an, wie viele Stellen neu generierte Benutzer-IDs lang sein sollen. Vorgabe ist `5`, bei sehr vielen vergebenen Schlüsseln kann eine Erhöhung auf bis zu `9` notwendig werden. Alle neuen Benutzer-IDs sind nummerisch, werden zufällig generiert und haben genau die hier eingestellte Anzahl Dezimalziffern.

Mit der Konstante `RequireSecurePassword` lässt sich festlegen, ob Benutzerkennwörter auf ihre Sicherheit hin überprüft werden sollen. Die genauen Kennwortrichtlinien sind im Quelltext einsehbar.

2.4.3. Funktionsklassen

`UnbEnvironment`

Diese Klasse enthält statische Methoden, um Kompatibilität mit unterschiedlichen Server-Umgebungen herzustellen und Angaben zum Client (Browser) abzufragen.

`UnbMarkup`

Diese Klasse enthält den Markup-Übersetzer, der die eingegebenen Mitteilungen aus ihrem Quelltext in eine HTML-Ausgabe zur Darstellung im Browser konvertiert.

`UnbSearch`

Diese Klasse stellt Suchmethoden bereit und verarbeitet Suchabfragen und ihre Ergebnisse.

`UnbSession`

Diese Klasse verwaltet eine Benutzersitzung und stellt somit eine Verbindung zwischen mehreren Webseitenabrufen her. Sie kann Benutzersitzungen starten, einrichten und beenden sowie Anmeldedaten erzeugen und überprüfen.

Mit den Konstanten `SessionName` und `SessionArrayVar` lässt sich die UNB-Session an die anderer Anwendungen anpassen. `SessionIpNetMask` und `SessionExpireTimeout` erlauben eine Änderung von sicherheitsrelevanten Funktionen einer Benutzersitzung. Mit `CookiePath` lässt sich die Nutzung von HTTP-Cookies auf bestimmte URLs beschränken.

`DefaultEncryptionKey` enthält einen Schlüssel, der zum Verschlüsseln von Daten verwendet wird, wenn durch die Konfiguration kein anderer Schlüssel angegeben wird. Diese Vorgabe ist nur als Notlösung anzusehen und sollte nicht verwendet werden.

2.4.4. Web-Interface

`UnbFormHandler`

Diese Klasse enthält die automatische Formularbehandlung (siehe Kapitel 2.3.1).

`UnbUi`

Diese Klasse stellt enthält statische Methoden, die einzelne HTML-Eingabeelemente anzeigen (siehe Kapitel 2.3.2).

2.4.5. Ausnahmen (Exceptions)

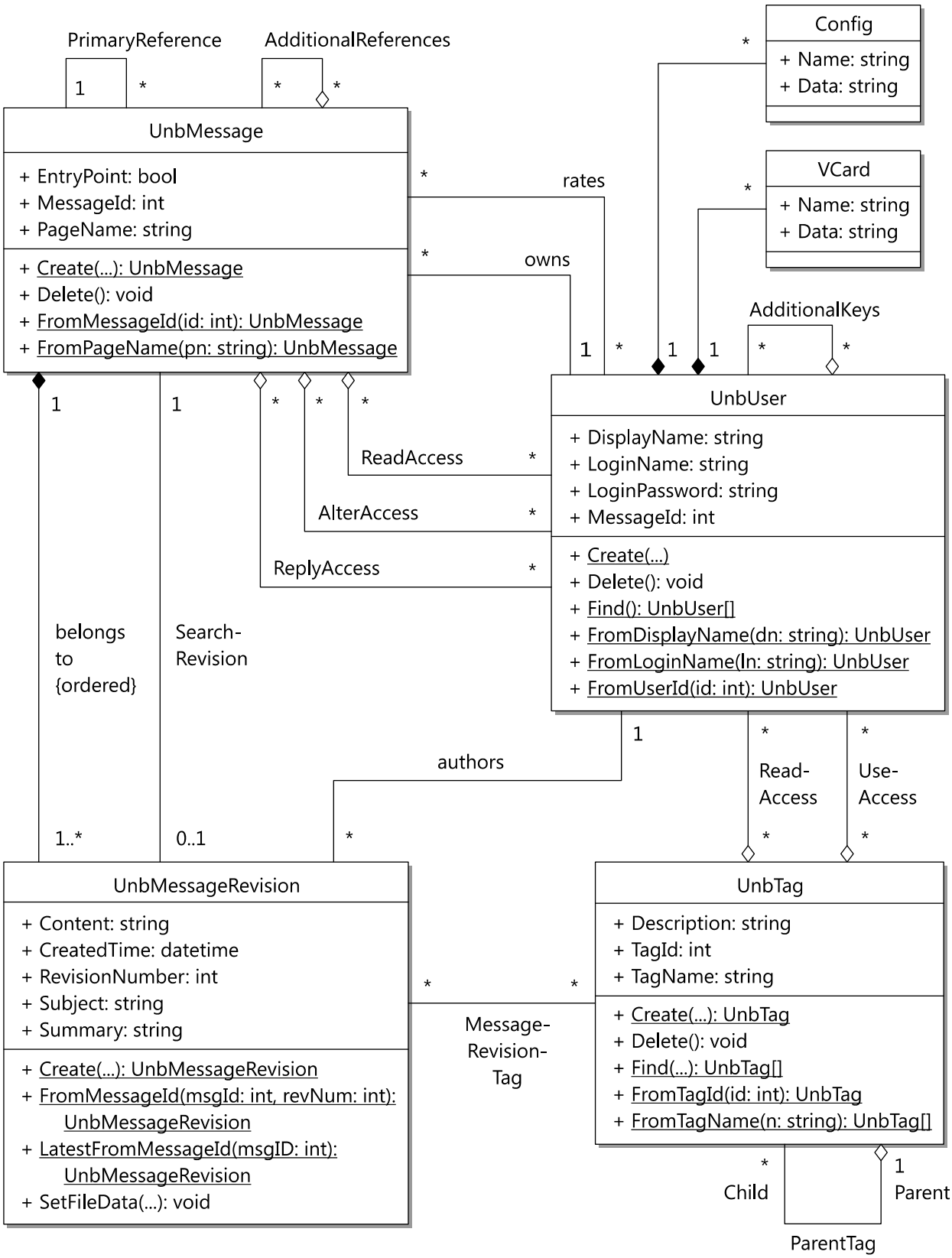
`UnbDatabaseException`

Die Datenbank-Ausnahme wird bei Fehlern in der Datenbankschicht wie ungültigen oder fehlgeschlagenen SQL-Abfragen oder Problemen mit der Datenbankverbindung verwendet.

`UnbSecurityException`

Die Sicherheits-Ausnahme wird bei sicherheitsrelevanten Fehlern verwendet, z.B. wenn eine Aktion ausgeführt werden soll, der Benutzer aber nicht angemeldet oder nicht dazu berechtigt ist.

Folgendes Klassendiagramm veranschaulicht die Beziehungen zwischen den Entity-Klassen. Die einzelnen Eigenschaften und Methoden sind zur besseren Darstellung teilweise vereinfacht und entsprechen nicht der genauen Implementierung.



2.5. Datenbank

Die folgenden Tabellen enthalten eine Beschreibung aller Felder der einzelnen Datenbanktabellen. Kursiv gedruckte Feldnamen können `NULL`-Werte enthalten.

2.5.1. Tabelle keylist

Diese Tabelle enthält Schlüssellisten für verschiedene Zwecke. In dieser Tabelle sind mehrere Beziehungen aus dem obigen E/R-Diagramm abgebildet, für die man normalerweise mehrere separate Tabellen verwenden würde. Da diese Tabellen aber alle gleich aussehen würden, sind sie hier in einer zusammengefasst.

Feldname	Datentyp	Beschreibung
`KeylistId`	`INTEGER UNSIGNED`	Alle Einträge in einer Schlüsselliste werden mit der Schlüssellisten-ID referenziert
`UserId`	`INTEGER UNSIGNED`	Ein Schlüsseleintrag in der Liste

2.5.2. Tabelle message

Diese Tabelle enthält alle Mitteilungen.

Feldname	Datentyp	Beschreibung
`MessageId`	`INTEGER UNSIGNED`	Mitteilungs-ID
`Owner`	`INTEGER UNSIGNED`	Benutzer, dem die Mitteilung zugewiesen ist, üblw. der Benutzer, der die Mitteilung erstellt hat
`PrimaryReference`	`INTEGER UNSIGNED`	Mitteilungs-ID, auf die sich die Mitteilung hauptsächlich bezieht, verwendet für Baumansicht oder Antworten
`EntryPoint`	`BOOLEAN DEFAULT FALSE`	Gibt an, ob die Mitteilung ein Diskussionseinstiegspunkt ist
`PageName`	`VARCHAR(255)`	Seitenname der Mitteilung für direkten Zugriff
`ReadAccessKeylistId`	`INTEGER UNSIGNED`	ID der Schlüsselliste, die alle Benutzer enthält, die die Mitteilung lesen dürfen (falls nicht `NULL`)
`AlterAccessKeylistId`	`INTEGER UNSIGNED`	ID der Schlüsselliste, die alle Benutzer enthält, die die Mitteilung verändern dürfen (falls nicht `NULL`)
`ReplyAccessKeylistId`	`INTEGER UNSIGNED`	ID der Schlüsselliste, die alle Benutzer enthält, die auf die Mitteilung antworten dürfen (falls nicht `NULL`)
`ModerationFlags`	`TINYINT UNSIGNED`	Moderations-Flags der ganzen Mitteilung. `1` = Gesperrt `2` = Versteckt `4` = Genehmigung der Antworten erforderlich (siehe `UnbMessageModerationFlag::*`)
`SearchOptimisedContent`	`MEDIUMTEXT`	Inhalt der Mitteilung, der für die Suche berücksichtigt wird, üblw. vereinfachte Nur-Text-Version der letzten genehmigten Mitteilungsrevision
`SearchOptimisedSubject`	`VARCHAR(255)`	Betreff der Mitteilung, der für die Suche berücksichtigt wird, üblw. aus der letzten genehmigten Mitteilungsrevision
`SearchRevision`	`INTEGER UNSIGNED`	Revisionsnummer, die für Tag-Suchen verwendet wird, üblw. die letzte genehmigte Mitteilungsrevision
`LastBeginEdit`	`DATETIME`	Zeit, zu der ein Benutzer zuletzt begonnen hat, die Mitteilung zu bearbeiten, verwendet für eine Warnmeldung an Benutzer, die die Mitteilung gleichzeitig bearbeiten wollen

2.5.3. Tabelle message_rating

Diese Tabelle enthält alle Mitteilungsbewertungen der Benutzer.

Feldname	Datentyp	Beschreibung
UserId	INTEGER UNSIGNED	ID des Benutzers, der die Mitteilung bewertet hat
MessageId	INTEGER UNSIGNED	ID der Mitteilung, die bewertet wurde
Rating	TINYINT	Bewertung, 0 ist ein neutraler Wert, positive Werte sind gut, negative sind schlecht
Weight	TINYINT UNSIGNED	Bewertungsgewicht, das angibt, wie bedeutend eine einzelne Bewertung für die Gesamtbewertung ist

2.5.4. Tabelle message_reference

Diese Tabelle enthält alle zusätzlichen Referenzen der Mitteilungen untereinander.

Feldname	Datentyp	Beschreibung
MessageId	INTEGER UNSIGNED	ID der Mitteilung, die eine andere Mitteilung referenziert
Reference	INTEGER UNSIGNED	ID der Mitteilung, die referenziert wird

2.5.5. Tabelle message_revision

Diese Tabelle enthält die Mitteilungsrevisionen, die die eigentlichen Inhalte der Mitteilungen darstellen.

Feldname	Datentyp	Beschreibung
MessageId	INTEGER UNSIGNED	ID der Mitteilung, zu der diese Revision gehört
RevisionNumber	SMALLINT UNSIGNED	Revisionsnummer, beginnend mit 1, mit jeder neuen Revision um 1 erhöht – Es wird nicht garantiert, dass der Wert 1 existiert und die Zahlen fortlaufend sind!
CreatedTime	DATETIME	Zeit, zu der die Revision erstellt wurde
Author	INTEGER UNSIGNED	ID des Benutzers, der die Revision erstellt hat
Subject	VARCHAR(255)	Betreff
Content	MEDIUMTEXT	Textinhalt
HtmlContent	MEDIUMTEXT	Zwischengespeicherte HTML-Ausgabe des Textinhalts, verwendet zur beschleunigten Anzeige der Mitteilungsrevision
Summary	VARCHAR(255)	Zusammenfassung der Änderungen in dieser Revision gegenüber der vorherigen
ModerationState	TINYINT UNSIGNED DEFAULT 0	Moderationsstatus dieser Revision 0 = Wartet auf Genehmigung 1 = Genehmigt 2 = Gesperrt (siehe UnbMessageRevisionModerationState::*)
Data	MEDIUMBLOB	Inhalt der angehängten Datei, wenn sie nicht in einer separaten Datei im Dateisystem gespeichert wurde
HasData	BOOLEAN	Gibt an, ob an die Revision eine Datei angehängt ist

2.5.6. Tabelle message_revision_tag

Diese Tabelle enthält alle Zuordnungen von Tags an Mitteilungsrevisionen und ggf. deren Attributwerte.

Feldname	Datentyp	Beschreibung
MessageId	INTEGER UNSIGNED	ID der Mitteilung, der das Tag zugewiesen ist
RevisionNumber	SMALLINT UNSIGNED	Revisionsnummer, der das Tag zugewiesen ist
TagId	INTEGER UNSIGNED	ID des zugewiesenen Tags
TagData	VARCHAR(255)	Attributwert für die Revision, falls tag.Flags UserData enthält

2.5.7. Tabelle tag

Diese Tabelle enthält alle Tags und deren Eigenschaften.

Feldname	Datentyp	Beschreibung
TagId	INTEGER UNSIGNED	Tag-ID
TagName	VARCHAR(255)	Angezeigter Name des Tags
ParentTag	INTEGER UNSIGNED	ID des übergeordneten Tags, verwendet für Baumansicht der Tags
Meaning	TINYINT UNSIGNED	Besondere Bedeutung des Tags
Description	VARCHAR(255)	Beschreibung des Tags, die dem Benutzer erklärt, wann und warum es verwendet werden soll
UseAccessKeylistId	INTEGER UNSIGNED	ID der Schlüsselliste, die alle Benutzer enthält, die das Tag verwenden dürfen (falls nicht NULL)
ReadAccessKeylistId	INTEGER UNSIGNED	ID der Schlüsselliste, die alle Benutzer enthält, die Mitteilungen mit diesem Tag lesen dürfen (falls nicht NULL)
Flags	TINYINT UNSIGNED	Flags für ein spezielles Verhalten des Tags 1 = Nicht auswählbar 2 = Unsichtbar 4 = Genehmigung der Antworten erforderlich 8 = Enthält Attributwerte (siehe UnbTagFlag::*)

2.5.8. Tabelle user

Diese Tabelle enthält alle Benutzer-IDs, also alle Schlüssel, und ggf. Anmeldedaten.

Feldname	Datentyp	Beschreibung
UserId	INTEGER UNSIGNED	Benutzer-ID, auch Schlüssel genannt
CreatedTime	DATETIME	Zeit, zu der der Benutzer erstellt wurde
LastLoginTime	DATETIME	Zeit, zu der sich der Benutzer zuletzt angemeldet hat
DisplayName	VARCHAR(255)	Angezeigter Name des Benutzers
AdditionalKeylistId	INTEGER UNSIGNED	ID der Schlüsselliste, die alle Schlüssel enthält, die dem Benutzer zugewiesen sind
LoginName	VARCHAR(255)	Anmeldename
LoginPassword	VARCHAR(255)	Anmeldekennwort, üblw. als sicherer Hash gespeichert

2.5.9. Tabelle user_config

Diese Tabelle enthält alle Konfigurationseinstellungen der Benutzer.

Feldname	Datentyp	Beschreibung
UserId	INTEGER UNSIGNED	ID des Benutzers, dem der Konfigurationseintrag gehört
Name	VARCHAR(255)	Name der Einstellung
Data	VARCHAR(255)	Wert der Einstellung

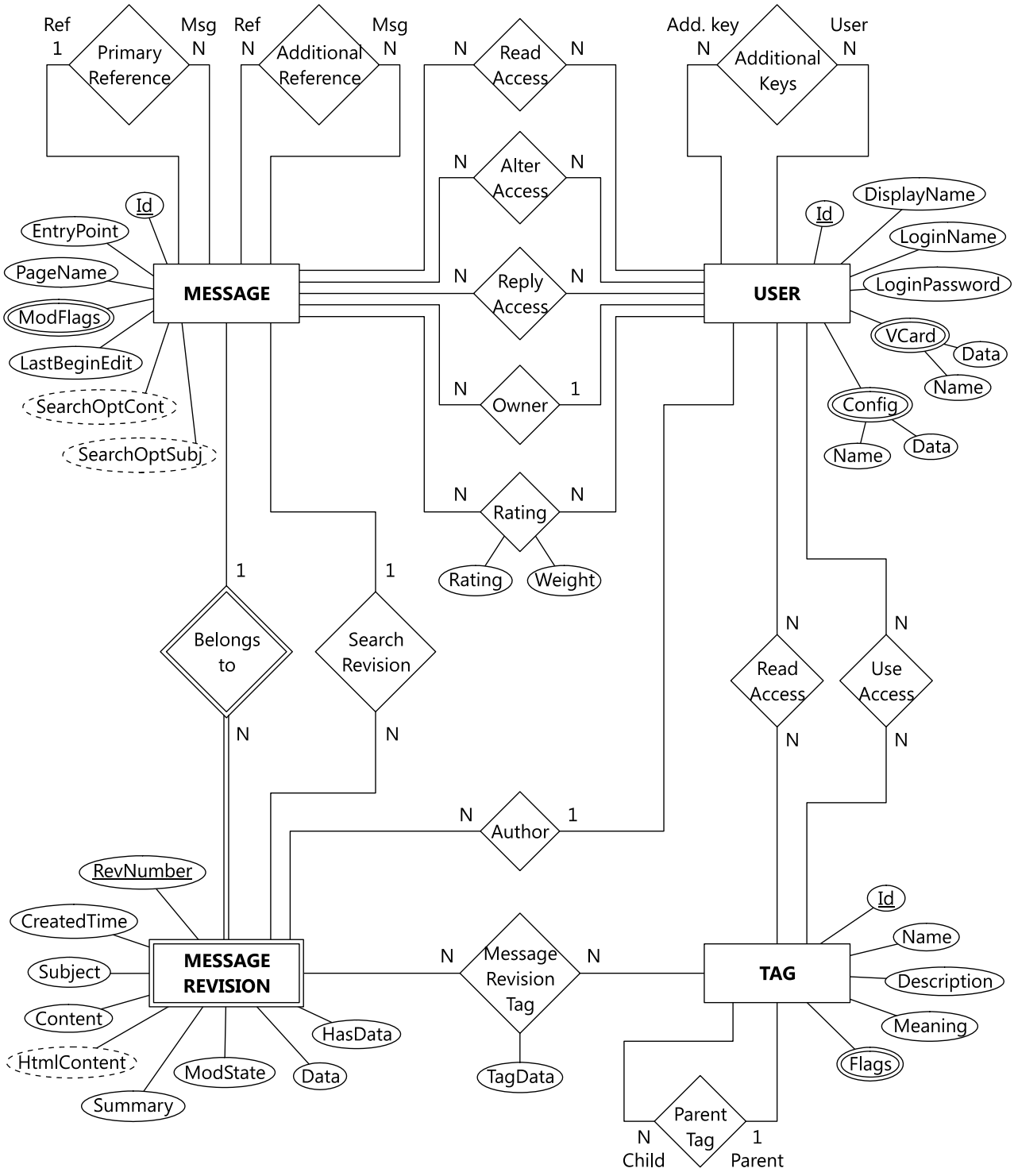
2.5.10. Tabelle user_vcard

Diese Tabelle enthält alle VCard-Einträge der Benutzer.

Feldname	Datentyp	Beschreibung
UserId	INTEGER UNSIGNED	ID des Benutzers, dem der VCard-Eintrag gehört
Type	VARCHAR(255)	Eintragstyp, z.B. RealName, EMail, Birthday, ICQ, ...

Data	TEXT	Wert des Eintrags
Visibility	TINYINT UNSIGNED DEFAULT 0	Gibt an, für wen dieser Eintrag sichtbar ist 0 = öffentlich 1 = intern (nur für angemeldete Benutzer) 2 = vertraulich (nur für vertrauenswürdige Benutzer) 3 = privat (nur für den Eigentümer) (siehe UnbUserVcardVisibility::*)

Das folgende Entity-Relationship-Diagramm stellt alle Zusammenhänge zwischen den vier Entitäten Benutzer, Mitteilung, Mitteilungsrevision und Tag dar. Es sind u.U. nicht alle Attribute dargestellt.



3. Anforderungen

3.1. Programmiersprache

Die UNB Components sind in der Programmiersprache PHP implementiert. Dabei werden Methoden der objektorientierten Programmierung verwendet, die ab Version 5 verfügbar sind. Außerdem wird die Datenbankabstraktion PHP Data Objects (PDO) verwendet, die ab Version 5.1 verfügbar ist. Zur Sicherung der Anmeldekennwörter in der Datenbank werden aktuelle kryptografische Hashfunktionen verwendet, die **PHP** ab Version **5.1.2** anbietet. Damit ist dies die minimale Version, mit der die UNB Components lauffähig sind.

Folgende PHP-Erweiterungen werden zusätzlich zu den standardmäßig aktivierten benötigt:

- PDO (PHP Data Objects, Datenbankabstraktion)
- Datenbankspezifischer Treiber für PDO, z.B. pdo_sqlite oder pdo_mysql, je nach verwendeter Datenbank
- mb_string (Multibyte Strings, für UTF-8-Unterstützung)
- mcrypt (Verschlüsselung, mind. Version 2.4)

Die Einstellungen bzgl. Register Globals, Magic Quoting und dem Error Reporting Level werden von der `UnbEnvironment`-Klasse erkannt und ausgeglichen bzw. angepasst. Hier gibt es keine speziellen Anforderungen.

3.2. Datenbank

Die UNB-Klassen sind darauf ausgelegt, mit verschiedenen Datenbanksystemen zu arbeiten und wurden mit mehreren Systemen getestet. Darunter sind folgende Systeme:

- **MySQL**, ab Version 5.0.15, mit InnoDB-Unterstützung
- **SQLite**, ab Version 3.2.3 – Da SQLite kein Row Level Locking unterstützt, muss in bestimmten Situationen die Datenbank vollständig gesperrt werden. Diese Sperren dauern nur sehr kurze Zeit an. Da SQLite beim Versuch, auf eine gesperrte Datenbank zuzugreifen, selbst keine Wiederholversuche unternimmt, wird dieser Umstand von der Datenbankklasse abgefangen und die Anweisung nach kurzen zufälligen Verzögerungen von wenigen Millisekunden mehrmals wiederholt. Außerdem erzwingt SQLite keine Fremdschlüsselbeziehungen. Stattdessen wurde diese Funktionalität mit Hilfe von Triggern nachgebildet. Diese können sich in bestimmten Situationen anders verhalten als andere Datenbanksysteme. SQLite erzwingt auch keine Datentypen für die Daten in den Tabellen. Es obliegt vielmehr der Anwendung, sich an das definierte Schema zu halten und z.B. keine Texte in Zahlenfelder zu schreiben.
- **PostgreSQL**, eingeschränkte Unterstützung ab Version 8.2 – Derzeit ist die Suchfunktion nur mit einem alternativen Codeabschnitt in der `UnbSearch`-Klasse lauffähig.

Microsoft SQL Server Express 2005 wird wegen mangelndem Verständnis der eingesetzten SQL-Syntax ausdrücklich *nicht* unterstützt. Version 2008 wurde noch nicht getestet. ~~Oracle Database Server 11g wird wegen der fehlenden Möglichkeit, Tabellen für Lesezugriffe zu sperren ausdrücklich *nicht* unterstützt. Diese Sperren sind in manchen Situationen notwendig, um die Konsistenz der Daten sicherzustellen, wo es SQL nicht unterstützt.~~ *(Hier sind neue Tests erforderlich.)*

Nicht genannte Datenbanksysteme können möglicherweise über die ODBC-Konfiguration verwendet werden.

Alle Daten werden in der UTF-8-Kodierung in der Datenbank gespeichert. Die getesteten Datenbanksysteme werden beim Aufbau der Verbindung von der Datenbankklasse entsprechend konfiguriert.

Das Datenbanksystem muss die ANSI-SQL-Syntax unterstützen, insbesondere werden Bezeichner in doppelte und Zeichenwerte in einfache Anführungszeichen eingeschlossen. MySQL kann dafür explizit in den ANSI-Modus versetzt werden (was die Datenbankverbindungs-Klasse und das Setup-SQL-Skript automatisch erledigen), andere Systeme unterstützen diese Syntax bereits von sich aus.

4. Funktionen im Detail

Bewertung von Beiträgen:

Gewichtung von Bewertungen abhängig von eigenen Bewertungen

Es soll nur ein einzelner Beitrag inhaltlich und objektiv bewertet werden.

Kriterien: nützlich, informativ, korrekt

Persönliche Zu- oder Abneigungen für den Autor des Beitrags sollen möglichst nicht berücksichtigt werden.

Sollten diese im Widerspruch zu einer objektiven Bewertung des Inhalts stehen, sollte man besser nicht bewerten.

5. Referenz

Dieses Kapitel dient als Referenz für Administratoren und Entwickler.

5.1. API-Referenz

Eine vollständige Übersicht aller im Programmcode definierten Klassen und Methoden wird automatisch generiert und liegt im HTML-Format vor.

5.2. Konfigurationsoptionen

Diese Übersicht enthält alle unterstützten Konfigurationsoptionen. Diese Angaben können in einer Datei gemacht werden oder in der Datenbank abgelegt werden. Die Angaben aus der Datei werden bevorzugt verwendet, können aber im Gegensatz zu denen in der Datenbank nicht vom Programm selbst geändert werden. Die Angaben zur Datenbankverbindung werden nur aus der Datei gelesen. Da die Angaben in einer regulären PHP-Datei gespeichert sind, könnten diese Werte auch durch eigenen Programmcode berechnet werden. In der Datenbank lassen sich dagegen nur konstante Werte abspeichern.

`attachment.storage.database-max-size`

Datenmenge in Bytes bis zu der angehängte Dateien automatisch in der Datenbank abgespeichert werden. Mit dem Wert `0` werden alle Dateien im Dateisystem abgelegt. Mit dem Wert `-1` werden alle Dateien in der Datenbank abgelegt (das ist die Vorgabe). Mit dem vorgegebenen Datenbankschema können maximal 16 MB große Dateien in der Datenbank abgelegt werden.

`captcha.enabled`

CAPTCHA-Prüfung verwenden. Diese Option hat u.U. nur Auswirkung, wenn die PHP-Klasse `UnbCaptcha` existiert.

`database.dbname`

Datenbankname (nicht für SQLite)

`database.driver`

Treibername für die Datenbankverbindung. Gültige Namen sind derzeit: `mysql`, `odbc`, `pgsql` und `sqlite`. Es gibt keinen Vorgabewert, es muss also eine Datenbankkonfiguration angegeben werden.

`database.dsn`

Datenbank-DSN-Angabe (nur für ODBC)

`database.filename`

Datenbank-Dateiname, relativ zur Quelldatei der Datenbankklasse (nur für SQLite)

`database.hostname`

Hostname des Datenbankservers (nicht für SQLite)

`database.password`

Anmeldekennwort der Datenbankverbindung (nicht für SQLite)

`database.persistent`

Persistente Datenbankverbindung verwenden. Mit dieser Option wird PHP die Verbindung zur Datenbank aufrecht erhalten und bei einem späteren Webseiten-Zugriff wiederverwenden, was zu Zeitersparnis führt. Diese Option sollte nicht für ODBC verwendet werden, stattdessen ist das Connection Pooling von ODBC zu bevorzugen. Dieser Wert wird als `boolean` interpretiert, Vorgabe ist `false`.

`database.port`

Portnummer des Datenbankservers (optional; nicht für SQLite)

`database.schema`

Schemaname der Datenbank (optional; nur für PostgreSQL)

`database.synchronous`

Synchronisierungsmodus (optional; nur für SQLite; `OFF`, `NORMAL` oder `FULL`, siehe SQLite-Dokumentation). Vorgabe für SQLite 3 ist ist `FULL`.

`database.table-prefix`

Präfix der Datenbank-Tabellennamen (optional). Vorgabe ist leer (kein Präfix).

`database.unix-socket`

UNIX-Socket des Datenbankservers (optional; nur für MySQL, nicht zusammen mit `hostname`/`port`)

`database.username`

Anmelde-Benutzername der Datenbankverbindung (nicht für SQLite)

`encryption.key`

Systemschlüssel zur Verschlüsselung von Daten. Der eingesetzte Twofish-Algorithmus verwendet maximal 32 Bytes dieses Schlüssels. Wenn diese Angabe fehlt, wird die Vorgabe in `UnbSession::DefaultEncryptionKey` verwendet. Dieser Schlüssel wird dafür verwendet, bestimmte Daten in der Datenbank zu verschlüsseln. Aus Sicherheitsgründen sollte der Schlüssel nur in der Konfigurationsdatei gespeichert und nicht in der Datenbank abgelegt werden.

`moderation.approve-from-trusted`

Aktiviert die automatische Genehmigung neuer Mitteilungen von vertrauenswürdigen Benutzern. Dieser Wert wird als `boolean` interpretiert, Vorgabe ist `false`.

`path.file-uploads`

Pfad in dem hochgeladene angehängte Dateien gespeichert werden. Dieser Pfad ist relativ zum aktuellen Verzeichnis, es können jedoch auch absolute Pfadnamen angegeben werden. Vorgabe ist das aktuelle Verzeichnis (`.`). Es wird automatisch ein Schrägstrich (`/`) angehängt falls erforderlich. Dieses Verzeichnis muss existieren und beschreibbar sein.

5.3. Benutzerkonfiguration

Diese Übersicht enthält alle vom System verwendeten Benutzerkonfigurationswerte. Diese Angaben werden für jeden Benutzer separat gespeichert und können verwendet werden, um dem Benutzer die Anpassung seiner Programmoberfläche oder des Programmverhaltens zu ermöglichen.

`autologin.password`

Zweitkennwort, das zur automatischen Anmeldung des Benutzers in seinem Browser gespeichert wird. Diese Angabe wird mit dem Systemschlüssel verschlüsselt in der Datenbank gespeichert. Wenn dieses Zweitkennwort an Dritte gelangt, kann der Benutzer es durch ein anderes ersetzen. Anschließend muss er sich auf anderen PCs (Browsern) einmalig erneut mit seinen Anmeldedaten anmelden.

5.4. Web-Interface-Formularfelder

Diese Übersicht enthält alle Aktionen, die von der automatischen Formularverarbeitung (siehe Kapitel 2.3.1) erkannt werden und nennt die dazu verwendeten Formularfelder und ihren Datentyp oder Wertebereich. Technisch können in HTML-Formularen nur Zeichenkettenwerte übertragen werden. Die Angabe „bool“ bedeutet, dass nur zwischen „0“ und einer von 0 verschiedenen Zahl (z.B. „1“) unterschieden wird.

Aktion	Feldname	Typ	Beschreibung
login	Führt eine Benutzeranmeldung oder Registrierung durch und setzt das Auto-Login-Cookie. Überspringt die Aktionen `set_user_loginname`, `set_user_loginpassword`, `set_user_displayname`.		
	`implicit_register`	bool	Gibt an, ob eine implizite Registrierung durchgeführt werden soll.
	`user_loginname`	string	Anmeldename
	`user_loginpassword`	string	Anmeldekennwort
	`allow_register`	bool	Gibt an, ob ein neuer Benutzer erstellt werden soll, wenn der Anmeldename unbekannt ist.
	`user_displayname`	string	Anzeigename (bei Registrierung)
	`user_repeatpassword`	string	Wiederholung des Kennworts (bei Registrierung)
	`captcha_word`	string	CAPTCHA-Eingabe (bei Registrierung)
logout	Meldet den Benutzer ab.		
			Keine Parameter
create_message	Erstellt eine neue Mitteilung. Überspringt die Aktionen `set_message_entrypoint`, `set_messagerevision_filedata`.		
	`message_entrypoint`	bool	Gibt an, ob die neue Mitteilung ein DEP ist.
	`message_subject`	string	Betreff
	`message_content`	string	Mitteilungsinhalt
	`message_filedata`	file	Hochgeladene angehängte Datei
	`replyto_messageid`	int	Mitteilungs-ID der primären Referenz
set_messagerevision_tags	Setzt die Tags einer Mitteilungsrevision.		
	`messageid`	int	Mitteilungs-ID (bei bestehender Mitteilungen)
	`revisionnumber`	int	Revisionsnummer (bei bestehender Revision)
	`message_tagid`	array	Zu setzende Tag-IDs
set_message_pagename	Setzt den Seitennamen einer Mitteilung.		
	`messageid`	int	Mitteilungs-ID (bei bestehender Mitteilungen)
	`message_pagename`	string	Seitenname
set_message_entrypoint	Legt fest, ob eine Mitteilung ein Diskussionseinstiegspunkt (DEP) ist.		
	`messageid`	int	Mitteilungs-ID (bei bestehender Mitteilungen)
	`message_entrypoint`	bool	DEP-Wert
set_message_readaccess	Setzt die ReadAccess-Liste einer Mitteilung.		
	`messageid`	int	Mitteilungs-ID (bei bestehender Mitteilungen)
	`message_ readaccess_empty`	bool	Gibt an, ob die Eingabe leer ist. Mit übertragene Hinweistexte werden nicht ausgewertet.
	`message_ readaccesslist`	string	Mehrzeilige Liste mit Schlüsselnummern oder Anzeigenamen der Benutzer
set_message_alteraccess	Setzt die AlterAccess-Liste einer Mitteilung.		
	`messageid`	int	Mitteilungs-ID (bei bestehender Mitteilungen)
	`message_ alteraccess_empty`	bool	Gibt an, ob die Eingabe leer ist. Mit übertragene Hinweistexte werden nicht ausgewertet.
	`message_ alteraccesslist`	string	Mehrzeilige Liste mit Schlüsselnummern oder Anzeigenamen der Benutzer
set_message_replyaccess	Setzt die ReplyAccess-Liste einer Mitteilung.		
	`messageid`	int	Mitteilungs-ID (bei bestehender Mitteilungen)
	`message_ replyaccess_empty`	bool	Gibt an, ob die Eingabe leer ist. Mit übertragene Hinweistexte werden nicht ausgewertet.

	message_replyaccesslist	string	Mehrzeilige Liste mit Schlüsselnummern oder Anzeigenamen der Benutzer
alter_message	Ändert eine Mitteilung (fügt eine neue Revision hinzu).		
	messageid	int	Mitteilungs-ID (bei bestehender Mitteilungen)
	revisionnumber	int	Revisionsnummer (bei bestehender Revision)
	message_subject_hash	string	MD5-Hash des vorherigen Betreffs. Wenn der Betreff, der Inhalt und die Datei nicht verändert und keine Änderungsnotiz eingegeben wurden, wird keine neue Revision angelegt.
	message_subject	string	Neuer Betreff
	message_content_hash	string	MD5-Hash des vorherigen Inhalts.
	message_content	string	Neuer Inhalt
	message_summary	string	Änderungsnotiz
	set_messagerevision_filedata	bool	Gibt an, ob das Formular Angaben über eine angehängte Datei enthält.
	message_cleardata	bool	Gibt an, ob die angehängte Datei gelöscht werden soll.
	message_filedata	file	Hochgeladene angehängte Datei
delete_message	Löscht eine Mitteilung.		
	messageid	int	Mitteilungs-ID (bei bestehender Mitteilungen)
moderate_message	Moderiert eine Mitteilung.		
	messageid	int	Mitteilungs-ID (bei bestehender Mitteilungen)
	message_locked	bool	Gibt an, ob die Mitteilung gesperrt werden soll.
	message_hidden	bool	Gibt an, ob die Mitteilung versteckt werden soll.
	message_enforce_approval	bool	Gibt an, ob Antworten auf die Mitteilung manuell genehmigt werden müssen.
rate_message	Bewertet eine Mitteilung.		
	messageid	int	Mitteilungs-ID (bei bestehender Mitteilungen)
	message_rating	int	Bewertung (-10 bis 10)
clear_rate_message	Entfernt die eigene Bewertung eine Mitteilung.		
	messageid	int	Mitteilungs-ID (bei bestehender Mitteilungen)
moderate_messagerevision	Moderiert eine Mitteilungsrevision.		
	messageid	int	Mitteilungs-ID (bei bestehender Mitteilungen)
	revisionnumber	int	Revisionsnummer (bei bestehender Revision)
	message_modstate	int	Neuer Moderationsstatus
create_tag	Erstellt ein neues Tag. Überspringt die Aktionen `set_tag_name`.		
	tag_name	string	Tag-Name
set_tag_name	Setzt einen neuen Tag-Namen.		
	tagid	int	Tag-ID (bei bestehendem Tag)
	tag_name	string	Tag-Name
set_tag_description	Setzt eine neue Tag-Beschreibung.		
	tagid	int	Tag-ID (bei bestehendem Tag)
	tag_description	string	Tag-Beschreibung
set_tag_flags	Setzt neue Tag-Flags.		
	tagid	int	Tag-ID (bei bestehendem Tag)
	tag_flag_notselectable	bool	Gibt an, ob das Tag nicht auswählbar sein soll.
	tag_flag_invisible	bool	Gibt an, ob das Tag unsichtbar sein soll.

	tag_flag_ enforceapproval	bool	Gibt an, ob Mitteilung mit diesem Tag manuell genehmigt werden müssen.
	tag_flag_userdata	bool	Gibt an, ob zu dem Tag ein Textwert gespeichert werden kann.
set_tag_parent	Setzt ein neues übergeordnetes Tag.		
	tagid	int	Tag-ID (bei bestehendem Tag)
	tag_parentid	int	Tag-ID des übergeordneten Tags, oder 0
set_tag_readaccess	Setzt die ReadAccess-Liste eines Tags.		
	tagid	int	Tag-ID (bei bestehendem Tag)
	tag_readaccess_ empty	bool	Gibt an, ob die Eingabe leer ist. Mit übertragene Hinweistexte werden nicht ausgewertet.
	tag_readaccesslist	string	Mehrzeilige Liste mit Schlüsselnummern oder Anzeigenamen der Benutzer
set_tag_useaccess	Setzt die UseAccess-Liste eines Tags.		
	tagid	int	Tag-ID (bei bestehendem Tag)
	tag_useaccess_empty	bool	Gibt an, ob die Eingabe leer ist. Mit übertragene Hinweistexte werden nicht ausgewertet.
	tag_useaccesslist	string	Mehrzeilige Liste mit Schlüsselnummern oder Anzeigenamen der Benutzer
delete_tag	Löscht ein Tag.		
	tagid	int	Tag-ID (bei bestehendem Tag)
create_user	Erstellt einen neuen Benutzer. Überspringt die Aktionen `set_user_loginname`, `set_user_loginpassword`, `set_user_displayname`.		
	user_displayname	string	Anzeigename
	user_loginname	string	Anmeldename
	user_loginpassword	string	Anmeldekennwort
	user_repeatpassword	string	Kennwortwiederholung
set_user_additionalkeys	Weist einem Benutzer zusätzliche Schlüssel zu. Überspringt die Aktionen `set_user_administrator`, `set_user_moderator`, `set_user_trusted`.		
	userid	int	Benutzer-ID (bei bestehendem Benutzer)
	user_additionalkeys _empty	bool	Gibt an, ob die Eingabe leer ist. Mit übertragene Hinweistexte werden nicht ausgewertet.
	user_ additionalkeyslist	string	Mehrzeilige Liste mit Schlüsselnummern oder Anzeigenamen der Benutzer
	user_is_ administrator	bool	Wählt den Administratorschlüssel aus (wenn Aktion `set_user_administrator` angefordert wird)
	user_is_moderator	bool	Wählt den Moderatorschlüssel aus (wenn Aktion `set_user_moderator` angefordert wird)
	user_is_trusted	bool	Wählt den Trusted-Schlüssel aus (wenn Aktion `set_user_trusted` angefordert wird)
set_user_administrator	Weist einem Benutzer den Administratorschlüssel zu oder entfernt ihn.		
	userid	int	Benutzer-ID (bei bestehendem Benutzer)
	user_is_ administrator	bool	Wählt den Administratorschlüssel aus
set_user_moderator	Weist einem Benutzer den Moderatorschlüssel zu oder entfernt ihn.		
	userid	int	Benutzer-ID (bei bestehendem Benutzer)
	user_is_moderator	bool	Wählt den Moderatorschlüssel aus (wenn Aktion `set_user_moderator` angefordert wird)

set_user_trusted		Weist einem Benutzer den Trusted-Schlüssel zu oder entfernt ihn.	
	`userid`	int	Benutzer-ID (bei bestehendem Benutzer)
	`user_is_trusted`	bool	Wählt den Trusted-Schlüssel aus (wenn Aktion `set_user_trusted` angefordert wird)
set_user_displayname		Setzt den Anzeigenamen eines Benutzers.	
	`userid`	int	Benutzer-ID (bei bestehendem Benutzer)
	`user_displayname`	string	Neuer Anzeigename
set_user_loginname		Setzt den Anmeldenamen eines Benutzers.	
	`userid`	int	Benutzer-ID (bei bestehendem Benutzer)
	`user_loginname`	string	Neuer Anmeldename
	`user_currentpassword`	string	Aktuelles Kennwort (wird zur Änderung benötigt)
set_user_loginpassword		Setzt das Anmeldekennwort eines Benutzers.	
	`userid`	int	Benutzer-ID (bei bestehendem Benutzer)
	`user_loginpassword`	string	Neues Anmeldekennwort
	`user_repeatpassword`	string	Kennwortwiederholung
	`user_currentpassword`	string	Aktuelles Kennwort (wird zur Änderung benötigt)
reset_user_autologin_password		Setzt ein neues Zweitkennwort zur automatischen Anmeldung mit Cookies.	
	`userid`	int	Benutzer-ID (bei bestehendem Benutzer)
	`reset_user_autologin_password_value`	string	Muss „1“ sein, damit die Aktion ausgeführt wird.
delete_user		Löscht einen Benutzer.	
	`userid`	int	Benutzer-ID (bei bestehendem Benutzer)
set_user_vcard		Setzt einen Benutzer-VCard-Eintrag.	
	`userid`	int	Benutzer-ID (bei bestehendem Benutzer)
	`vcard_data_*`	string	Neuer Wert für den Eintrag „*“
	`vcard_remove_*`	string	Gibt an, ob der Eintrag „*“ gelöscht werden soll.
	`vcard_visibility_*`	int	Sichtbarkeit des Eintrags „*“
	`vcard_newtype`	string	Name des neu hinzuzufügenden Eintrags
	`vcard_newdata`	string	Wert des neu hinzuzufügenden Eintrags
optimise_database		Optimiert die Datenbank.	
	`optimise_db_extensive`	bool	Gibt an, ob eine ausführliche Optimierung erfolgen soll. Näheres zur Bedeutung des Werts ist der Beschreibung der Methode `UnbDatabase::Optimise` zu entnehmen.

5.5. Fehlermeldungen

Diese Übersicht enthält alle Fehlermeldungs-Nummern, die in verschiedenen Situationen in Fehlermeldungen enthalten sind, um bestimmte Fehlersituationen automatisiert zu erkennen und entsprechend zu behandeln. Eine Fehlernummer besteht aus einer Ganzzahl und wird im folgenden Format in einer zurückgegebenen Fehlermeldung eingesetzt:

```
"[#1234] Fehlermeldung"
```

Dabei ist 1234 der Fehlercode und „Fehlermeldung“ die textuelle Fehlermeldung, die zusätzliche Informationen enthalten kann.

Fehlermeldungen werden durch Exceptions mitgeteilt. Manche Fehler sind bereits am Typ der Exception erkennbar, für allgemeinere Fälle oder wenn Benutzerfehler angenommen werden können, werden zusätzlich die hier beschriebenen Fehlernummern eingefügt. Folgende Exception-Typen werden verwendet:

InvalidArgumentException	Ungültiger Typ oder Wert eines Funktionsarguments. Wird oft durch Implementierungsfehler ausgelöst.
BadMethodCallException	Aufruf der Funktion wird zur Zeit nicht unterstützt. Wird oft durch Implementierungsfehler ausgelöst.
UnbDatabaseException	Fehler bei der Ausführung einer Datenbankabfrage. Wird oft durch Implementierungsfehler oder technische Probleme des Datenbanksystems ausgelöst.
UnbSecurityException	Keine Berechtigung zur Ausführung der Funktion. Kann auch durch Implementierungsfehler ausgelöst werden.
LengthException	Zulässige Datenlänge überschritten. Wird oft durch Eingabefehler ausgelöst.
Exception	Allgemeiner Fehler, siehe Fehlercodes-Tabelle.

Fehlercodes werden nach ihrer Integration ins Programm fortlaufend vergeben. Dadurch ist eine inhaltliche Gruppierung durch Zahlenvergleich nicht möglich. Gelöschte Nummern werden nicht wieder vergeben.

1	Die Konfigurationsdatei ist defekt.
2	Der Seitenname ist zu lang (maximal 255 Zeichen).
3	Der Inhalt ist zu lang (maximal 16.777.215 Zeichen).
4	Der Betreff ist zu lang (maximal 255 Zeichen).
5	Die Zusammenfassung ist zu lang (maximal 255 Zeichen).
6	Die anzuhängende Datei kann nicht kopiert/verschoben werden.
7	Die anzuhängenden Daten konnten nicht in die Datei geschrieben werden.
8	Die anzuhängenden Daten konnten nicht aus der Datei gelesen werden.
9	Die angehängte Datei wurde nicht gefunden.
10	Die angehängte Datei konnte nicht gelöscht werden.
11	Die Tag-Daten sind zu lang (maximal 255 Zeichen).
12	Eine Benutzeranmeldung ohne Kennwort ist nicht zulässig.
13	Falsches Kennwort angegeben.
14	Die Tag-Bezeichnung ist zu lang (maximal 255 Zeichen).
15	Die Tag-Beschreibung ist zu lang (maximal 255 Zeichen).
16	Der Anzeigename ist zu lang (maximal 255 Zeichen).
17	Der Anmeldename ist zu lang (maximal 255 Zeichen).
18	Anmeldename und Kennwort müssen entweder beide angegeben werden oder keines der beiden.
19	Das Kennwort wird als unsicher angesehen und deshalb nicht akzeptiert. (Begründung in der Fehlermeldung.)
20	Der Anzeigename ist bereits vergeben.
21	Der Anmeldename ist bereits vergeben.
22	Keine freie Benutzer-ID gefunden. Die Schlüssellänge sollte erhöht werden.
23	Der VCard-Typname ist zu lang (maximal 255 Zeichen).
24	Die VCard-Daten sind zu lang (maximal 65.535 Zeichen).
25	Der Konfigurationsname ist zu lang (maximal 255 Zeichen).
26	Die Konfigurationsdaten sind zu lang (maximal 255 Zeichen).
27	Kein Anmeldename zur Anmeldung angegeben.
28	Kein Kennwort zur Anmeldung angegeben.
29	Kein Anzeigename zur Registrierung angegeben.
30	Falsche Kennwortwiederholung angegeben.
31	Falsches CAPTCHA-Wort angegeben.
32	Der Anmeldename ist unbekannt.

33	Kein Mitteilungsinhalt angegeben.
34	Falsches aktuelles Kennwort angegeben.
35	Die Benutzer-ID ist ungültig.
36	Die Benutzer-ID ist unbekannt.

5.6. Code-Konventionen

In diesem Kapitel werden alle Richtlinien beschrieben, nach denen sich die Organisation des Quelltextes und Benennungen von Funktionen, Feldnamen etc. richten. Durch die Einhaltung dieser Richtlinien werden eine gleichbleibende Code-Qualität erreicht und häufige Fehlerquellen in ihrer Ursache ausgeschlossen. Die Einhaltung dieser Richtlinien ist nicht zwingend notwendig, maximiert jedoch die Portabilität und Stabilität des Programms und wird daher dringend empfohlen.

5.6.1. Code-Formatierung

- Alle Funktionen, Methoden, Klassenfelder und Konstanten müssen dokumentiert werden und zumindest deren Funktion bzw. Bedeutung und ggf. die Aufrufparameter beschreiben.
- Einrückungen mit Tabulator-Zeichen (Empfehlung: Darstellung mit 4 Leerzeichen)
- Keine Leerzeichen zwischen Funktionsnamen und Parameterliste oder innerhalb von runden oder eckigen Klammern. Zwischen Schlüsselwörtern (`catch`, `die`, `echo`, `for`, `foreach`, `if`, `include`, `require`, `return`, `switch`, `throw`, `while`) und ihren Parametern steht ein Leerzeichen. Bei spracheigenen Anweisungen (z.B. `echo`, `include`) werden keine Klammern um den/die Parameter gesetzt. Um (binäre) Operatoren (z.B. `+`, `-`, `&&`, `.`, **außer** `++`, `--`, `!`, `~`, `^`, `->`, `::`) werden Leerzeichen gesetzt.
- Geschweifte Klammern (`{`, `}`) stehen in einer eigenen Zeile und auf der Einrückungsebene der übergeordneten Anweisung.
- Die Nutzung geschweifter Klammern zur Blockbildung und das Absetzen von bedingten Anweisungen (z.B. nach `if`) in eine neue Zeile ist freibleibend, der Code muss aber übersichtlich bleiben. Kurze Anweisungen nach kurzen Bedingungen können z.B. auch in derselben Zeile stehen.
- Klassennamen und Namen öffentlicher Klassenmethoden und -felder beginnen mit einem Großbuchstaben, lokale Variablen und private Methoden und Felder mit einem Kleinbuchstaben. Ansonsten wird für Bezeichner CamelCase verwendet, keine Unterstriche (_).
- Dateinamen für Klassen enden mit „.class.php“, für Funktionsbibliotheken mit „.lib.php“ und für sonstige einzubindende Dateien mit „.inc.php“.

5.6.2. Unicode

- Alle String-Funktionen, die möglicherweise mit Unicode-Eingaben in Kontakt kommen, müssen `mb_*` verwenden, z.B. `mb_strtolower()`. Der UTF-8-Zeichensatz wird durch `UnbEnvironment` voreingestellt.

5.6.3. Datenbank

- Alle Datenbankabfragen (außer einzelne `SELECT`s) müssen in einer Transaktion stattfinden
- Funktionen, die innerhalb einer Datenbank-Transaktion andere Funktionen aufrufen, die Exceptions werfen können, müssen darauf reagieren, und die Transaktion auf ihrer Ebene korrekt beenden (`COMMIT` oder `ROLLBACK`), bevor sie die Exception ggf. weiterwerfen, damit die Transaktion am Ende nicht offen bleibt.
- Tabellennamen bestehen nur aus Kleinbuchstaben (Portabilität, v.a. MySQL unter Unix/Windows).
- Alle Tabellen-/Spaltennamen (Bezeichner, *Identifier*) werden in SQL-Abfragen mit doppelten Anführungszeichen (nach ANSI-SQL) „gequotet“.
- Alle Bezeichner in der Datenbank und im Code sind auf englisch.

- Entity-Bezeichnungen sind im Singular formuliert (z.B. „user“ statt „users“).
- Statt einfach nur `ID` wird immer der Typ der ID verwendet, z.B. `UserId`, `MessageId` (das vereinfacht Tabellen-`JOIN`s).
- Integer-Typen sollen nur so groß sein wie notwendig, nicht einfach überall `INTEGER` verwenden; auch `UNSIGNED`-Typen verwenden (sofern unterstützt, wie bei MySQL).
- Die maximale Länge von `VARCHAR`-Typen kann ruhig 255 Zeichen betragen, der Speicherplatzbedarf richtet sich immer nach dem Inhalt. So hat man später keine unnötigen Einschränkungen.
- Von Anfang an überlegen, ob ein Feld `NULL` werden darf oder nicht.

5.6.4. API

- Alle Parameter, die API-Funktionen übergeben werden, werden auf gültige Datentypen und grob gültige Werte geprüft (z.B. negative Zahlen, wo nur positive erwartet werden; leere Zeichenketten oder Arrays; zulässige Zeichen/Format).
- Übergebene Zeichenketten werden auf ihre zulässige Länge (siehe Datenbankstruktur) geprüft.
- Außerdem finden Normalisierungen wie `mb_strtolower()` oder `trim()` statt. Optionale leere Zeichenketten werden wo sinnvoll durch `null` ersetzt.
- Die Zugriffsrechte werden endgültig von den API-Funktionen erzwungen, sollten aber teilweise bereits durch die Anwendung beachtet werden, um unnötige Fehlermeldungen zu vermeiden.
- Alle API-Funktionen sollten atomar erfolgreich sein oder fehlschlagen. Im Fehlerfall darf es keine Nebenwirkungen oder Teilausführung geben. (Dieses Paradigma sollte bis an die Benutzeroberfläche durchgesetzt werden.)

6. Häufig gestellte oder anderweitig interessante Fragen

6.1. Warum heißt es „Mitteilung“ und nicht „Beitrag“ oder „Posting“?

In Webforen ist häufig von Beiträgen die Rede, aus dem Englischen wird auch öfter das Wort Posting verwendet. Das trifft die wahre Bedeutung aber nicht genau. Ein Beitrag, im Englischen wohl am ehesten als „contribution“, manchmal auch „article“ übersetzt, beschreibt den Umstand, dass jemand etwas beigesteuert hat, seinen Beitrag geleistet hat. Das Posting ist neben einer „Entsendung“ mehr der eigentliche Vorgang des Absendens, hier einer Mitteilung, an den Webserver. Man könnte es auch „submit/submission“ nennen. Was ich aber eigentlich bezeichnen will, ist ein Textstück, das entweder von Benutzern oder automatischen Systemen generiert wird und eine bestimmte Nachricht enthält. Hier wird also etwas mitgeteilt, daher die „Mitteilung“.

6.2. Warum werden Mitteilungen in Diskussionen in einer Baumstruktur verwaltet und nicht linear?

Für die Übersichtlichkeit der Mitteilungen in einer längeren Diskussion ist es unerlässlich, genau festzuhalten, welche Mitteilung auf welche antwortet. Denn in der Realität ist es erfahrungsgemäß nunmal so, dass sich Diskussionen in verschiedene Richtungen entwickeln, die einzeln weiter behandelt werden. Wozu gibt es mehrere Threads? Für verschiedene Themenrichtungen. Warum sollte nach der ersten Mitteilung eines Threads damit Schluss sein? Warum sollen abweichende Richtungen von diesem Moment an zum Untergang verurteilt sein, sofern sich nicht ein Moderator erbarmt, entsprechende Korrekturen durchzuführen – die bei vielen Softwarelösungen im Übrigen sehr aufwändig sind, wenn ein Autor mehrere Teilbeiträge auf einmal veröffentlicht hat, die zunächst aufwändig getrennt werden müssen. In einer linearen, flachen Struktur der Antworten ist es einfach so, dass eher unauffällige Randbemerkungen oder Fragen untergehen, weil sie spätestens auf der nachfolgenden Seite nicht mehr gesehen werden und sich niemand daran erinnert. In einer Baumstruktur fällt so eine Antwort eher auf. Außerdem ist hier deutlich besser erkennbar, wer auf wen geantwortet hat.

Die Baumansicht von Diskussionen leidet unter weit verbreiteten Vorurteilen der Benutzer, unabhängig von ihrer Erfahrung mit derartigen Systemen. Ihnen wird Unübersichtlichkeit und umständliche Handhabung nachgesagt. Dabei sind für diese Nachteile überwiegend die Implementierungen solcher Systeme verantwortlich, nicht das Konzept selbst. In E-Mail-Programmen funktioniert das besser, da beide Teile permanent sichtbar sind (d.h. lange Mitteilungen nicht die Baumansicht wegblättern), einzelne Nachrichten praktisch ohne Verzögerung angezeigt werden können (im Gegensatz zu den traditionell langsameren Webseiten) und man nicht so oft wegen jedem Einzeiler (oder „Einwörter“) eine Nachricht verfasst. Diese Vorteile gilt es nun ins Web zu übertragen.

Dafür müssen einige Punkte beachtet werden, zu denen u.a. folgende gehören:

- Es darf keine Wahl der Anzeigeform durch den Benutzer möglich sein. Benutzer, die Mitteilungen linear lesen, werden üblicherweise nicht gezielt auf Mitteilungen antworten, auf die sie sich beziehen. Sie haben auch keinen Grund dafür, denn ihre Antwort erscheint ja doch immer ganz am Ende der langen Liste.
- Der Baum mit allen Antworten muss permanent sichtbar sein und sollte nicht mit dem Mitteilungsinhalt geblättert werden. Ansonsten müsste man erst wieder zum Baum blättern und sich in der Struktur erneut orientieren, um die nächste oder eine andere Mitteilung auszuwählen. Eine seitliche Darstellung ist praktisch, da man so die volle Höhe nutzen kann und die Breite der Mitteilungen verringert wird. Und kurze Textzeilen lassen sich einfach besser lesen – Zeitungstexte sind auch aus diesem Grund mehrspaltig formatiert. Der Trend zu Widescreen-Bildschirmen unterstützt dieses Vorhaben.
- Es darf immer nur eine einzige Mitteilung angezeigt werden. Das erhöht das Bewusstsein des Benutzers für die Eigenständigkeit der Mitteilungen und fördert das Antworten auf die richtige Mitteilung, was die

6.3. Warum werden Mitteilungen ausschließlich durch Referenzen verknüpft und nicht in Threads gruppiert?

Mitteilungen sind uns aus heutigen Forensystemen oftmals als Teil eines Themenverlaufs (*Thread*) bekannt. Ein Thread besteht dabei aus einer ersten Mitteilung und den Antworten darauf, hierarchisch oder auch nicht. Ein Thread ist eine abgeschlossene Einheit, er hat einen Anfang und einen bestimmten Verlauf. Um eine Mitteilung zu veröffentlichen, braucht man zuerst einen Thread, und gibt es noch keinen passenden, wird einer eröffnet. Manchmal weiß man aber gar nicht so genau, in welchen Thread man seine Mitteilung veröffentlichen soll, und wählt den falschen aus. Gibt es dann auch noch Antworten auf diese falsch einsortierte Mitteilung, hat der Moderator die Arbeit, den Thread wieder zu bereinigen und die Mitteilungen zu verschieben. Tut er das nicht, findet man die Mitteilung nach einer Weile nicht mehr.

Mitteilungen allein durch eine Referenz auf eine andere Mitteilung in einen Zusammenhang einzuordnen ist da das praktischere Verfahren. Da gerade in langen Diskussionen öfter mal das Thema wechselt, kann man richtungswechselnde Mitteilungen als Diskussionseinstiege ansehen und sie dementsprechend markieren. Neue Mitteilungen verlieren so zuerst einmal den Bezug zu festen Threads, was gerade Einsteigern den Umgang vereinfachen dürfte: Neue Mitteilungen werden einfach nur noch veröffentlicht, *wo* ist gar nicht mehr so wichtig. Und ändert sich während einer Diskussion einmal das Gesprächsthema, markiert man die Mitteilung als neuen Einstiegspunkt, um so einen direkten Zugriff darauf zu vereinfachen. Dieser Einstieg in das neue Thema kann dabei durchaus eine oder gar mehrere Referenzen auf andere Mitteilungen haben, diese können zum Nachlesen angezeigt werden. Und sollte die neue Mitteilung gar keinen Bezug auf die angegebene Referenz nehmen (gelegentlich verwechseln Benutzer einfach die Funktionen „Antworten“ und „Neues Thema“), wird sie eben wieder entfernt, wodurch sich dynamisch die Verknüpfung zwischen einzelnen Mitteilungen ergibt. Mitteilungen ohne eine Referenz auf andere Mitteilungen (also keine Antworten) können dabei generell als Diskussionseinstieg angesehen werden.