UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4

Ein System zur sicheren Durchführung von Online Wahlen

JavaEE Web Applications 2014
Kursprojekt, Gruppe bravo

August 2014

Eingereicht von: Daniel Vivas Estevao (211 200 044)
Maximilian Strauch (211 201 869)

Eingereicht bei: Dr. Volker Riediger

Inhaltsverzeichnis

1 Installation

Das nachfolgende Kapitel beschreibt kurz wie Vote! zu Testzwecken lokal installiert wird. Es wird lediglich ein möglicher Installationsweg beschrieben.

1.1 Systemvoraussetzungen

Das vorliegende JavaEE–Projekt Vote! hat die folgenden *grundsätzlichen* Systemvoraussetzungen, um lauffähig zu sein:

- **GlassFish Server Open Source Edition** in der Version 4.0 (Build 89)
  - mit **Mojarra 2.2.2 Server Faces**, siehe auch 2.5.1 Darstellungsprobleme
- **Apache Derby** als Datenbankserver in der Version 10.9.1.0
- **Netbeans IDE** 8.0 zum einfachen V*erteilen* des Projekts oder alternativ über das Kommandozeilenprogramm `asadmin` des GlassFish Servers (hier nicht beschrieben)
- Einen modernen Browser; **Firefox** 29 oder neuer.

1.2 Installation

Das nachfolgende Unterkapitel beschreibt die Installation des Projektes Vote! auf einem Computer zu Testzwecken unter Verwendung eines lokalen GlassFish Servers sowie einem Apache Derby Datenbankserver. Dabei wird das Verteilen sowie Starten und Herunterfahren des GlassFish– und Datenbankservers über die Netbeans IDE vorgenommen.

Schritt 1: Einrichten der Doppelauthentifizierung

Damit sich später Benutzer am Vote! System anmelden können, muss dies zunächst im GlassFish Server konfiguriert werden. Dazu wird ein eigens für dieses Projekt angefertigter Realm verwendet. Dieses Stück Software erledigt dann die Überprüfung von Benutzernamen und Passwort und gewährt einem Nutzer gegebenenfalls Einlass. Da das Vote! System eine Art Doppelauthentifizierung benötigt, siehe auch 2.3.1 Der ComboRealm, wurde ein eigener Realm geschrieben, der zunächst in den GlassFish Server integriert werden muss, bevor man ihn verwenden kann. Die Installation läuft folgendermaßen ab:

1. Stellen Sie sicher, dass der GlassFish Server herunterfahren ist oder fahren Sie diesen über Netbeans im Reiter „Services“ unter „Servers > GlassFish Server 4“ über das Kontextmenü herunter.

2. Navigieren Sie nun mit einem Dateiexplorer in das Verzeichnis des GlassFish Servers. Von dort aus navigieren Sie in das Verzeichnis `glassfish/domains/<IhreDomain>/lib` wobei `<IhreDomain>` durch den Namen Ihrer Domain ersetzt werden muss. I.d.R. sollte hier jedoch nur die Wahloption `domain1` zur Verfügung stehen.
3. Platzieren Sie nun in diesem Verzeichnis die Datei `combo-realm.jar`.
4. Wechseln Sie jetzt – relativ zum Basisverzeichnis des GlassFish Servers wie in Schritt 2 – in das Verzeichnis `glassfish/domains/<IhreDomain>/config` und öffnen Sie dort, mit einem Texteditor Ihrer Wahl, die Datei `login.conf`.
5. Platzieren Sie nun am Ende der Datei das nachfolgende Stückchen Quelltext. Dieses sorgt später dafür, dass das zuständige Login–Modul mit der Doppelauthentifizierung von Vote! gefunden und verwendet werden kann:

```
ComboLoginModule {
      de.combo.auth.ComboModule required;
};
```

6. Nun starten Sie den GlassFish Server und rufen die Administrationskonsole über die Adresse *http://localhost:4848/* auf.
7. In dem Menübaum auf der linken Seite navigieren Sie zu der Seite „server-config > Sicherheit > Realms“ und klicken dort auf „Neu“, um einen neuen Realm für das Projekt anzulegen.
8. Fügen Sie als Namen bitte den Namen „**VoteRealm**“ ein und wählen Sie bei Klassenname die zweite Auswahlmöglichkeit und geben eine benutzerdefinierte Klasse an: in das aktivierte Textfeld fügen Sie bitte den vollen Klassennamen „**de.combo.auth.ComboRealm**“ ein.
9. Legen Sie nun vier weitere Eigenschaften mit den hier angegeben Werten an:

Name	Wert
`jaas-context`	`ComboLoginModule` *(der Name des Login–Moduls aus der login.conf)*
`jndi-auth-bean`	`java:global/vote/vote-ejb/VoteAuthenticator!de.vote.secturity.VoteAuthenticator` *Achtung: der String darf kein Leerzeichen enthalten!*
`fallback-realm`	*Name des „Fallback Realms“. Lesen Sie hierzu auch 2.3.1 Der ComboRealm.*
`fallback-group`	`organizer`

10. Nachdem Sie alle Werte eingetragen haben, können Sie mit einem Klick auf „Speichern“ die Werte abspeichern.
11. Damit alle Änderungen übernommen werden, sollten Sie jetzt den Server herunterfahren und erneut starten. Während des Startens können Sie auf der Konsole von NetBeans die Ausgaben verfolgen und werden – bei erfolgreicher Installation – folgende „neuen“ Zeilen sehen:

```
Information:   Creating realm "de.combo.auth.ComboRealm"...
Information:   fallback-group = organizer
Information:   jndi-auth-bean = java:global/vote/vote-ejb/Vote
             Authenticator!de.vote.secturity.VoteAuthenticator
Information:   fallback-realm = file
```

```
Information:   jaas-context = ComboLoginModule
Information:   Using fallback realm: file
Information:   Using fallback group: [organizer]
Information:   SEC1115: Realm [VoteRealm] of classtype
[de.combo.auth.ComboRealm] successfully created.
```

Diese Ausgabe zeigt an, dass der ComboRealm erfolgreich initialisiert wurde. Möglicherweise weichen bei Ihnen die Angaben zum „Fallback Realm“ ab – je nach dem was Sie konfiguriert haben.

Schritt 2: JavaMail-Session einrichten

Wir verbleiben noch ein wenig in der Administrationskonsole, um die JavaMail-Session einzurichten, damit Vote! auch E-Mails versenden kann. Da diese Installationsanleitung nur eine Testinstallation beschreibt, wird im Folgenden die Einrichtung eines Testmailservers beschrieben. Natürlich können Sie hier einen realen Mailserver konfigurieren – es ergibt sich kein Unterschied. Nur der Ressourcenname muss gleich bleiben.

1. Navigieren Sie in der Hauptnavigation auf der rechten Seite zum Punkt „Ressourcen > JavaMail-Sessions“ und klicken dort wieder auf „Neu“ um eine neue Mail-Session anzulegen.
2. Als JNDI-Namen vergeben Sie bitte „**VoteMail**“. Dieser Name ist die Brücke, die es Vote! erlaubt, durch die nachfolgenden Konfigurationsdaten auf den Mailserver zuzugreifen. Fügen Sie weiter folgende Werte in die Textfelder:

Name des Textfelds	Wert
Mailhost:	`localhost`
Standardbenutzer:	`nobody`
Standardmäßige Absenderadresse:	`nobody@localhost`
Speicherprotokoll:	`POP3`
Speicherprotokollklasse:	`com.sun.mail.pop3.POP3Store`
Transportprotokoll:	`SMTP`
Transportprotokollklasse:	`com.sun.mail.smtp.SMTPTransport`

Unter „Weitere Eigenschaften“ fügen Sie drei Eigenschaften mit den folgenden Namen und Werten hinzu:

Name	Wert
`mail.smtp.auth`	`false`
`mail.smtp.port`	`2500`
`mail.smtp.host`	`localhost`

3. Klicken Sie schließlich auf „Speichern“, um die JavaMail-Session anzulegen.
4. Starten Sie das Java-Programm `TestSmtpServer.jar` aus dem Projektverzeichnis. Hierbei handelt es sich um den Testmailserver. Dieser sollte, während Sie Vote! testen, immer laufen, da ansonsten bei einigen Aktionen Fehler verursacht werden, da der nicht vorhandene Mailserver nicht auf Anfragen des Vote! Systems antwortet.

Schritt 3: Datenbankpool und -connection einrichten

Damit das Vote! System richtig arbeiten kann, benötigt es auch eine Datenbank. Dazu wird die Apache Derby Datenbank verwendet. Diese kommt mit NetBeans und muss im Folgenden erst noch für die Verwendung konfiguriert werden:

1. Wechseln Sie in NetBeans zu dem Reiter „Services“ und erweitern Sie „Databases“.
2. Starten Sie über einen Rechtsklick im Kontextmenü des Punktes „Java DB“ die Datenbank.
3. Wählen Sie dann die Option „Create Database...“ im selben Kontextmenü und legen Sie eine neue Datenbank an. Merken Sie sich dabei alle vergebenen Werte.
4. Wechseln Sie nun wieder zur Administrationskonsole des GlassFish Servers und navigieren Sie in der Hauptnavigation (links) zu dem Punkt „Ressourcen > JDBC > JDBC-Connection Pools“ und wählen Sie dort die Option „Neu“.
5. Geben Sie als Poolnamen „**VotePool“** ein und wählen Sie als Ressourcentyp „**javax.sql.DataSource**“ aus. Der Datenbankhersteller ist „**Derby**“. Bestätigen Sie mit einem Klick auf „Weiter“.
6. Die meisten Einstellungen sind nun automatisch vergeben worden und in Ordnung. Sie müssen lediglich die weiteren Eigenschaften anpassen. Löschen Sie dazu alle vorhandenen Eigenschaften und fügen Sie die folgenden Eigenschaften ein:

Name	Wert
`URL`	`jdbc:derby://localhost:1527/`*<Benutzername aus Schritt 3>*
`User`	*Benutzername aus Schritt 3.*
`DatabaseName`	*Datenbankname aus Schritt 3.*
`Password`	*Passwort aus Schritt 3.*
`ServerName`	`localhost`
`PortNumber`	`1527`

7. Speichern Sie nun den Pool und begeben Sie sich zu dem Menüpunk „JDBC-Ressourcen“. Klicken Sie dort auf „Neu“, um eine neue Ressource anzulegen.
8. Für die neue JDBC-Ressource vergeben Sie nun den JNDI-Namen „**jdbc/vote**“ und wählen bei Poolname den gerade erzeugten Pool, „**VotePool**“ aus. Speichern Sie nun die neu angelegte Ressource. Hiermit ist das Anlegen der Datenbankverbindung abgeschlossen.

Schritt 4: Vote! auf den GlassFish Server bringen

Der finale Schritt besteht nun darin, das Projekt auf den GlassFish Server zu verteilen. Dazu wird die NetBeans IDE verwendet. Legen Sie sich auch die Projektdateien in einem Verzeichnis bereit:

1. Starten Sie die NetBeans IDE, sofern diese bisher noch nicht gestartet wurde.
2. Klicken Sie auf „File > Open Projekt ...“ oder „Datei > Projekt öffnen ...“ und wählen Sie das Projekt*verzeichnis* aus. NetBeans sollte automatisch ein Projekt erkennen und Ihnen dies bereits im „Projekt öffnen“–Dialog visuell anzeigen.

3. Ist das Projekt von NetBeans geöffnet und erstmalig durchsucht worden, können Sie mit einem Rechtsklick auf „vote“ die Option „Deploy“ auswählen. Nun wird das Projekt auf den GlassFish Server verteilt.
4. Sobald NetBeans mit dieser Aufgabe fertig ist, sollte sich im Browser die Startseite von Vote! öffnen und Sie anlachen. Falls nicht, haben Sie möglicherweise diese Funktion abgestellt und müssen manuell die folgende Adresse eingeben: *http://localhost:8080/vote-war/*. Das System wird Sie aber aus Sicherheitsgründen sofort auf eine sichere Verbindung umleiten, die über SSL abgesichert ist: *https://localhost:8181/vote-war/*. **Achtung: Da es sich hier um eine Testinstallation handelt und keine echten Zertifikate verwendet werden, wird Ihr Browser Sie warnen, dass die Seite nicht sicher sei. Diese Warnung müssen Sie übergehen.**

*Hiermit ist die Installation von Vote! abgeschlossen und das System kann nun verwendet oder getestet werden. Sie können sich standardmäßig als Administrator anmelden mit dem Benutzernamen „**admin**“ und dem Passwort „**admin**“. Dieser Benutzer wurde automatisch angelegt und es ist ratsam das Kennwort recht bald zu ändern.*

2 Datenmodell und Anforderungen

In diesem Kapitel wird kurz auf das verwendete Datenmodell eingegangen sowie einige Systemgrundlagen – also Anforderungen und die Struktur des erarbeiteten Projektes – näher erläutert, um einen kurzen Überblick über das System zu geben.

2.1 Das Datenmodell

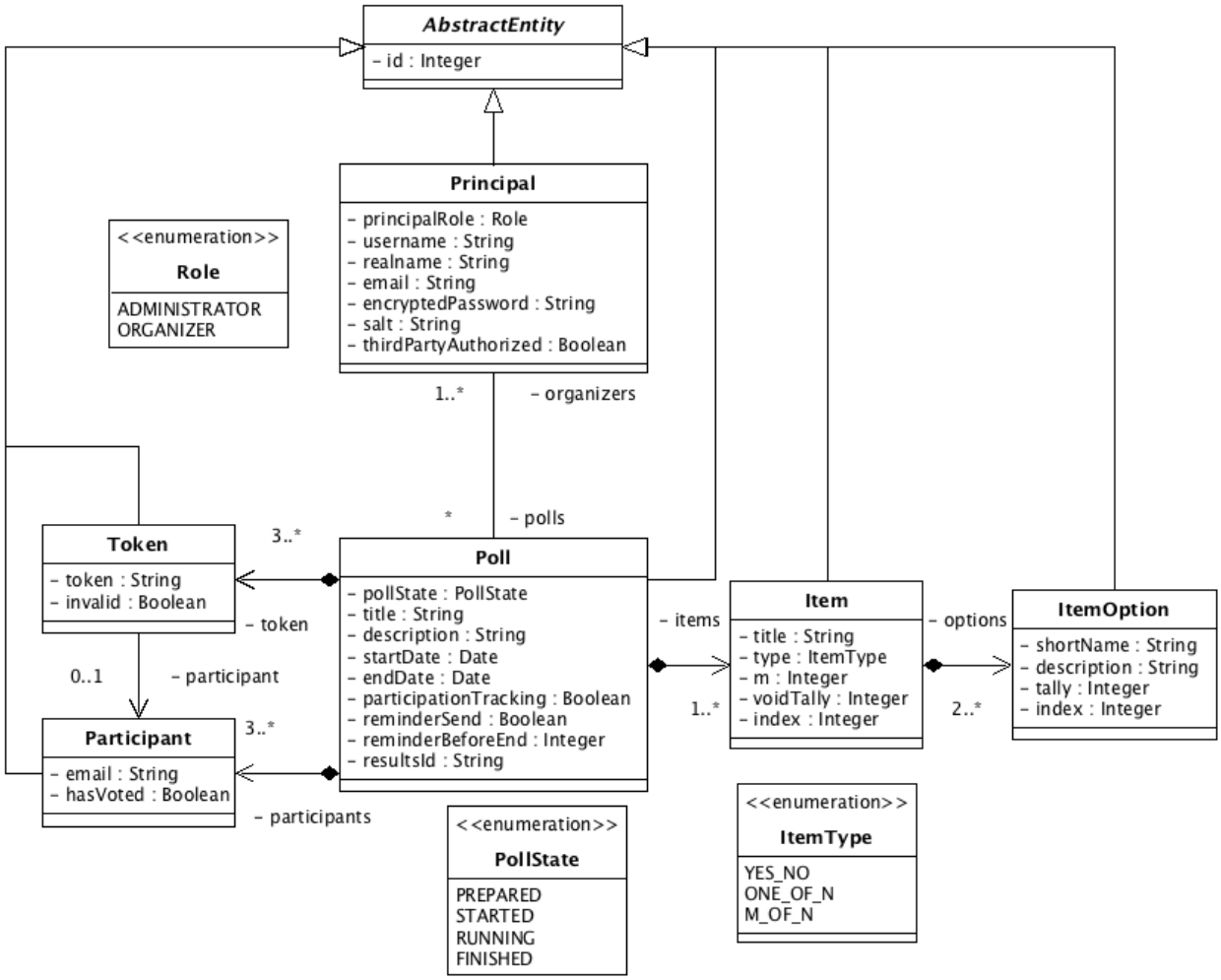

Abbildung 1: Datenmodell aller Entity-Klassen des Systems Vote!.

Abbildung 1 zeigt das Datenmodell, das dem Vote! System zugrunde liegt und verwendet wird, um Daten über JPA unter Verwendung von EclipseLink in der Apache Derby Datenbank zu speichern.

Folgende „Besonderheiten“ bzw. Änderungen wurden eingebaut, ausgehend von dem initialen

Vorschlag, der in den Übungen erarbeitet wurde:

- **Speicherung der Ergebnisse in den Klassen Item und ItemOption.** Da die „Ergebnisse“ von einzelnen Teilnahmen nur Striche einer Strichliste sind, genügt in der Klasse ItemOption ein einfacher Zähler („tally“), der zählt, wie oft diese Option gewählt wurde. Daneben gibt es noch in der Klasse Item einen zweiten Zähler („voidTally“), der die ungültigen Stimmen für ein Item zählt.

  Obgleich während der Übungen der Gedanke im Raum stand, die Ergebnisdaten in eine zweite Datenbank auszulagern, wurde die Entscheidung des obigen Vorgehens bewusst getroffen. Ausgehend von der Systemannahme, dass der Server, auf dem das Vote! System läuft, sicher ist, kann nicht zurückverfolgt werden, welche Person eine Stimme abgegeben hat, da lediglich ein Zähler inkrementiert wird und keine weiteren Daten gespeichert werden. Einzig der Fall, dass bisher nur ein Teilnehmer teilgenommen hat, erlaubt eine Zurückverfolgung der abgegebenen Stimme. Allerdings tritt dieses Problem auch bei obiger Lösung mit einer Doppeldatenbank auf. Daher ist der Aufwand für die Verwaltung zweier Datenbanken überflüssig und kann durch diese einfache Lösung ersetzt werden.

- **Principal als allgemeiner Verwalter.** Die Klasse Principal beherbergt sowohl Administratoren als auch Organisatoren. Die Unterscheidung wird über das Attribut „principalRole“ getroffen. Natürlich hätte man hier auch eine Klasse mit zwei Spezialisierungen verwenden können. Allerdings ist diese Lösung kürzer, da es sich bei den Entity–Klassen nur um Datenkapseln handelt und Vererbungsbeziehungen eher verwendet werden, wenn es darum geht, Verhalten, d.h. Methoden, zu vererben oder zu verändern (überschreiben).

- **AbstractEntity als Kapsel für allgemeine Methoden.** Die abstrakte Klasse wird von allen Entity–Klassen implementiert und stellt die ID sowie die Methoden `equals()` und `hashCode()` zur Verfügung, da diese für alle Entity–Klassen notwendig sind.

- **Beziehung zwischen Token und Participant.** Die Beziehung zwischen Token und Participant ist nur gesetzt, falls das sogenannte „participation tracking“ – also die Teilnehmerrückverfolgung – aktiviert ist. Sobald ein Token verwendet wurde, wird diese Beziehung gelöscht. Sie dient dazu, die Erinnerungsmail an all die Teilnehmer zu senden, die noch nicht ihr Token verwendet haben. Dazu werden alle Token betrachtet, die noch gültig sind und die E-Mail-Adresse des Empfängers wird über diese Beziehung bestimmt.

- **Weitere Einschränkung.** Das Klassendiagramm ist mit weiteren Einschränkungen beschriftet (einige Multiplizitäten), wie beispielsweise der Einschränkung für mindestens drei Teilnehmer an einer Wahl. Diese Einschränkungen wurden in der Business–Schicht der Implementierung umgesetzt.

- **Kleinere Umbenennungen.** Bei den Attributen und Klassennamen wurden kleinere Umbenennungen vorgenommen im Vergleich zu dem initialen Modell aus der Übung. Dies ergab sich aus Kollisionen mit Schlüsselwörtern aus dem SQL99-Standard.

2.2 Umgesetzte Anforderungen

Aus der Anforderungsliste wurden **alle** Anforderungen bis auf die Anforderungen 12.7 und 12.8 umgesetzt. Letztere beiden Anforderungen bieten Freitextfelder als Wahloptionen für Frage an. Diese Anforderungen haben einen größeren Implementationsaufwand zur Folge und bieten keine automatisierte Ergebnisdarstellung an. Außerdem werden hierdurch die Anforderungen 8.1 bis

8.3 aufgeweicht, da die Freitexteingaben nicht mehr vollständig anonym sind. Daher wurden diese beiden Anforderungen ausgelassen und an anderer Stelle die Anforderungen erweitert, je nachdem, wo es sinnvoll erschien. Siehe dazu auch 2.3 Features, die über die Anforderungen hinaus gehen.

Die Systemvision wurde auch **vollständig** bis auf den zweiten Satz im vierten Abschnitt erfüllt. In diesem geht es um spezielles Verhalten im Bezug auf ungültige Wahlen für verschiedene Wahlmodi (absolute Mehrheit, relative Mehrheit, einfache Mehrheit). Um Vote! generisch zu halten, wurde diese Option ausgelassen und dem Benutzer überlassen, wie das Ergebnis interpretiert werden soll. Dies erlaubt maximale Flexibilität.

2.3 Features, die über die Anforderungen hinaus gehen

Folgende Anwendungsfälle oder Features gehen über die in der Anforderung genannten hinaus und wurden zusätzlich implementiert, um das Vote! System komfortabler zu machen:

- ✔ Implementation einer systemeigenen Authentifizierung und einer LDAP Authentifizierung (bzw. geht die Authentifizierung darüber hinaus). Siehe 2.3.1 Der ComboRealm (Anforderungen 3.2, 3.2.1, 3.2.2).
- ✔ Passwort-Ändern-Funktion für alle systemeigenen Benutzer. Da Benutzer möglicherweise ihr Passwort ändern möchten, wurde eine Funktion hinzugefügt, die es allen systemeigenen Benutzern erlaubt ihr Passwort zu ändern (Anforderung 4.4).
- ✔ Der Administrator hat keine Kontrolle über die Benutzerpasswörter; diese werden automatisch per Mail von dem Vote! System an die angegebene E-Mail-Adresse versendet. So kann der Administrator nicht die Identität eines Nutzers annehmen, um die Wahlergebnisse einer Wahl einzusehen. Siehe auch 2.4 Weitere Sicherheitsaspekte (Anforderung 4.4).
- ✔ Benutzer deaktivieren oder aktivieren. Es ist dem Administrator möglich, Benutzer zu deaktivieren (können sich nicht mehr anmelden) und wieder zu aktivieren. In letzterem Fall wird dem Benutzer eine Aktivierungs-E-Mail mit Benutzername und neuem Passwort zugesandt. Auch hier hat der Administrator keinen Zugriff auf das Passwort und kann somit nicht die Identität des Benutzers annehmen, um Anforderung 4.3 auszuhebeln.
- ✔ Ergebnisdarstellung in grafischer Form. Die Ergebnisse einer Wahl können in grafischer Form (Balkendiagramm) betrachtet werden (Anforderung 13.6).
- ✔ Wenn die Ergebnisse einer Wahl veröffentlicht wurden und ein Link bereit steht, kann dieser von dem Organisator veröffentlicht werden. Da es wahrscheinlich ist, dass der Organisator allen Teilnehmern diesen Link zukommen lassen möchte, ist diese Funktion bereits in Vote! integriert. Mit einem Klick auf das Brief-Symbol in der Liste aller geschlossenen Wahlen wird eine Automatische E-Mail mit dem Link generiert und an alle Teilnehmer gesendet.
- ✔ Keine automatische Vervollständigung des Tokenfeldes einer Wahl. Siehe auch 2.4 Weitere Sicherheitsaspekte.

2.3.1 Der ComboRealm

Anforderung 3.2 fordert, dass das Vote! System einen geschützten Login–Bereich für Organisa-

toren und Administratoren haben soll. Die Anforderungen 3.2.1 und 3.2.2 spezifizieren, dass entweder eine Authentifizierung über LDAP oder eine systemeigene Authentifizierung vorgenommen werden muss.

Der Vorteil einer systemeigenen Authentifizierung liegt darin, dass kein LDAP Server existieren muss, damit Vote! verwendet werden kann. Außerdem können so die Benutzer flexibler verwaltet werden und es ist möglich bei der (Weiter-)Entwicklung des Systems auch außerhalb der Reichweite eines LDAP Servers zu arbeiten. Hingegen ist es der Vorteil eines LDAP Servers, dass sich Nutzer sehr schnell authentifizieren können und das Vote! System verwenden können, ohne einen Administrator zu konsultieren, damit dieser einen Benutzeraccount für Sie anlegt.

Um die oben genannten Vorteile beider Verfahren auszunutzen und für das Vote! System bereitzustellen, wurde ein eigener Realm, also ein Modul für den GlassFish Server, das die Authentifizierung von Benutzern übernimmt, geschrieben – der „ComboRealm“.

Die Funktionsweise dieses Realms ist relativ einfach und kann – wenn sich ein Benutzer anmelden möchte – in zwei Schritte aufgeteilt werden, die nacheinander abgearbeitet werden. Nachdem Benutzername und Passwort abgeschickt wurden, passiert Folgendes:

1. Der ComboRealm versucht über den vorkonfigurierten JNDI Pfad (Eigenschaft `jndi-auth-bean`) eine Bean zu kontaktieren, die das `de.combo.auth.ComboAuthInterface` Interface implementiert. Dieses Interface spezifiziert unter anderem die Methode `authenticateUser()`. Diese wird nun mit Benutzernamen und Passwort zur systemeigenen Authentifizierung des Benutzers aufgerufen. Im Vote! System ist die Implementierung – wie oben angegeben – die Klasse `de.vote.security.VoteAuthenticator`. Dort wird versucht, in der Datenbank ein Principal Entity zu finden, das zu Benutzername und Passwort passt. Wird ein passender Principal gefunden, aber passt das Passwort nicht, wird (nur) eine `LoginException` geworfen. Hiermit weiß der ComboRealm, dass er keine weiteren Bemühungen vornehmen muss und einen Loginfehler verursachen kann, da der Benutzer ein falsches Passwort eingegeben hat. Wenn allerdings eine `IllegalArgumentException` geworfen wird, signalisiert das Vote! System dem ComboRealm, dass kein passender Benutzer zu den Daten existiert.
2. Da Schritt 1 ergeben hat, dass kein Benutzer mit den angegebenen Daten im Vote! System vorhanden ist, wird nun der Benutzer über einen sogenannten „Fallback Realm“ authentifiziert. Dabei verwendet der ComboRealm den in der Konfiguration unter `fallback-realm` angegebenen Realm des GlassFish Servers. Hier muss der Name eines beliebigen anderen Realms angegeben werden. Es kann so also ein LDAP Realm hier verknüpft werden, aber auch ein anderer Realm ist möglich. Der ComboRealm übergibt dann die Benutzerdaten an diesen spezifizierten Realm und es wird versucht, den Benutzer anzumelden. Bei Misserfolg gibt es eine Loginfehlermeldung und bei Erfolg wird der Benutzer über diesen „Drittanbieter“ authentifiziert. Die aktuelle Authentifizierungsart wird auch im Vote! System in der Fußzeile angezeigt. Damit auch die richtige Benutzergruppe gesetzt wird, wird diese im Konfigurationsfeld `fallback-group` festgelegt und an dieser Stelle einfach verwendet. Damit das Vote! System Kenntnis davon erlangt, dass ein Principal von dritter Stelle authentifiziert wurde, wird noch die Methode `fallbackLoginPerformed()` des Interfaces `ComboAuthInterface` aufgerufen mit Benutzername und Gruppen. Hier kann das Vote! System dann eine Principal Entity für diesen Benutzer anlegen.

Der entwickelte ComboRealm bietet somit maximale Flexibilität um einerseits systemeigene

Benutzer zu verwalten und zu authentifizieren und andererseits die Anmeldung von Benutzern über beispielsweise einen separaten LDAP Realm zu ermöglichen. Die Konfiguration eines anderen Realms erlaubt somit maximale Freiheit, um anstatt eines LDAP Realms auch einen beliebigen anderen Realm des GlassFish Servers zu verwenden.

2.4 Weitere Sicherheitsaspekte

- ✔ **HTTPS–Verbindungen.** Das Vote! System verwendet durchgängig sichere HTTP Verbindungen. Der `de.vote.web.filter.HttpToHttpsFilter` Filter prüft bei jeder Seitenanfrage, ob es sich um eine sichere Verbindung handelt. Falls dies nicht der Fall ist, wird der Nutzer zunächst auf eine sichere Verbindung weitergeleitet, bevor er weiterarbeiten kann. *Achtung: gerade im Testbetrieb werden keine „korrekt" ausgestellten Zertifikate verwendet. Dies munieren die Browser und man muss der Verbindung erst zustimmen. Trotzdem ist die Verbindung sicher.*
- ✔ **UUID als zufällige Zeichenketten.** Wann immer im Vote! System eine zufällige Zeichenkette benötigt wird, wird auf die Java Methode `java.util.UUID.randomUUID()` zurückgegriffen. Diese erzeugt einen „Universally unique identifier", wie in RFC4122[1] oder in ISO-/IEC 11578:1996[2] beschrieben. Eine UUID ist eine 16 Byte (128 Bit) Zahl, die aus vielen Informationen besteht und in der Informatik an vielen Stellen zur eindeutigen Kennzeichnung verwendet wird. Die Wahrscheinlichkeit zwei gleiche UUIDs zu erhalten, wenn man beispielsweise 2^{36} = 68.719.476.736 UUIDs generiert, liegt bei 0,00000000000004 %[3]. Dies sollte ausreichend sicher für die Anwendungsfälle des Systems Vote! sein. Zu erwähnen sei auch noch, dass eine UUID als Zeichenkette aus 32 (36) zufälligen Zeichen besteht und somit keine Gefahr besteht, dass jemand eine UUID fälscht oder durch Ausprobieren herausfindet (in kurzer Zeit).
- ✔ **Verschlüsselte Passwörter in der Datenbank.** Die Systemeigene Authentifizierung verwendet Passwörter zur Nutzeridentifizierung. Um höchste Sicherheit zu gewährleisten werden die Passwörter mit dem Hashingverfahren SHA-1 gehashed und in der Datenbank gesichert, so dass keine Klartextpasswörter in der Datenbank zu finden sind. Da jedoch zwei gleiche Passwörter, aufgrund der Kollisionsresistenz der verwendeten Hash Funktion, den gleichen Hashwert ergeben, wird vor dem Hashing des Passworts noch ein sogenannter „salt" („Körnchen Salz") angehangen. Dies ist wieder eine 36 Zeichen lange, zufällig erzeugte UUID. Somit sind die sich ergebenden Hashwerte für zwei gleiche Passwörter unterschiedlich.
- ✔ **Administrator kann nicht Alles.** Die Rolle Administrator im Vote! System kann dieses zwar verwalten, kann jedoch nicht alles. Folgende Einschränkungen gelten:
    1. Der Administrator kann alle Benutzer bearbeiten und löschen.
    2. Der Administrator kann für keinen Benutzer ein Passwort vergeben, damit er nicht seine Identität annehmen kann, um Anforderung 4.3 auszuhebeln und sich die Ergebnisse einer Wahl anzusehen kann. Das Vote! System versendet bei der Erzeugung oder Reaktivierung eines Benutzers eine E-Mail an die hinterlegte E-Mail-Adresse mit einem neu generierten Passwort.

1 http://tools.ietf.org/html/rfc4122
2 http://www.iso.org/iso/catalogue_detail.htm?csnumber=2229
3 http://en.wikipedia.org/wiki/Universally_unique_identifier#Random_UUID_probability_of_duplicates

3. Der Administrator kann nur Wahlen und deren Ergebnisse einsehen, wenn er Organisator dieser Wahl ist.
4. Der Administrator kann alle Wahlen löschen, aber nicht einsehen.

- ✔ **Passwortänderungen werden per E-Mail kommuniziert.** Änderungen am eigenen Benutzerpasswort sind möglich und werden durch eine E-Mail vom Vote! System quittiert. Dies soll die Benutzer in dem Fall informieren, falls sie nicht selbst das Passwort geändert haben.
- ✔ **Ergebnisse einer Wahl nur über einen Link mit Zufallswert erreichbar.** Sind die Ergebnisse einer Wahl vom Organisator veröffentlicht, so können diese über einen Link erreicht werden. Um unerlaubten Zugriff zu vermeiden, ist in dem Link eine UUID eingebaut. Diese bietet genügend Zufall, um ein „blindes" Erraten und somit den unerlaubten Zugriff auf ein Ergebnis zu vermeiden.
- ✔ **Zurück–Button ohne Funktion bei den Wahlen.** Die Nichtverwendung des Zurück-Buttons nach einer Wahl ist wichtig, um die Geheimhaltung der Wahl eines Nutzers zu gewährleisten. Zunächst ist hierzu jedoch zu sagen, dass es keine 100%-ige Lösung gibt und es immer das Beste ist, wenn der Nutzer das Browserfenster schließt und somit den Cookie mit der Session ID vernichtet und so ein Zugriff auf diese Session verhindert wird. Dies wird dem Benutzer auch in einem textuellen Hinweis nach der Wahl mitgeteilt. Darüber hinaus sind jedoch noch weitere Mechanismen in das Vote! System eingebaut, die verhindern sollen, dass der Zurück–Button verwendet werden kann, wenn der Browser nicht geschlossen wurde. Diese sind:
  - Der `de.vote.web.filter.NoCacheFilter` ist dafür zuständig, an den Browser die zusätzlichen HTTP-Header „`Cache-Control`", „`Pragma`" und „`Expires`" zu senden, falls dieser die Wahlseiten (`index.xhtml`, `vote.xhtml`, `thankyou.xhtml`) anfordert. Diese HTTP-Header werden mit Werten befüllt, die dem Browser ein Caching der Seite verbieten sollen.
  - Darüber hinaus sind obige HTTP-Header auch nochmals in dem Template für Seiten des Frontends (`WEB-INF/templates/frontend.xhtml`) als HTML-meta-Tag eingefügt.
  - Browser wie beispielsweise der Firefox erlauben trotzdem noch die Benutzung des Zurück–Buttons. Nach längerer Internet-Recherche ist herausgekommen, dass man den body-Tag mit dem onunload-Event erweitern muss: `<h:body onunload="">`. Macht man dies, lassen es die meisten Browser nicht mehr zu, dass man über einen Klick auf den Zurück–Button auf die vorherige Seite mit ausgefülltem Formular kommt, sondern lädt die Seite neu vom Server. An dieser Stelle schlagen dann die Filter zu und leiten den Benutzer auf die Startseite weiter.
  - Workarounds, die auf der Javascript-Funktion „`window.history.forward()`" basieren, haben sich im empirischen Test als nicht funktionsfähig herausgestellt und wurden daher nicht weiter beachtet.
- ✔ **Keine Autovervollständigung des Token-Feldes.** Mit etwas größerem Aufwand konnte dafür gesorgt werden, dass das Token-Feld auf der Startseite nicht autovervollständigt wird mit vorherigen Tokens oder gar auf öffentlichen Computern mit fremden Tokens. Es liegt folgende Idee zugrunde: das Token-Feld hat einen Namen mit einem zufälligen Wert. Da

der Browser die Autovervollständigung an dem Namen des Eingabefeldes festmacht, sorgt ein zufälliger Wert, der bei jedem Neuladen der Seite neu erzeugt wird, dafür, dass das Feld immer für den Browser „unbekannt“ ist und somit keine Autovervollständigung angezeigt werden kann, weil keine Werte verfügbar sind. Der `de.vote.web.filter.JumpToIndex-Filter` sorgt dann dafür, dass im Falle der `index.xhtml` Seite die Parameter-Map des `HttpServletRequest` nach eben diesem zufälligen Namen durchsucht wird. Wurde ein Wert gefunden, wird dieser einfach der entsprechenden Backing-Bean zugewiesen. Da die Filter sehr früh ausgeführt werden, kann die Operation in der Backing-Bean bereits mit dem Wert arbeiten. Zusätzlich wurden noch die HTML-Attribute „`autocorrect="off" autocapitalize="off" autocomplete="off"`“ in das Textfeld eingefügt. Diese verbieten zwar eine Autovervollständigung, verhindern diese aber nicht immer.

- ✔ **Direktlink zu einer Wahl aus der E-Mail.** Ein Direktlink für eine Wahl, der das Token des Benutzers enthält und das Token-Feld ausfüllt, ist problematisch, weil eben die URL bereits das Wählertoken enthält. Die URL wird dabei vom Browser als Historie gesichert. Auf einem öffentlichen Computer könnte beispielsweise ein Angreifer an Wählernummern kommen (z.B. im Falle eines Abbruchs), die noch nicht eingelöst wurden und so diese missbrauchen. Dieses Problem kann man jedoch nicht aus Serversicht umgehen. Daher bekommt der Benutzer in der E-Mail einen Hinweis auf die mögliche Gefahr, wenn er dem Link folgt. Außerdem wird dies auch nach der Wahl nochmals erwähnt und der Benutzer wird aufgefordert, seine Historie zu löschen.

2.5 Nicht-funktionale Umsetzung

Die Umsetzung der Präsentationsschicht (die Benutzeroberfläche) wurde mit Hilfe von CSS (3), HTML und ein wenig JavaScript im Backend gelöst. Es ist zu bemerken, dass keine fertigen Bibliotheken verwendet wurden, sondern die Präsentation selbst entwickelt und umgesetzt wurde. Dies gilt vor allem für die grafische Gestaltung mittels CSS.

Schließlich noch ein Wort zu der Browserkompatibilität: das System wurde mit Firefox (29) auf Windows und Mac OS entwickelt und ist dementsprechend zu diesem Browser maximal kompatibel. Es sollte jedoch kein Problem darstellen, wenn andere *moderne* Browser verwendet werden. Für den Gebrauch mit Tablet–Computern (z.B. iPad oder Android) ist das Vote! System auch geeignet. Getestet wurde diese Funktion mit dem NVIDIA Tegra Note 7 sowie dem Archos Internet Tablet 10.1 – also auf einem 7" und 10.1" Bildschirm. Für kleinere Auflösungen ist zumindest das Frontend konzipiert (kein festes Layout), konnte jedoch aufgrund mangelnder Hardware nicht getestet werden. Folgende Browser wurden getestet:

- *Windows:* Mozilla Firefox 31 (**erfolgreich**), Internet Explorer 9 (**erfolgreich**)
- *Mac OS:* Mozilla Firefox 29 (**erfolgreich**), Safari 7.0.3 (**erfolgreich**), Google Chrome 36 (**erfolgreich**)
- *Linux (Lubutu 14.0):* Mozilla Firefox 31 **(erfolgreich)**
- *Android 4.4 KitKat:* Google Chrome 35 **(erfolgreich)**
- *Android 2.2 Froyo:* Systembrowser **(mäßig, einige Darstellungsprobleme vorhanden)**

2.5.1 Darstellungsprobleme

Wenn Sie nicht die aktuelle Version der **Mojarra Server Faces (2.2.2)** Frameworks verwenden, kann es sein, dass Elemente wie `<ui:repeat>` mit in die generierte HTML Seite gerendert werden. Die ist ein bekanntes Problem und wird von Version 2.2.2 behoben. Es kommt hierbei zu Problemen, wenn der Browser nicht tolerant ist und somit XML Tags entdeckt, die nicht in das HTML hineingehören. Um das Problem zu lösen, ist wie folgt vorzugehen:

1. Laden Sie sich die aktuellste Version herunter[4].
2. Benennen Sie diese Datei um in `javax.faces.jar`.
3. Navigieren Sie in das Basisverzeichnis des GlassFish Servers und gehen Sie dann in das Verzeichnis `glassfish/modules`. Dort sichern Sie die existierende Datei `javax.faces.jar` und ersetzen Sie die durch die heruntergeladene Datei.
4. Starten Sie nun den GlassFish Server neu. Jetzt sollte Mojarra 2.2.2 verwendet werden.

4 https://maven.java.net/content/repositories/releases/org/glassfish/javax.faces/2.2.2/javax.faces-2.2.2.jar