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

Kommunikationsverschlüsselung: Durch das vorrausgesetzte SSL-Zertifikat des Tomcat-Servers wird der Anwendung die Kommunikation mit dem HTTPS-Protokoll ermöglicht. Dies ist eine Ergänzung der HTTP-Kommunikation um eine Transportverschlüsselung. Zusätzlich zur Vereitelung von Abhörversuchen der an Ihren Server gesendeten Anfragen signalisieren Sie den Klienten so auch die Vertrauenswürdigkeit Ihrer Institution durch die CA-Zertifizierung.
Nutzerberechtigungen: Um zu Verhindern, dass Nutzer ohne ausreichende Berechtigungen auf zugangsbeschränkte Bereiche des Webspaces zugreifen können, werden entsprechende Maßnahmen ergriffen: Um den direkten Aufruf von Seiten, die für den Nutzer nicht zugänglich sein sollten, zu unterbinden, verwenden wir einen sogenannten Phaselistener. Wie jede JSF-Anwendung durchläuft auch unsere nach Erhalt einer Anfrage eine Reihe von Verarbeitungsphasen, bevor die Antwort an den Klienten fertiggestellt und verschickt wird. Ein Phaselistener kann während dieses Prozesses zusätzliche Integritätsbedingungen abprüfen und gegebenenfalls die Antwort auf die Anfrage verändern/verhindern. Zusätzlich wird durch das in JSF eingebaute Session-Tracking[1] sichergestellt, dass die verschiedenen Nutzerrollen eine unterschiedliche Version der gleichen Seiten angezeigt bekommen, deren Funktionalitäten den Berechtigungen der Rollen entsprechen.

[1] Das Verfolgen der Klientenverbindung zwischen den eigentlich verbindungslosen HTTP-Anfragen

Vorbeugen von Angriffen: Durch geschicktes Einschleusen von SQL[2]-Code in ungesicherte Formularfelder könnten Unbefugte direkten Zugriff auf die darunterliegende Datenbank erhalten. Der Datenbankzugriff in unserem System erfolgt via JDBC[3], welche bereits über einen Mechanismus zur Verhinderung von diesen SQL-Injections verfügt. Hierbei wird verhindert, dass von Nutzern eingegebener Text vom Managementsystem der Datenbank interpretiert wird. Ein weiterer Angriffsvektor stellt das sogenannte Cross-Site-Scripting dar. Dabei wird versucht, nutzergenerierte Teile von Websiten mit vom Browser interpretierbaren Codeabschnitten zu füllen, welche dann beim Anzeigen des Inhalts bei anderen Klienten ausgeführt werden. Es gibt verschiedene Versionen dieser Angriffsart, von denen nicht alle am Server verhindert werden können. Jedoch wird in der Anwendungslogik jede Nutzereingabe auf interpretierbaren HTML (und eingebundenen JavaScript) Code untersucht und gegebenenfalls entschärft.

5.2 Fehlerbehandlung und Logging

Von uns vorhergesehene Fehler und einige im Hintergrund ablaufende Prozesse werden durch einen selbst erstellten Logging-Mechanismus dokumentiert. Da dieser Mechanismus während des Entwicklungsprozesses ebenfalls nützlich ist und auf feinster Einstellung für den Anwender uninteressant sein könnte, wird es eine Option geben, die den Detailgrad und die Anzahl der ausgegebenen Meldungen festlegt. Alle Meldungen werden in eine seperate Log-Datei geschrieben und können wahlweise auf der Konsole angezeigt werden. Vorhergesehene Fehler in Abläufen werden von der Anwendung bestmöglich abgehandelt und wie erwähnt dokumentiert. Sollte jedoch ein nicht vorhergesehener Fehler auftreten, wird der Grund dafür in From der Java-eigenen Exceptions auf die Konsole ausgegeben. Durch umfangreiches Testen der Anwendung vor Auslieferung versuchen wir, Sie vor einem solchen Fall zu bewahren.

5.3 Selbstständige Prozesse

Im System gibt es verschiedene Fristen (Abholungsmarkierung, Ausleihrückgabe [global, pro Medium, pro Nutzer], Gültigkeit des Verifizierungs-/Zurücksetzungslinks) und damit verbundene Aktionen. Um diese Aktionen ausführen zu können, muss das System in gewissen Abständen selbstständig auf die Datenbank zugreifen. In der Klasse ???? ist dafür ein eigener Wartungsthread implementiert.

5.4 Starten und Stoppen der Anwendung

Start: Beim Systemstart wird zunächst der oben beschriebene Logging-Mechanismus initialisiert. Im Anschluss liest das System die Konfigurationsdatei ein und versucht die dort angegebene Verbindung zur Datenbank. Bei erfolgreicher Verbindung wird überprüft, ob die benötigten Tabellennamen alle existieren bzw. legt diese bei deren Fehlen an. Sollte die Struktur der Tabellen fehlerhaft sein, wird eine Fehlermeldung geloggt und die Tabellen werden nicht vom System benutzt. Wenn das System gerade hochfährt oder die Datenbank fehlerhaft/nicht verbunden ist, werden eingehende Anfragen mit einer Fehlerseite beantwortet. Nach erfolgreichem Ablauf der vorhin genannten Prozesse wird als nächstes der Wartungsthread angestoßen, um fristenabhängige Aktionen durchzuführen.

[2]Structured Query Language, eine weit verbreitete Datenbankanfragesprache
[3]Java Database Connectivity, eine universelle Datenbankschnittstelle

Stop: Beim planmäßigen Herunterfahren des Systems werden zunächst die noch offenen Datenbankverbindungen geschlossen und danach die Anwendung gestoppt, da dies sowohl sehr schnell passiert alsauch technisch schwierig ist, wird währenddessen auf eine extra Umleitung der Nutzer auf eine Fehlerseite verzichtet. Da SQL-Transaktionen dem ACID[4]-Prinzip folgen, wird die Datenbank auch bei Herunterfahren während eines Zugriffs in konsistentem Zustand hinterlassen. Wenn die Java Laufzeitumgebung des Systems plötzlich beendet wird und das System abstürzt, wird mittels einer sogenannten ShutdownHook trotzdem noch versucht, offene Datenbankverbindungen zu schließen. Dies ist bei einem Stromausfall allerdings unmöglich.

6 Datenfluss

Autor: Sergei Pravdin

7 ER-Modell

Autor: Jonas Picker

7.1 Legende

[4]Atomicity, Consistency, Isolation, Durability