1 Einleitung

Autor: Jonas Picker

Hier wird der Entwurf des Bibliotheksmanagementsystems BiBi beschrieben. Das Dokument bezieht sich auf das Lastenheft von Christian Bachmaier und Armin Größlinger und stellt eine technische Vertiefung unseres Pflichtenhefts dar, welches im Folgenden wiederholt referenziert wird.

2 Systemarchitektur

Autor: Ivan Charviakou

3 Klassendiagramm

Autor: Mohamad Najjar

4 JSF-Dialoge

Autor: León Liehr

5 Systemfunktionen

Autor: Jonas Picker

5.1 Technische Systemsicherheit

Kommunikationserschlüsselung: Durch die CA-Zertifizierung des Servers wird eine TLS-Transportverschlüsselung bei der Kommunikation zwischen Klient und Server verwendet. Ist die Datenbank, wie im Pflichtenheft (Abschnitt 4.3 'Server') beschrieben, über SSL-VPN angebunden, ist die Kommunikation zwischen ihr und dem Server ebenfalls verschlüsselt.
Nutzerberechtigungen: Für die Überprüfung der Zugangsberechtigungen bei jeder HTTPS-Anfrage implementiert die Klasse !!!!!! das PhaseListener-Interface, welches zu implementierende Methoden zum Ausführen von Code vor oder nach jeder Phase des JSF-Life-Cycle vorgibt. In unserem Fall wird am frühestmöglichen Punkt (vor der Restore-View-Phase) geprüft, ob der Anfragesteller auch berechtigt ist, die vorhergesehene Antwort vom Server zu erhalten. Aus Redundanzgründen verwenden wir für Seiten, auf die mehrere Nutzerrollen Zugriff haben, die gleichen Facelets. In den jeweiligen Backing-Beans wird dann, durch Zugriff auf die von JSF getrackte Nutzer-Session der Klasse !!!!!!, die Rolle des Benutzers überprüft. Für jeden Seitenbesucher werden dann nur die rollenspezifischen Knöpfe und Anzeigen gerendert (siehe !!!auch!!!). Durch die inklusive Rollenhierarchie lassen sich so alle Funktionalitäten der Rollen im gleichen Facelet einbinden.
Session-Hijacking[1]**:** Um diesen Angriffsvektor zu schützen, wird der Identifikator der Nutzer-Session an kritischen Stellen (z.B. nach dem Login) manuell ausgetauscht.

[1] Das Stehlen einer validen Nutzer-Session um Zugriff auf den Account und seine Berechtigungen zu erhalten

Cross-Site-Scripting[2]**:** Das Escapen von HTML-Sonderzeichen wird von JSF bei allen nutzergenerierten Teilen der Anwendung unterstützt. Alle Elemente der JSF-Facelets, die nutzergenerierten In- und Output weitergeben, haben den impliziten Standartwert 'escape=true'. Da wir diesen in unserer Implementierung nie manuell auf 'false' setzen, ist XSS in dieser Applikation unmöglich.
SQL-Injection[3]**:** Durch das Konsequente Verwenden der 'Prepared Statements' in unserer !!!Datenzugriffsschicht!!! beugen wir SQL-Injections vor. Diese JDBC-Funktionalität trennt das eigentliche SQL-Statement von den nutzergenerierten Parametern und Verhindern so das Ausführen von maliziösem SQL-Code in der Datenbank.

5.2 Logging

Mit der Klasse !!!!!! ist unser System mit einer eigenen Log-Funktion ausgestattet. Diese dient sowohl zum Debugging während der Entwicklung, als auch zum Protokollieren der Fehler und Abläufe im laufenden System. Durch Setzten der Variable 'LOG_CONSOLE: ' mit den Werten 'TRUE' oder 'FALSE' kann eingestellt in der Konfigurationsdatei werden, ob die Meldungen auch in Echtzeit auf der Konsole ausgegeben oder nur in das Log-File geschrieben werden. Es gibt drei Log-Level, zwischen denen, durch Setzen der Variable 'LOG_LEVEL: ' mit einem der unten aufgeführten Werte, beim Systemstart umgeschaltet werden kann. Die folgenden Kategorien sind inklusiv, die Letzte schließt somit die oberen beiden mit ein.
'SEVERE': Diese Einstellung des Loggers protokolliert nur schwere Fehler, die unmittelbare Konsequenzen für den Anwendungsbetrieb haben. Um ein schnelles Volllaufen des Log-Files zu vermeiden, ist dies die Standarteinstellung.
'DETAILED': Fehler und fehlgeschlagene Prozeduren, die die Integrität der Anwendung nicht gefährden, werden zusätzlich protokolliert.
'DEVELOPMENT': Hierunter fallen sowohl Protokollierungen von erfolgreichen oder seltenen Abläufen, alsauch sonstige nützliche Meldungen. Da das Log-File schnell sehr groß und unübersichtlich werden könnte, empfehlen wir, diese Option nur bei Problemen zu wählen.

5.3 Selbstständige Abläufe

Die Klasse !!!!!! ermöglicht dem System, selbstständig in einstellbaren Abständen Wartungsaufgaben durchzuführen. Alle unten aufgelisteten Aufgaben sind somit von der der aktuellen Systemzeit des Servers abhängig und werden darüberhinaus nur mit einer maximalen zeitlichen Unsicherheit des gewählten Intervalls durchgeführt. Die Einstellung wird beim Systemstart durch den gesetzten Wert der Variable 'SCAN_INTERVAL: ' bestimmt. Der von uns empfohlene Standartwert repräsentiert eine Minute, sollten Sie diesen verlängern wollen tragen Sie stattdessen eine andere positive ganze Zahl ein. Der Wartungsthread vergleicht die Operationsfrist der zur Abholung markierten Exemplare mit der aktuellen Systemzeit und setzt bei Überschreitung den Verfügbarkeitsstatus wieder auf 'verfügbar'. Bei Exemplaren mit dem Status 'ausgeliehen' wird, zusätzlich zur Frist, die Mahnungsversatzzeit beachtet und bei Bedarf die Klasse !!!!!! zum Versenden einer E-Mail angestoßen. Die abgelaufenen Tokens der Benutzer-Tabelle werden ebenfalls gelöscht.

[2] Auch XSS: das Einschleusen von browserinterpretierbarem HTML-Code auf ungesicherte Teile einer Website
[3] Einschleusen von SQL-Code in Formularfelder, um Zugriff auf die Datenbank zu erhalten

5.4 Datenbankverbindung

Wenn die Java Laufzeitumgebung des Systems plötzlich beendet wird und das System abstürzt, wird trotzdem mittels einer ShutdownHook noch das Schließen offener Datenbankverbindungen versucht. Außerdem wird durch das logische Zusammenfassen der voneinander abhängigen Datenbanktransaktionen in den Steuermethoden der !!!DAOs!!! sichergestellt, dass das ACID-Prinzip bestmöglich eingehalten und die Datenbank im Fehlerfall in einem konsistenten Zustand hinterlassen wird. Dies ist jedoch (z.B. bei einem Stromausfall) nicht immer möglich.

5.5 Starten und Stoppen der Anwendung

Start: Durch Implementierung des SystemEventListener-Interface werden beim Systemstart von der Klasse!!!!!! alle weiteren Aktionen angestoßen. Die Initialisierung des !!!Loggers!!! geschieht zuerst, danach wird die Konfigurationsdatei eingelesen, das Log-Level und der gewählte Konsolenausgabemodus gesetzt und mit den restlichen Parametern eine !!!eigene Initialisierungsklasse!!! für Prozesse der !!!Datenzugriffsschicht!!! aufgerufen, in der zunächst die Datenbankverbindung überprüft wird. Sollten die Verbindung fehlschlagen, wird, ähnlich zu !!!diesem Prozess!!!, eine Fehlermeldung ausgegeben/geloggt. Im Erfolgsfall wird dann überprüft, ob die benötigten Tabellenstrukturen bereits vorhanden sind. Falls nicht wird das manuelle Anlegen der Tabellen (Mediumsschema wird mit dem Standartattributsatz befüllt) über einen Konsoleninput mit 'Y' oder 'N' entschieden werden müssen. Bei existierenden Tabellen wird nun der !!!Connection-Pool!!! initialisiert und der Startprozess der !!!Datenzugriffsschicht!!! ist abgeschlossen. Es wird nun das zuvor ausgewählte Farbschema aus der Datenbank abgefragt, an die !!!darüberliegende Schicht!!! weitergeleitet und von !!!dort!!! aus der !!!Wartungsthread!!! aktiviert und das ausgewählte Farbschema eingestellt, danach ist das System betriebsbereit.
Stop: Beim planmäßigen Herunterfahren des Systems durch das Ausschalten des Tomcat auf dem Server (z.B. via shutdown.sh) reicht die Klasse !!!!!! eine Nachricht an die Klasse !!!!!!, von wo aus zunächst der !!!Wartungsthread!!! beendet und danach die noch offenen Datenbankverbindungen über den !!!Connection-Pool!!! geschlossen werden. Nach Durchreichen der Erfolgsmeldung nach oben, wird die Anwendung gestoppt. Nutzersessions überleben das Ausschalten des Servers nicht.

6 Datenfluss

Autor: Sergei Pravdin

7 ER-Modell

Autor: Jonas Picker

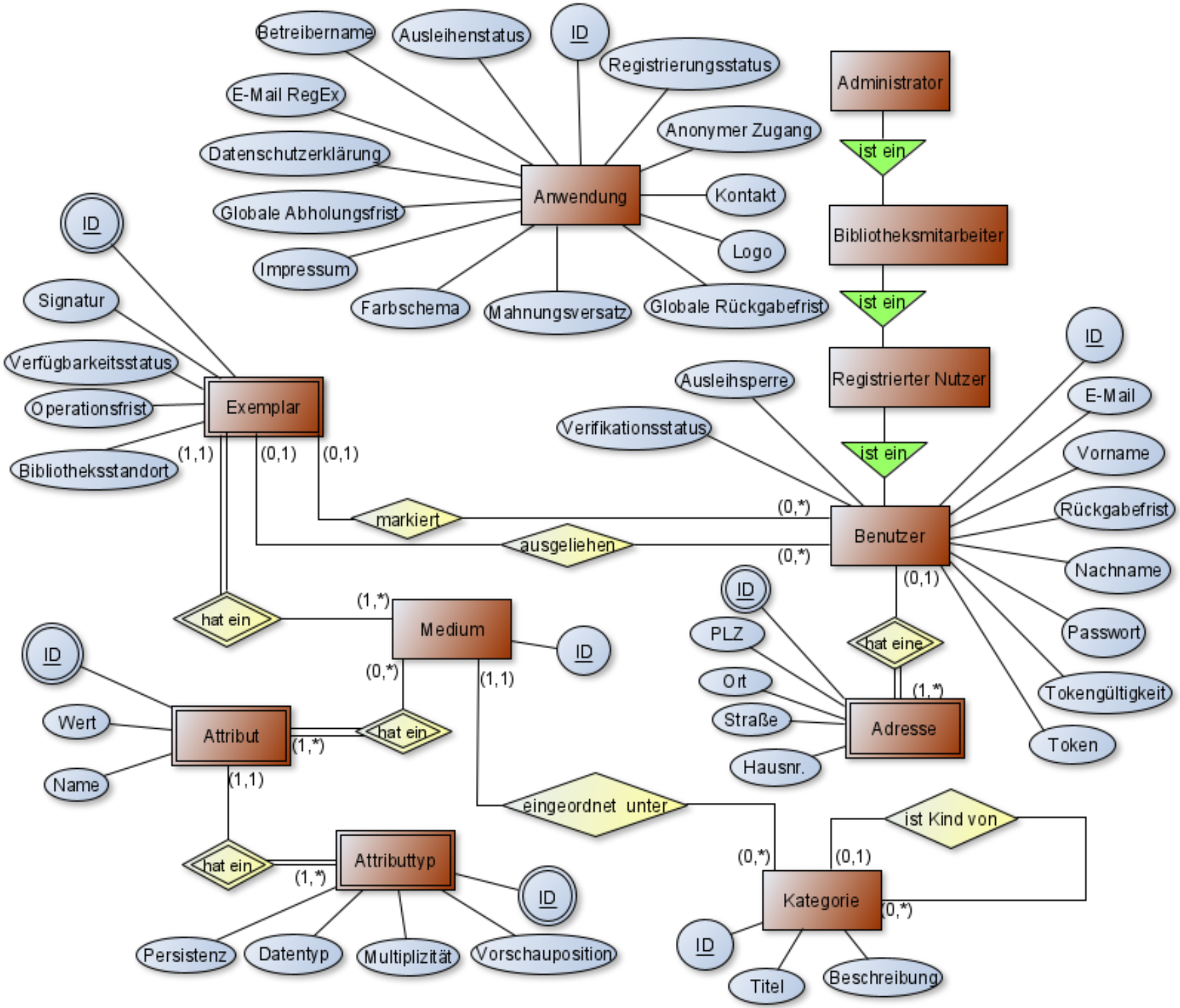

Abbildung 1: Entity-Relationship Diagramm

7.1 Legende

Beschreibung des ER-Diagramms:

Medium: Diese Entität modelliert die von der Bibliothek verwalteten Medien. Außer des Primärschlüssels besitzt sie mindestens ein zugehöriges Exemplar und eine variable Anzahl von Attributen. Der Standartattributsatz für Medien wird unten aufgeführt. Es können benutzerdefinierte Attribute hinzugefügt werden (PfHft. /F380/) und jedes der folgenden Attribute ist löschbar (PfHft. /F381/):

Titel	Erscheinungsjahr	Verlag
Medientyp	Version	Freitext
Autoren	Index (ISBN/ISSN)	Link auf elektronische Version

Attribut: Diese schwache Entität hält den Namen und Wert eines Medienattributs, ihr ist genau ein Attributtyp zugeordnet.
Attributtyp: Vom Attribut abhängige, schwache Entität. Hiermit werden dem Attribut zugehlrige Eigenschaften modelliert. 'Vorschauposition' bestimmt, ob und an welcher Stelle das Attribut in der Medienvorschau (z.B. in der Listenansicht der Suchergebnisse) angezeigt werden soll. 'Multiplizität' entscheidet, ob das zugehörige Attribut mehrfach pro Medium mit unterschiedlichen Werten vorkommen kann oder nicht. 'Persistenz' ist eine Markierung für Attribute, die nicht zum modifizierbaren Attributsatz gehören (z.B. die Rückgabefrist für Medien). 'Datentyp' beschreibt die im Attributwert gespeicherten Daten (z.B. 'String' für Textattribute oder 'Image' um Bilder zu speichern).
Kategorie: Die mit dieser Entität verbundenen Relationen ordnen jedem Medium genau eine Kategorie zu und modellieren die Kategoriehierarchie (PfHft. /W440/) durch die Selbstbeziehung. Es wird einen unlöschbaren Top-Knoten in der Hierarchie geben, zu dem alle Medien, die nie in eine Kategorie eingeteilt wurden oder deren Kategorie gelöscht wurde, gehören. Sollten alle Medien in benutzerdefinierten Kategorien stecken, hat der Top-Knoten keine zugeordneten Medien und nimmt somit nicht an der 'eingeordnet unter'-Relation teil.
Exemplar: Diese schwache Entität ist vom zugehörigen Medium abhängig. Ein bestimmtes Exemplar kann von genau einem Nutzer zur Abholung markiert oder ausgeliehen werden, diese Aktionen schließen sich gegenseitig aus (PfHft. /F310/) und ändern (genau wie eine Rückgabe) den Verfügbarkeitsstatus und die dazugehörige Operationsfrist dementsprechend. 'Signatur' und 'Bibliotheksstandort' halten bibliothekspezifische Kodierungen.
Benutzer: Ob ein Benutzer die Ausleihfunktion benutzen kann, wird durch das Attribut 'Ausleihsperre' modelliert. 'Verifizierungsstatus' zeigt hingegen an, ob der Nutzeraccount bereits den Verifizierungsprozess durchlaufen hat (PfHft. /W70/). Das Passwort wird in gehashter Form abgespeichert. Zur Passwortzurücksetzung und E-Mail-Verifikation wird pro Nutzer ein begrenzt gültiges, einzigartiges 'Token' verwendet, um den entsprechenden Link zu bauen. Die 'Rückgabefrist' für Ausleihen eines Benutzers wird ebenfalls modelliert.
Registrierter Nutzer: Diese Nutzer haben positiven 'Verifizierungsstatus'.
Bibliotheksmitarbeiter: Diese Entität modelliert die Rolle der Bibliotheksmitarbeiter.
Administrator: Diese Entität modelliert die Rolle der Administratoren. Die von der Benutzerentität ausgehende Hierarchie soll die inklusiven Rollen im System modellieren.
Adresse: Vom Benutzer abhängig. Enthält die Bestandteile einer Nutzeradresse als Attribute.
Anwendung: Hier werden die setzbaren globalen Variablen und Anwendungseinstellungen modelliert. 'Anonymer Zugang' bestimmt die Berechtigungen anonymer Nutzer beim Besuchen des

Webspaces (PfHft. /F10/), während 'Registrierungsstatus' eine offene (in diesem Fall gilt der 'E-Mail RegEx') oder geschlossene Registrierungsfunktion modelliert (PfHft. /F20/). 'Ausleihenstatus' steht für das Umschalten des Systems zur manuellen Freischaltung der Ausleihfunktion. Aus dem 'Mahungsversatz' ergibt sich der Zeitpunkt vor Ablauf einer Rückgabefrist, an dem eine automatische Benachrichtigung versendet wird (PfHft. /F240/). 'Farbschema' merkt sich das zuletzt ausgewählte Farbschema.