Projekt SpamGourmet

Projekt SpamGourmet

Inhaltsverzeichnis

Tabellenverzeichnis

Teil I. Project

Inhaltsverzeichnis

Spamgourmet Dokumentation

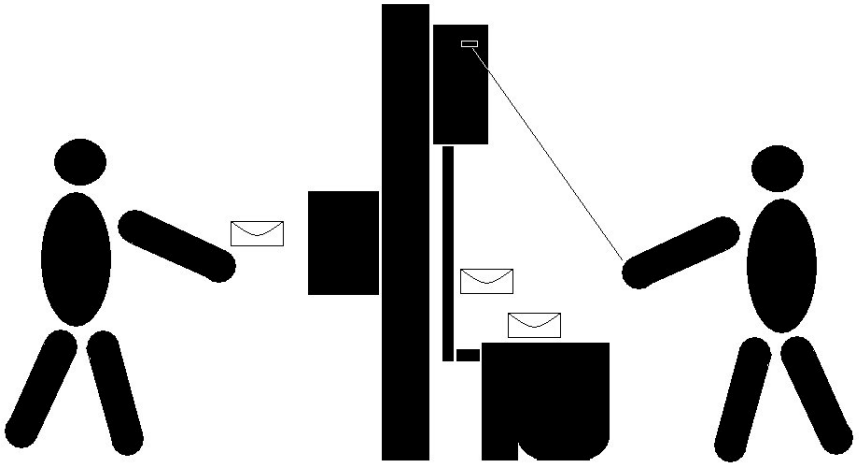
Mike Schwarz <ms111@hdm-stuttgart.de>
Matthias Frenzel <mf044@hdm-stuttgart.de>
Markus Zobel <mz014@hdm-stuttgart.de>
Norman Pohl <np012@hdm-stuttgart.de>

Zusammenfassung

Diese Dokumentation beschreibt unsere Arbeit an „SpamGourmet“, einer Software, die wir im Sommersemester 2006 an der HdM in Stuttgart erstellt und im Rahmen eines Softwareprojektes bearbeitet haben. Es handelt sich dabei um einen Service, der temporäre Email-Adressen auf real existierende Adressen mappt. Damit ist es möglich z.B. beim Registrieren im Internet eine solche temporäre Adresse anzugeben um ein Passwort zu erhalten. Nachfolgende Spammails können dann leicht über Regeln gefiltert und verworfen werden.

Betreut wurde das Projekt von Prof. Dr. Martin Goik.

Diese Hilfe ist auch als PDF [../pdf/de_docu.pdf] erhältlich.



Projekt

Projektbeschreibung

Der Anbieter http://www.spamgourmet.com bietet die Möglichkeit zur Definition temporärer E-Mailadressen, welche mit einer nur dem Benutzer bekannten, persönlichen E-Mail Adresse verbunden sind. Auf diese Weise kann man beispielsweise für die License Key Zusendung einer Software Teststellung anstelle der eigenen E-Mailadresse eine solche Temporäradresse verwenden und auf diese Weise die spätere Zusendung von Werbemüll vermeiden.

Die Idee des Projektes besteht nun darin, an der HdM einen eigenen Service zu etablieren, welcher folgende Randbedingungen erfüllt:

- Authentifizierung der zulässigen User (Studenten, Angestellte, Professoren) über den HdM LDAP Server
- Definition (fast) beliebiger E-Mail Alias Definitionen, z.B.

  anton@tmpmail.hdm-stuttgart.de
- Möglichkeit zur Deaktivierung oder Begrenzung der Lebensdauer definierter Alias Adressen
- Forwarding an beliebige eigene E-Mailadressen.

Prof. Dr. Martin Goik

Anforderungen

Anwendung zum Erstellen und Verwalten von temporären Wegwerf-E-Mailadressen.

- Weboberfläche:
  - um persönliche Einstellungen und Adressen zu erstellen und verwalten
  - Datenbankanbindung um die Emailadressen zu speichern
  - Benutzerauthentifizierung über HdM LDAP Server

    Wenn keine weiteren Daten vom Server abgefragt werden müssen kann der Apache Tomcat die Authentifizierung über einen LDAP Server durchführen.
- Weiterleitung an den Emailserver der HDM
- Filterregeln:
  - Maximale Anzahl zu empfangender Emails.
  - Gültigkeitszeitraum der temporären Adresse.
  - Emails von bestimmten Absenderdomains akzeptieren bzw. verwerfen.
- Emails, die nicht weitergeleitet werden sollen auch nicht angenommen werden.
- Jeder Benutzer hat die Möglichkeit, beliebige Email Adressen definieren zu können.

Datenbank

Klassen-Schema

In diesem Kapitel werden die Objekte und deren Beziehungen zueinander kurz beschrieben. Eine genauere Beschreibung ist im Kapitel „Beans“.

Mapping mit Hibernate

In diesem Kapitel:

- wie wird gemappt?
- warum wird so gemappt?

Mit Hibernate können Objekte in einer relationalen Datenbank gespeichert und aus ihr gelesen werden. Das Programm muss sich nicht um die Struktur und Beziehungen in der Datenbank kümmern. Aus Sicht des Programms setzt Hibernate die Objekte in die relationale Welt um. Die Regeln für die Umsetzung entnimmt Hibernate aus Mapping-Dateien. Das sind XML-Dateien, die beschreiben, welche Klassen auf welche Tabellen abgebildet werden sollen. Das kann in einer Datei angegeben werden. Um eine bessere Übersicht zu erhalten wird aber für jede Klasse eine separate Mapping-Datei verwendet. Dabei hat jede Mapping-Datei die gleiche Bezeichnung wie ihre zugehörige Klasse, jedoch mit der Endung „.hbm.xml“.

```
<hibernate-mapping package="de.sg.beans" default-cascade="all"
         default-lazy="false">
  <class name="Namespace">

    <id name="id" type="long">
      <generator class="native"/>
    </id>

    <property name="namespace"
              type="string"
              length="40" unique="true"
              generated="never" not-null="true"
              insert="true" update="true"/>

  </class>
</hibernate-mapping>
```

Das Wurzelelement jeder Mapping-Datei ist *hibernate-mapping*. Es kann ein oder mehrere *class*-Elemente beinhalten, in denen die jeweilige Abbildungsvorschrift enthalten ist. Das *name*-Attribut legt dabei die Klassenbezeichnung fest. Optional kann hier auch noch ein Attribut *table* angegeben werden, falls der Name der Klasse nicht mit dem Namen der Tabelle übereinstimmen sollte.

Im Wurzelelement wird über das Attribut *default-cascade* festgelegt, dass wenn zum Beispiel ein Benutzer aktualisiert wird auch sein Profil gesichert wird. *default-lazy* gibt an, ob alle Daten sofort aus der Datenbank geladen werden sollen, oder erst beim Zugriff darauf. Per Default lädt Hibernate alle Collections und Referenzen lazy nach. Das kann man durch die Angabe `default-lazy="false"` ändern.

Das Element *id* gibt an, wie Hibernate mit dem Primärschlüssel umgehen soll. Der Name ist in jeder Tabelle *id* und vom Typ *long*. Diese Nummer wird von der Datenbank selbst in aufsteigender Folge vergeben.

Die *property*-Elemente ordnen den Attributen die entsprechenden Spalten der Tabelle zu. *Namespace* enthält eine Spalte *namespace*, in der eine Zeichenkette mit einer maximalen Länge von 40 Zeichen abgelegt werden kann. Sie muss innerhalb der Tabelle eindeutig sein, was im Attribut *unique* festgelegt wird. Die Groß und Klein Schreibung wird nicht berücksichtigt (case-insensitive). Beispiel: „NameSpaceName“ ist dasselbe wie „namespacename“

Mehrere Namespaces können zu einem Profil gehören. Im Java-Code haben die jeweiligen Profile Referenzen auf ihre Namespaces. In der Datenbank jedoch muss ein Fremdschlüsselverweis in der Tabelle *Namespace* auf die Tabelle *Profile* existieren. Für ein solches Mapping mit Hibernate gibt es das *list*-Element:

```
<list name="namespaces">
  <key column="profileId" not-null="true"/>
  <list-index column="listIndex"/>
  <one-to-many class="de.sg.beans.Namespace"/>
</list>
```

Jedes *list*-Element muss ein *key*-Element und ein *list-index*-Element enthalten. Der *key* gibt an, welche Spalte den Fremdschlüssel enthält und der *list-index* bezeichnet die Spalte, die für jeden Eintrag die laufende Nummer (Array-Index) enthält. Das Element *one-to-many* gibt an, dass es sich um eine 1:n Beziehung handelt, d.h. ein Profil kann mehrere Namespaces haben. Die Fremdschlüssel-Spalte in der Tabelle *Namespace* für das Mapping der Beziehung *Profile* und *Namespace* hat die Bezeichnung „profileId“ und verweist auf den Primärschlüssel der *Profile*-Tabelle. Angegeben wird dies aber in der Mapping-Datei „Profile.hbm.xml“ des Profils.

Das Element *one-to-one* in der Datei „User.hbm.xml“ definiert eine 1:1 Beziehung zwischen einem Benutzer und einem Profil. Zu jedem Benutzer gehört genau ein Profil und umgekehrt hat ein Profil genau einen Benutzer, für den es die Daten enthalten kann.

```
<one-to-one name="profile" class="Profile"/>
```

Für diese Beziehung muss kein separates Feld existieren. Sie kann in der Datenbank so abgebildet werden, dass zwei zusammengehörige Datensätze denselben Primärschlüsselwert haben.

Um Vererbungshierarchien in einer Datenbank zu speichern gibt es bei Hibernate 3 Möglichkeiten:

- Tabelle je Klasse
- Tabelle je konkrete Klasse
- Tabelle je Klassenhierarchie

Für das Speichern der Regeln wird die ganze Klassenhierarchie in einer Tabelle abgelegt.

Das Element *discriminator* gibt die Spalte an, mit deren Inhalt Hibernate die verschiedenen Klassen auseinander hält. Der Wert wird als Zeichenkette mit einer Länge von 2 Zeichen angegeben.

```
<discriminator column="disc"
               type="string" length="2"
               force="false" not-null="true"/>
```

Jedes *subclass*-Element definiert das Mapping für eine der Subklassen und gibt dabei als *discriminator-value* an, welcher Wert für diese Klasse in das Discriminator-Feld geschrieben wird.

```
<subclass name="de.sg.beans.TimeRule" discriminator-value="tr">

  <property name="startTime"
            type="long"
            unique="false"
            generated="never"
            optimistic-lock="true"
            access="property"
            insert="true" update="true"
            not-null="true"/>

  <property name="endTime"
            type="long"
            unique="false"
            generated="never"
            optimistic-lock="true"
            access="property"
            insert="true" update="true"
            not-null="true"/>

</subclass>
```

Dieses Wissen reicht nun bereits aus um alle Mapping-Dateien zu verstehen. Weitere Informationen zum Hibernate-Mapping ist unter

http://www.hibernate.org/hib_docs/reference/en/html_single/#mapping

zu finden.

Tabellen

Die Tabelle Namespace.

Tabelle 1. Namespace

Column Name	Data Type	Size	Decimal Digits	AcceptNullValue	
id	bigint	20	0	NO	
namespace	varchar	40	0	NO	
profileId	bigint	20	0	NO	
listIndex	int	11	0	YES	

Die Tabelle Profile.

Tabelle 2. Profile

Column Name	Data Type	Size	Decimal Digits	AcceptNullValue	
id	bigint	20	0	NO	

Column Name	Data Type	Size	Decimal Digits	AcceptNullValue	
langCode	varchar	2	0	NO	
defaultCount	int	11	0	NO	
realEmail	varchar	255	0	NO	

Die Tabelle Rule.

Tabelle 3. Rule

Column Name	Data Type	Size	Decimal Digits	AcceptNullValue	
id	bigint	20	0	NO	
disc	varchar	2	0	NO	
expired	bit	1	0	NO	
startTime	bigint	20	0	NO	
endTime	bigint	20	0	NO	
domainName	varchar	255	0	NO	
acceptDomain	bit	1	0	NO	
maxNumber	int	11	0	NO	
currentNumber	int	11	0	NO	
tempEmailId	bigint	20	0	NO	
listIndex	int	11	0	YES	

Die Tabelle TempEmail.

Tabelle 4. TempEmail

Column Name	Data Type	Size	Decimal Digits	AcceptNullValue	
id	bigint	20	0	NO	
tempEmailAddress	varchar	255	0	NO	
active	bit	1	0	NO	
logic	int	11	0	NO	
userId	bigint	20	0	NO	
listIndex	int	11	0	YES	

Die Tabelle User.

Tabelle 5. User

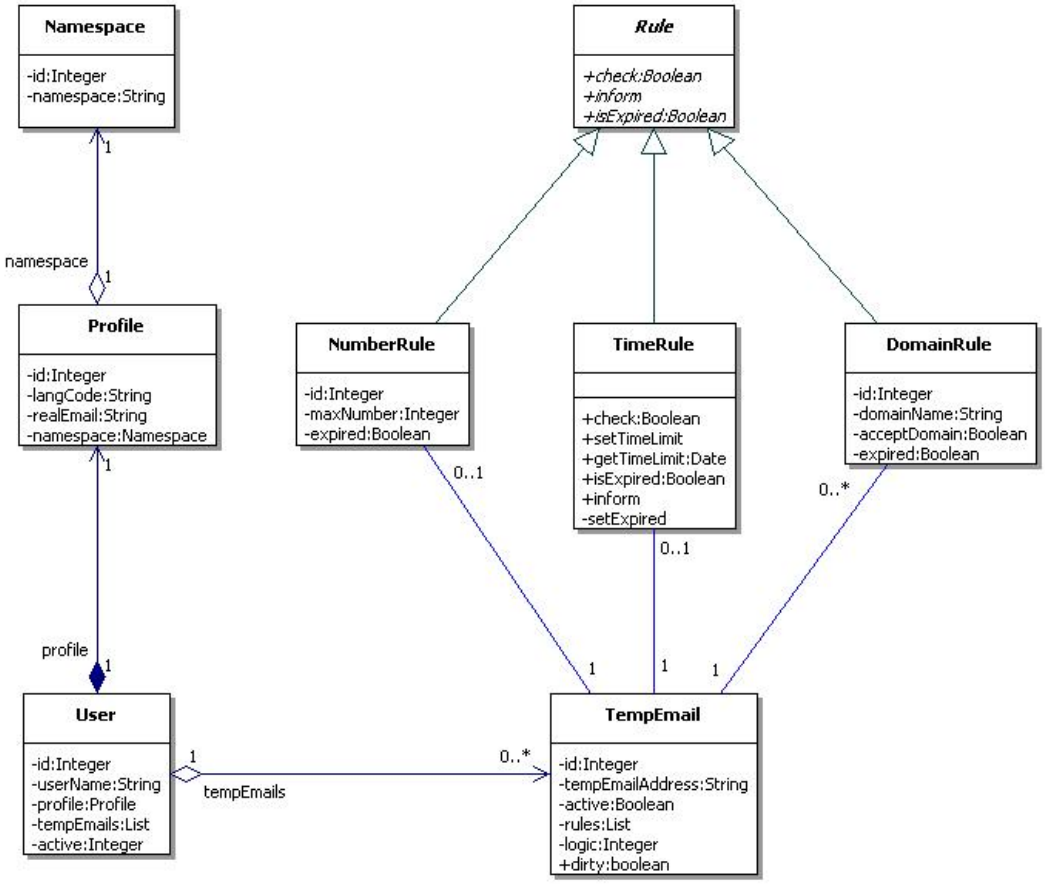
Column Name	Data Type	Size	Decimal Digits	Accept NullValue	
id	bigint	20	0	NO	

Column Name	Data Type	Size	Decimal Digits	Accept NullValue	
userName	varchar	60	0	NO	
active	bit	1	0	NO	

Beans

In diesem Kapitel werden die Beans genau beschrieben.

Die Beans dienen als Datenspeichercontainer. Sie beinhalten die Benutzerdaten. Über die Schnittstelle *UserController* können die Beans aus der Datenbank geholt bzw. in die Datenbank geschrieben werden.



An diesem Klassendiagramm kann man erkennen, dass der *User* das zentrale Objekt ist. Von ihm aus können alle benutzerspezifischen Daten erreicht werden. Jeder *User* hat ein einziges *Profil*, das seine Eigenschaften enthält. Des Weiteren kann ein *User* beliebig viele *TempEmail* Objekte haben. Sie repräsentieren die temporären Emailadressen mit allen zugehörigen Regeln. Es können 3 verschiedene Regeln angewendet werden.

- TimeRule zur Freigabe innerhalb eines bestimmten Zeitraumes
- NumberRule zur Freigabe einer bestimmten Anzahl zu empfangender Emails
- DomainRule zur Angabe einer bestimmten Domain, von der Emails empfangen bzw. verweigert werden

Die folgenden Methoden werden von jedem Bean implementiert.

Von der Datenbank wird eine *id* als Primärschlüssel vergeben. Diese *id* dient zur eindeutigen Identifizierung des Datensatzes. In der Datenbank wird der Wert als *bigint* Ganzzahl abgelegt. Mit Hilfe von Hibernate wird die *id* im jeweiligen Objekt auf einen *long*-Wert gemappt. Die Zugriffsmethoden werden versteckt, um sicherzustellen, dass der Wert nur von Hibernate gesetzt werden kann. Für die weiteren Funktionen der Anwendung ist

die *id* nicht relevant.

```
private long getId()
```

Gibt die Primärschlüssel-ID des jeweiligen Datensatzes in der Datenbank zurück. Wird nur von Hibernate verwendet. Die id wird von der Datenbank selbst fortlaufend vergeben.

```
private void setId(final long newId)
```

Setzt die Primärschlüssel-ID des jeweiligen Datensatzes in der Datenbank.

```
public final boolean equals(final Object aObject)
```

Prüft, ob das übergebene Objekt gleich ist. Diese Methode wird zu Testzwecken verwendet.

Namespace

Das Namespace Bean speichert die Daten zum Namensraum. Ein Namensraum wird vom Benutzer bei Registrierung an der Weboberfläche gewählt. Er muss dabei eindeutig sein. Das heisst jeder Namensraum darf nur an einen Benutzer vergeben werden. Innerhalb eines Namensraums können dann frei nach Belieben Emailadressen angelegt werden.

Angenommen an der Weboberfläche wird der Namensraum „ababab“ ausgewählt. Dann kann anstelle des X eine beliebige für Emailadressen gültige Zeichenkette verwendet werden.

```
X.ababab@hdm-stuttgart.de
```

Sinnvollerweise begrenzt die Weboberfläche diese Zeichenanzahl auf minimal 5 Zeichen. Von der Datenbank werden jedoch alle Zeichenketten akzeptiert. So können Änderungen dieser Regelung später leichter durchgeführt werden.

Zum Beispiel

```
tempmail.ababab@hdm-stuttgart.de
peter.ababab@hdm-stuttgart.de
```

Innerhalb eines anderen Namespaces „blubbelBlubb“ kann „peter“ trotzdem verwendet werden.

```
peter.blubbelBlubb@hdm-stuttgart.de
```

Ohne Namespaces wäre das Anlegen unter Umständen um einiges schwieriger. Einfache Namen und kurze Bezeichnungen wären schon sehr früh vergeben. Beim Anlegen würde der Benutzer ständig aufgefordert werden, sich einen neuen Namen auszudenken weil der ausgewählte bereits vergeben ist. Das Problem würde sich bei steigender Anwenderanzahl immer weiter vergrössern, bis letztendlich der Service unbenutzbar und unkomfortabel erscheinen würde.

Methoden.

```
public final String getNamespace()
```

Gibt den Namespace als Zeichenkette zurück.

```
public final void setNamespace(final String newNamespace)
```

Setzt den Namespace. Als Parameter wird eine Zeichenkette mit dem neuen Namespace erwartet.

Profile

Das Profile Bean speichert die Profildaten eines Benutzers. Im Profil sind die Namespaces und die echte Emailadresse zu finden. Alle Emails an eine existierende temporäre Adresse werden nach erfolgreicher Prüfung aller angelegter Regeln an die echte Adresse weitergeleitet.

Der *DefaultCount* wird beim Registrieren festgelegt und gibt die maximale Anzahl der Emails vor, die von einer neuen temporären Emailadresse empfangen werden dürfen. Ist der Wert erreicht, werden zukünftig keine Emails mehr akzeptiert.

Der Sprachcode *LangCode* legt die bevorzugte Sprache des Benutzers fest. In dieser Sprache erscheint der Text auf der Weboberfläche. Derzeit ist es nur möglich den Text auf englisch und deutsch anzeigen zu lassen. Für weitere Sprachunterstützung müssen die Texte allerdings nur in die entsprechende, gewünschte Sprache übersetzt werden.

Im Bean wird eine Liste von *Namespaces* gespeichert. Von der Weboberfläche wird derzeit nur ein *Namespace* pro Benutzer erlaubt. Das liegt an der Vermutung, dass die Benutzer sonst zu viele Namespaces unnötig anlegen würden und damit der eigentlichen Sinn des Namespaces verfehlt wäre. Die Begrenzung stellt auch keine echte Einschränkung dar, da trotzdem innerhalb des Namespace beliebig viele beliebige Adressen angelegt werden können. Trotzdem ist das Speichern der Namespaces in einer Liste flexibler. Damit ist das Verwalten mehrerer Namespaces datenbankseitig ohne Änderungen möglich.

Methoden.

```
public final String getLangCode()
```

Gibt die bevorzugte Sprache des Profils zurück. Sie wird zur Einstellung der Sprache der Weboberfläche verwendet.

```
public final void setLangCode(final String newLangCode)
```

Setzt die bevorzugte Sprache des Profils. Als Parameter wird eine Zeichenkette mit dem neuen Sprachcode erwartet.

```
public final int getDefaultCount()
```

Gibt die Standardanzahl zu empfangender Emails, die beim Anlegen einer neuen temporären Emailadresse verwendet werden, zurück. Der Wert gibt vor, wie viele Emails maximal von dieser temporären Adresse empfangen werden sollen. Diese Regel kann explizit im NumberRule Bean der TempEmail geändert werden.

```
public final void setDefaultCount(final int newDefaultCount)
```

Setzt die Standardanzahl zu empfangender Emails.

```
public final List<Namespace> getNamespaces()
```

Gibt eine Liste der Namespaces dieses Profils zurück. Von der Weboberfläche wird bis jetzt nur ein Namespace erlaubt. Die Liste dient der Vorbereitung falls später jedes Profil mehrere Namespaces verwenden darf.

```
private final void setNamespaces(final List<Namespace> namespaceList)
```

Setzt die Liste der Namespaces dieses Profils. Die Referenz der Liste sollte nicht verändert werden, um Fehler sowie unnötige Einträge in der Datenbank zu vermeiden. Deshalb ist diese Methode nicht sichtbar. Diese Methode wird für das Mapping der Objekte mit Hibernate verwendet. Hibernate kommt mit versteckten Methoden zurecht.

```
public final String getRealEmail()
```

Gibt die reale Emailadresse des Benutzers zurück. Alle Emails an eine temporäre Emailadresse werden geprüft und diese Adresse weitergeleitet.

```
public final void setRealEmail(final String newRealEmail)
```

Setzt die reale Emailadresse des Benutzers.

```
public final void addNamespace(final Namespace newNamespace)
```

Fügt der Liste einen neuen Namespace hinzu. Als Parameter wird der neue Namespace erwartet.

```
public final void removeNamespace(final Namespace newNamespace)
```

Entfernt den übergebenen Namespace aus der Liste, falls er vorhanden ist. Als Parameter wird der Namespace, der entfernt werden soll, erwartet.

TempEmail

Das TempEmail Bean speichert alle Daten der temporären Emailadresse.

Es beinhaltet den Adressnamen bzw. die Adressbezeichnung. Jeder Name kann selbstverständlich nur einmal angegeben werden. Durch das Verwalten von Namespaces ist es jedoch für verschiedene Benutzer möglich, den gleichen Namen zu verwenden. Genaueres ist bei Namespace zu finden.

Die temporäre Adresse kann über einen Kippschalter aktiviert und deaktiviert werden. An eine deaktivierte temporäre Adresse adressierte Email wird sofort verworfen. Die Daten bleiben jedoch erhalten. Das heisst, dass die Adresse jederzeit wieder aktiviert werden kann, wobei die „alten“ Regeln wieder angewandt werden.

Die Regeln der temporären Adresse werden in einer Liste gespeichert. So kann der

Benutzer beliebig viele Regeln festlegen. Prinzipiell ist es möglich z.B. mehrere Regeln festzulegen, die je eine verschiedene maximale Anzahl zu empfangender Emails vorgeben. Da dies aber sinnlos ist, wird von der Weboberfläche geprüft, ob bereits eine solche Regel existiert. Falls ja wird ein erneutes Anlegen einer solchen Regel nicht mehr ermöglicht.

Mehrere Regeln werden logisch verknüpft. Dabei gibt es die boolschen Operationen „UND“ bzw. „ODER“. Bei der „UND“-Verknüpfung müssen alle Regeln zutreffen damit die Email im Postfach landet. Die „ODER“-Verknüpfung erfordert nur eine zutreffende Regel für die Weiterleitung. Wie die Regeln zu verknüpfen sind wird im Attribut *logic* festgelegt. Wenn nichts anderes angegeben wird verwendet die Anwendung die „UND“-Verknüpfung.

Methoden.

```
public final boolean isActive()
```

Gibt an, ob diese temporäre Emailadresse aktiv ist. Falls sie deaktiviert wurde werden unabhängig der Regeln keine Emails weitergeleitet.

```
public final void setActive(final boolean newActive)
```

Setzt die temporäre Adresse als aktiv bzw. inaktiv.

```
public final List<Rule> getRules()
```

Gibt die Liste der Regeln zurück.

```
private void setRules(final List<Rule> rulesList)
```

Setzt die Liste der Regeln dieser temporären Adresse. Die Referenz der Liste sollte nicht verändert werden, um Fehler sowie unnötige Einträge in der Datenbank zu vermeiden. Deshalb ist diese Methode nicht sichtbar. Diese Methode wird für das Mapping der Objekte mit Hibernate verwendet. Hibernate kommt mit versteckten Methoden zurecht.

```
public final String getTempEmailAddress()
```

Gibt die temporäre Emailadresse zurück. Alle Emails an diese Adresse werden auf die Regeln geprüft und gegebenenfalls an die reale Adresse des Profils weitergeleitet.

```
public final void setTempEmailAddress(final String newTempEmailAddress)
```

Setzt die temporäre Emailadresse. Als Parameter wird eine Zeichenkette mit der neuen temporären Adresse erwartet.

```
public final int getLogic()
```

Gibt an wie die Regeln logisch verknüpft (UND / ODER) werden.

```
public final void setLogic(final int newLogic)
```

Legt fest wie die Regeln logisch verknüpft (UND / ODER) werden.

```
public final void addRule(final Rule rule)
```

Fügt der Regelliste eine neue Regel hinzu. Die Regeln werden mit der Operation, die in *logic* steht, verknüpft.

```
public final void removeRule(final Rule rule)
```

Entfernt die übergebene Regel aus der Regelliste, falls sie dort vorhanden ist.

```
public final boolean isDirty()
```

Gibt an, ob die temporäre Adresse (dieses Objekt) seit dem letzten Speichervorgang verändert wurde.

Tipp

Momentan wird dies nicht verwendet. Am Anfang des Projektes war geplant, das der User explizit seine Änderungen speichern muss, bzw vor dem Ausloggen darauf hin gewiesen wird, das die Einstellungen XY noch nicht gespeichert wurden. Im aktuellen Betrieb werden die Änderungen sofort gespeichert, ohne das es explizit eine Eingabe vom User geben muss.

```
public final void setDirty(final boolean isDirty)
```

Legt fest, ob die temporöre Adresse (dieses Objekt) seit dem letzten Speichervorgang verändert wurde.

Tipp

siehe Methode isDirty.

User

Das *User* Bean speichert alle benutzerspezifischen Daten und legt fest, ob der Benutzeraccount aktiv ist oder nicht. Ein inaktiver Benutzer kann sich an der Weboberfläche nicht mehr anmelden und es werden auch keine Emails mehr weitergeleitet. Seine Daten bleiben aber trotzdem erhalten. Das heißt, dass der Account jederzeit wieder aktiviert werden kann und der Benutzer den Dienst mit seinen „alten“ Daten weiternutzen kann.

Das *User* Bean ist zudem, wie bereits erwähnt, das zentrale Objekt, von dem aus alle Daten erreicht werden können.

Der Benutzername des *User* identifiziert den Benutzer.

Jeder *User* hat genau ein Profil. Es kann nicht durch ein anderes ersetzt werden. Die Gründe liegen in der Abbildung der 1-zu-1 Beziehung zwischen den beiden Tabellen. Genaueres ist im Kapitel „Datenbank“ beim one-to-one-Mapping zu finden. Um die Werte des Profils zu verändern muss das Profil des Benutzers über die get-Methode geholt werden. Danach können Veränderungen vorgenommen werden.

Ein Beispiel:

```
User user = UserController.getInstance().getUserByName(username);
Profile profile = user.getProfile();
```

```
profile.setDefaultCount(5);
UserController.getInstance().save(user);
```

Ein Benutzer kann beliebig viele temporäre Emailadressen anlegen. Sie werden in einer Liste verwaltet. Die Liste selbst soll, wie das Profil, nicht ersetzt werden. Diesmal aus dem Grund, dass die „alten“, nicht mehr benötigten Daten sonst in der Datenbank bleiben würden und nicht mehr erreichbar sind.

Methoden.

```
public final boolean isActive()
```

Gibt an, ob der Benutzer aktiv ist, oder nicht.

```
public final void setActive(final boolean newActive)
```

Legt fest, ob der Benutzer aktiv ist.

```
public final Profile getProfile()
```

Gibt das Profil des Benutzers zurück.

```
private final void setProfile(final Profile newProfile)
```

Setzt das Profil dieses Benutzers. Die Referenz sollte nicht verändert werden, um Fehler sowie unnötige Einträge in der Datenbank zu vermeiden. Deshalb ist diese Methode nicht sichtbar. Diese Methode wird für das Mapping der Objekte mit Hibernate verwendet. Hibernate kommt mit versteckten Methoden zurecht.

```
public final List<TempEmail> getTempEmails()
```

Gibt die Liste der temporären Emailadressen zurück. Die Referenz der Liste sollte nicht verändert werden, um Fehler sowie unnötige Einträge in der Datenbank zu vermeiden. Deshalb ist diese Methode nicht sichtbar. Diese Methode wird für das Mapping der Objekte mit Hibernate verwendet. Hibernate kommt mit versteckten Methoden zurecht.

```
private void setTempEmails(final List<TempEmail> tempEmailsList)
```

Setzt die Liste der temporären Emailadressen.

```
public final String getUserName()
```

Gibt den Benutzernamen zurück.

```
public final void setUserName(final String newUserName)
```

Setzt den Benutzernamen. Als Parameter wird eine Zeichenkette mit dem neuen Benutzernamen erwartet.

```
public final void addTempEmail(final TempEmail tempEmail)
```

Fügt dem Benutzer eine neue temporäre Emailadresse hinzu. Emails an diese Adresse werden auf ihre zugehörigen Regeln geprüft und gegebenenfalls an die reale Adresse weitergeleitet.

```
public final void removeTempEmail(final TempEmail tempEmail)
```

Entfernt die übergebene temporäre Emailadresse, falls sie in der Liste existiert.

```
public final boolean isDirty()
```

Gibt an, ob die Daten des Users (dieses Objekt) seit dem letzten Speichervorgang verändert wurde.

```
public final void setDirty(final boolean dirty)
```

Legt fest, ob die Daten des Users (dieses Objektes) seit dem letzten Speichervorgang verändert wurde.

Rule

Für die Gültigkeit der temporären Emailadressen existieren 3 Regeln:

- NumberRule
- TimeRule
- DomainRule

Emails an eine temporäre Adresse werden nur weitergeleitet wenn alle dieser Regeln zutreffen.

Die Regeln können mit den boolschen Operatoren „UND“ bzw. „ODER“ miteinander verknüpft werden. Bei der UND-Verknüpfung wird eine Email erst akzeptiert, wenn alle Regeln zutreffen. Bei der ODER-Verknüpfung reicht es, wenn eine der angegebenen Regeln zutrifft.

Alle Regeln werden in der Datenbank in einer einzigen Tabelle abgebildet. Für jede konkrete Regel wird ein Kürzel in einer bestimmten Spalte der Datenbank abgelegt, anhand der Hibernate entscheidet, um welche konkrete Regel es sich handelt. Näheres dazu im Kapitel Datenbank.

NumberRule

Das NumberRule Bean implementiert eine Regel, die prüft, ob eine vorgegebene, maximale Anzahl an Emails bereits empfangen wurde.

Methoden.

```
public final boolean check(final Object email)
```

Prüft, ob die Anzahl der empfangenen Emails bereits erreicht ist.

```
public void inform(final boolean emailAccepted)
```

Informiert die Regel über eine akzeptierte Email...

```
public final int getMaxNumber()
```

Gibt die maximale Anzahl der Emails zurück, die von dieser Regel akzeptiert wird, zurück.

```
public final void setMaxNumber(final int newMaxNumber)
```

Setzt die maximale Anzahl der Emails, die von dieser Regel akzeptiert wird, zurück.

```
public final int getCurrentNumber()
```

Gibt die Anzahl der Emails zurück, die bisher von dieser Regel akzeptiert wurden.

```
public final void setCurrentNumber(final int newCurrentNumber)
```

Setzt die Anzahl der Emails, die bisher von dieser Regel akzeptiert wurden.

TimeRule

Das TimeRule Bean implementiert eine Regel, die prüft, ob das aktuelle Datum innerhalb eines vorgegebenen Zeitraums liegt. Liegt es ausserhalb wird die Email verworfen.

Methoden.

```
public final boolean check(final Object email)
```

Prüft, ob der aktuelle Zeitpunkt in dem in der Regel festgelegten Zeitraum liegt.

```
public void inform(final boolean emailAccepted)
```

Informiert die Regel über eine akzeptierte Email...

```
public final long getStartTime()
```

Gibt den Startzeitpunkt, ab dem diese Regel Emails akzeptiert, zurück.

```
public final void setStartTime(final long newStartTime)
```

Legt den Startzeitpunkt, ab dem diese Regel Emails akzeptiert, fest.

```
public final long getEndTime()
```

Gibt den Endzeitpunkt, ab dem diese Regel keine Emails mehr akzeptiert, zurück.

```
public final void setEndTime(final long newEndTime)
```

Legt den Endzeitpunkt, ab dem diese Regel keine Emails mehr akzeptiert, fest.

DomainRule

Das DomainRule Bean implementiert eine Regel, die die Domain des Absenders prüft. Die Regel entscheidet, ob Emails von dieser Domain empfangen werden dürfen, oder nicht.

Methoden.

```
public final boolean check(final Object email)
```

Prüft, ob die Email dieser Absenderdomain akzeptiert wird.

```
public void inform(final boolean emailAccepted)
```

Informiert die Regel über eine akzeptierte Email...

```
public final boolean isAcceptDomain()
```

Gibt an, ob diese Regel Emails dieser Domain akzeptiert.

```
public final void setAcceptDomain(final boolean newAcceptDomain)
```

Legt fest, ob diese Regel Emails dieser Domain akzeptieren soll.

```
public final String getDomainName()
```

Gibt den Domainnamen zurück, von dem Emails akzeptiert bzw. verweigert werden sollen.

```
public final void setDomainName(final String newDomainName)
```

Setzt den Domainnamen, von dem Emails akzeptiert bzw. verweigert werden sollen.

Struts

View Controller (MVC) Prinzip

Der MVC-Ansatz sieht eine Aufteilung der Applikation in drei Schichten vor:

- die Datenrepräsentation und Geschäftslogik (Model)
- die Darstellung und Interaktion mit dem Benutzer (View)
- einem Vermittler zwischen diesen beiden Komponenten (Controller)

Bei webbasierten Systemen besteht der Controller natürlich aus einem Servlet, das zunächst alle eingehenden Requests entgegennimmt und dann entsprechend seiner Konfiguration an verschiedene Model-Komponenten weiterleitet. Ist die Anfrage bearbeitet. Leitet es den Request an eine JSP weiter, die die Ausgabe übernimmt.

Struts-basierte Komponenten werden über eine zentrale Konfigurationsdatei (struts-config.xml) konfiguriert. Diese Datei wird vom ActionServlet während der Initialisierung geparst und verarbeitet. In dieser werden werden alle verwendeten Komponenten benannt und miteinander verknüpft.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
    "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">
<struts-config>
```

FormBeans sind spezielle JavaBeans, die die Aufgabe haben, Formulardaten die der Benutzer in eine JSP einträgt, zur weiteren Verwendung zwischenzuspeichern. Die Überführung von Formulardaten in diese JavaBeans und zurück übernimmt das Framework.

```
<!-- ============================ Form Bean Definitions -->
<form-beans>
  <form-bean name="TempEmailForm" type="de.sg.form.TempEmailForm"
             dynamic="true">
    <form-property name="tempEmailName" type="java.lang.String" />
  </form-bean>
</form-beans>
```

Hier können globale exeptions definiert werden, die jedoch bei Spamgourmet bisher nicht benötigt wurden.

```
<!-- ======================= Global Exception Definitions -->
<global-exceptions></global-exceptions>
```

Mit der Definitionen von global-forwards hat man die Möglichkeit, einem symblolischen Link eine andere Action zuzuweisen bzw. eine Action unter verschiedenen Links ereichbar zu machen.

```
<!-- ========================= Global Forward Definitions -->

<global-forwards>
  <forward
    name="overview" path="/pages/overview.jsp" redirect="true">
  </forward>
</global-forwards>
```

Um eine lose Koppelung zwischen den JSP´s und der Anwenderlogik zu ermöglichen, verwenden wir statt fester Links vordefinierte ActionForwards, um von einer Komponente auf eine andere zu verweisen. Dadurch können wir einzelne Komponenten problemlos austauschen, indem wir einfach das Ziel unserer ActionForwards umkonfigurieren. Bei dem ActionMapping werden die ActionForwards mit unseren JSP´s und den zugehörigen Actions verbunden. Im oberen Mapping (siehe struts-config.xml Auszug unten) werden alle Anfragen mit den entsprechenden Namen, nach erfolgreicher Verarbeitung in der Action, an die jeweilige jsp weitergeleitet. In diesem Fall wird die {1} mit dem entsprechenden Namen ersetzt. Dies spart eine Menge Code, da nicht jedes einzelne Mapping definieret werden muss.

```
<!-- ========================== Action Mapping Definitions -->

<action-mappings>
  <action path="/actions/*Action" type="de.sg.actions.{1}Action">
    <forward name="loggedOut" path="/pages/loggedout.jsp" />
    <forward name="namespace" path="/pages/namespace.jsp" />
    <forward name="tempEmailEdit" path="/pages/tempEmail.jsp" />
    <forward name="settingsEdit" path="/pages/settings.jsp" />
  </action>
</action-mappings>
```

Hier wird ein Object (type) an einen definierten Pfad (path) gebunden. Scope gibt den Gültigkeitsbereich an, welche eine Form hat. Des weiteren wird mit den Attribut validate angegeben, ob in der Form eine Validierung der Formularfelder stattfinden soll. Mit dem name Attribut wird der Name der Form angegeben. Das Attribut input benennt die entsprechende jsp, in der sich die Formularfelder der Form befinden.

```
<!-- ========================== Action Mapping Definitions -->

<action-mappings>
  <!-- Namespace -->
  <action path="/actions/namespaceEditSave"
          type="de.sg.actions.NamespaceAction" name="NamespaceForm"
          scope="request" validate="true"
          input="/pages/namespace.jsp">
  </action>
</action-mappings>
```

Struts besitzt ebenfalls eine Template-Erweiterung, Tiles genannt. Sie ermöglichen dem Entwickler, seine Webseiten komponentenbasiert aufzubauen (Header, Footer, Content usw.). Tiles geht dabei weit über das statische Inkludieren von JSP-Inhalten hinaus. So können Tiles weitere Tiles inkludieren wohingehend einfache JSP Inkludierungen nur JSP Seiten einbeziehen können. Im folgenden struts-config.xml Auschnitt wird die Tiles Funktion aktiviert.

```
<!-- ============================ Controller Configuration -->

<controller
```

```
    processorClass="org.apache.struts.tiles.TilesRequestProcessor" />
```

Um eine dynamische mehrsprachigkeit zu gewährleisten, werden im folgenden die message.properties Dateien definiert. Je nachdem was der User im Browser für eine beforzugte Spracheinstellung gewählt hat, bzw er später in SpamGourmet eingestellt hat, wird der entsprechende Text aus der jeweiligen Sprachdatei dynamisch geladen. In unserem Projekt werden momentan die Sprachen Englisch und Deutsch unterstützt. Die Dateien haben folgendes Aussehen: Messages_de.properties und Messages.properties (default)

```
<!-- ======================== Message Resources Definitions -->

<message-resources parameter="Messages"/>
<message-resources parameter="Application" key="app"/>
```

Für Struts gibt es diverse Plugins, wie zum Beispiel ein "Tab-Menu". Will man ein solches PlugIn benutzen, kann man ein entsprechendes Plugin-Tag in der struts-config einfügen.

```
<!-- ======================== Plug Ins Configuration -->

<!-- ================================= Tiles plugin -->

<plug-in className="org.apache.struts.tiles.TilesPlugin">
  <!-- Path to XML definition file -->
  <set-property property="definitions-config"
                value="/WEB-INF/tiles-defs.xml" />
  <!-- Set Module-awareness to true -->
  <set-property property="moduleAware" value="true" />
</plug-in>

</struts-config>
```

Vorteile von Struts:

- Robust und fehlerarm, da Struts OpenSource ist.
- Leicht modifizierbar (struts-config.xml).
- Trennung der Komponenten. JSP Designer brauchen keine Java Kenntnisse.
- Kostenlos, da Struts OpenSource ist.
- Ständige Weiterentwicklung.

Nachteile von Struts:

- Längere Einarbeitungszeit.
- Wird Struts überhaupt nach JSF (Java-Server-Faces) weiterentwickelt?
- Probleme beim Einsatz von JSTL, bzw verschachtelten Struts-TagLibs.

Besipiel zum letzten Punkt:

```
<c:out value="${test}"/> bzw.
```

```
<html:hidden property="test" value="${index}"/>
```

wird nicht interpretiert. Andere JSTL Tags funktionien wiederrum. Ebenfalls eine Verschachtelung von struts-Taglibs ist nicht möglich, wie zum Beispiel:

```
<html:hidden property="index" value="<bean:write name='index'/>"/>
```

Die folgende Zeile funktioniert wieder herum:

```
<input type="hidden" name="index" value="<bean:write name='index'/>"/>
```

Tag Library

Einführung

Tag-Libraries sind ein Bestandteil der JSP-Spezifikation. Mit dem Einsatz von Java Server Pages hat man bereits erreicht, dass nur noch wenig Java-Code nötig ist, um eine dynamische Web-Seite zu erstellen. JSPs sind jedoch trotzdem nicht frei von Java-Code und deshalb für Web-Designer schwer zu bearbeiten.

Mit Hilfe von Tag-Libraries ist es möglich, JSP-Seiten zu entwickeln, die nur noch wenig bis gar keinen Java-Code beinhalten. Solche JSP-Seiten bieten dann die Schnittstelle zwischen dem Web-Designer, der kein Java versteht, und dem Web-Entwickler, der die dynamischen Teile einer Seite entwickelt. Tag-Libraries können zudem in mehreren JSP-Seiten verwendet werden.

Eine Tag-Library besteht aus einer Sammlung von Tag-Klassen und einer Tag-Library-Description (TLD). Tag-Klassen sind Java-Klassen, die eine bestimmte Schnittstelle implementieren. In der TLD steht für jedes Tag, welche Klasse dafür zuständig ist und welche Attribute es bietet. In der JSP können diese speziellen Tags in XML-Notation eingebunden werden, z. B.:

```
<mylib:mytag myattr1="25" myattr2="xyz"/>
```

Der Java-Code ist somit von der JSP-Seite in die Tag-Klasse ausgelagert.

Sobald die Abarbeitung einer JSP-Seite das Start- bzw. End-Tag eines Tags erreicht, ruft die Servlet-Engine bei der Tag-Klasse bestimmte Methoden auf. Die Tag-Klasse kann dann im Java-Code Berechnungen durchführen, Daten von einer Persistenzschicht lesen oder schreiben oder auch zusätzlichen HTML-Code in die Antwortseite schreiben.

Taglib Tutorial

Unter http://jakarta.apache.org/taglibs/tutorial.html ist ein Tutorial zu finden welches beschreibt wie einige der Tags der Jakarta-Taglibs library erstellt wurden beziehungsweise wie überhaupt solche Tag-Libraries erstellt werden können.

Erstellung der Taglib

Um eine Taglib zu erstellen wird als erstes die eigentliche Tag-Klasse(n) benötigt. Diese Tag Klassen können einfach von einer dieser Klassen abgeleitet werden:

```
javax.servlet.jsp.tagext.TagSupport
javax.servlet.jsp.tagext.BodyTagSupport
```

Die Klasse BodyTagSupport ist bereits von TagSupport abgeleitet und unterscheidet sich von der Klasse TagSupport indem sie die Manipulation des Bodys innerhalb des aktuellen Tags erlaubt.

Nun müssen eventuell die Methoden:

```
doStartTag()
doEndTag()
```

implementiert werden. Wie der jeweilige Name schon erahnen lässt handelt es sich dabei

um Methoden die gerufen werden wenn das Tag geöffnet wird beziehungsweise wenn es wieder geschlossen wird. In diesen Methoden wird nun die Business Logik untergebracht. Z.B. irgendwelche Berechnungen oder Datenbank Zugriffe etc., bei unserer Taglib wird anhand der *id* der entsprechende Text aus der XML Datei gelesen. Dies erfolgt in der Hilfs-Methode *getDocument(Locale)* abhängig von der eingestellten Locale des Webinterfaces. Wird keine passende XML Datei gefunden wird die Standard Datei *tooltips.xml* verwendet. Für die Realisierung dieser Hilfsmethode müssten wir allerdings noch eine kleine Änderung am XML Schema vornehmen. Das holen des tooltip Elements per *getElementById* funktioniert mit dem verwendeten XML Parser leider nur wenn die XML mit einer DTD validiert wird. Da die DTD´s noch nicht das elegante definieren der ID´s mit einem „Scope“ unterstützen mussten wir auf die Definition einer globalen ID zurückgreifen um das Schema mit den Möglichkeiten einer DTD gleichzuziehen. Aus dem angepassten Schema generierten wir nun die verwendete DTD.

Für die Attribute „id“ und „link“ des Tags mussten zusätzlich noch die benötigten Getter und Setter implementiert werden. Das Attribut link steht dafür ob zusätzlich noch ein Hyperlink zur entsprechenden Stelle im Handbuch generiert werden soll oder nicht. Im Beispiel (Screenshot) würde das bedeuten wenn man auf das Fragezeichen klickt würde ein Fenster aufgehen mit dem Handbuch und dort zur entsprechenden Stelle (wird ermittelt anhand der jeweiligen id) gesprungen.

```
public final int doStartTag() throws JspException {
    Locale currLoc = (Locale) pageContext.getSession()
            .getAttribute(Globals.LOCALE_KEY);
    Document doc = getDocument(currLoc);
    StringBuffer tooltipTag = new StringBuffer("<span");
    // Generieren des Links
    try {
        if (getLink() != null && getLink().equals("true")) {
            int pos1 = getId().indexOf(".");
            int pos2 = getId().indexOf(".", pos1 + 1);
            tooltipTag.append(" class=\"tooltip\""
                    + " onclick=\"showHelp('"
                    + ApplicationProperties.get("manual.base.url")
                    + "/manual." + getId().substring(pos1 + 1, pos2)
                    + ".html#" + getId() + ".anchor');\"");
        }
    } catch (Exception e) {
        LOGGER.info(e);
    }
    try {
        Element tooltipElement = doc.getElementById(getId());
        NodeList childs = tooltipElement.getChildNodes();
        String tooltipTitle = "", tooltipContent = "";
        for (int i = 0; i < childs.getLength(); i++) {
            Node node = childs.item(i);
            if (node.getNodeType() == Node.ELEMENT_NODE) {
                String name = node.getNodeName();
                if (name.equals("title")) {
                    try {
                        tooltipTitle = ((Element) node)
                                .getFirstChild().getNodeValue();
                    } catch (Exception e) {
                        LOGGER.info(e);
                        tooltipTitle = "";
                    }
                    LOGGER.info("Title: " + tooltipTitle);
                } else if (name.equals("content")) {
                    try {
                        tooltipContent = ((Element) node)
                                .getFirstChild().getNodeValue();
                    } catch (Exception e) {
                        LOGGER.info(e);
                        tooltipContent = "";
                    }
                    LOGGER.info("Content: " + tooltipContent);
```

```
                }
            }
        }
        if (!tooltipTitle.equals("") || !tooltipContent.equals("")) {
            tooltipTag
                    .append(" onmouseout=\"tooltip();\""
                            + " onmouseover=\"tooltip('"
                            + tooltipTitle
                            + "','"
                            + tooltipContent
                            + "');\"");
        }
    } catch (Exception e) {
        tooltipTag = new StringBuffer("<span");
    } finally {
        tooltipTag.append(">");
    }
    JspWriter out = pageContext.getOut();
    try {
        out.write(tooltipTag.toString());
    } catch (IOException e) {
        e.printStackTrace();
    }
    return BodyTagSupport.EVAL_BODY_INCLUDE;
}
```

In „doEndTag“ kann dann wieder aufgeräumt werden, evtl. die Datenbank Connection schließen oder ähnliches. Bei unserer *doEndTag* Methode wird lediglich das geöffnete „span“ Tag geschlossen und mit dem JspWriter auf die Ausgabe geschrieben.

Um dieses Tag nun zu verwenden benötigt man noch eine TLD (Tag Library Descriptor) Datei. Darin werden die einzelnen Attribute des Tags aufgezählt mit Name, ob sie zwingend erforderlich sind (required) und ob in den Attributen des Tags „RUN-TIME EXPRESSION VALUES“ wie zum Beispiel andere Tags oder Scriptlets erlaubt bzw. ausgewertet werden sollen. Zum Beispiel:

```
<mysimpleTag value="<%= scripletVariable %>"/>
```

Im folgenden ist der Inhalt dieser Datei aufgelistet.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE taglib PUBLIC
    "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
    "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
<taglib>
    <tlibversion>1.2</tlibversion>
    <jspversion>1.1</jspversion>
    <shortname>help</shortname>
    <uri>/tags/tempmail-help</uri>
    <tag>
        <name>tooltip</name>
        <tagclass>de.sg.taglibs.TooltipTag</tagclass>
        <bodycontent>JSP</bodycontent>
        <attribute>
            <name>id</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>link</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
```

```
</taglib>
```

Man könnte diese Klasse mit der Tag Library Descriptor Datei nun in ein JAR packen und in anderen Webanwendungen verwenden. Dort muss diese JAR Datei nur ins WEB-INF/lib Verzeichnis gelegt werden und über die web.xml Datei der Webanwendung angekündigt werden.

Um die Taglib in einer JSP zu verwenden muss diese noch der Webanwendung „angekündigt“ werden. Dies erfolgt in der web.xml Datei der jeweiligen Webanwendung.

```
<taglib>
    <taglib-uri>/tags/tempmail-help</taglib-uri>
    <taglib-location>/WEB-INF/tempmail-help.tld</taglib-location>
</taglib>
```

Jetzt muss diese Taglib nur in der JSP „importiert“ werden:

```
<%@ taglib uri="/WEB-INF/tempmail-help.tld" prefix="help"%>
```

Danach kann sie verwendet werden mit z.B.

```
<help:tooltip id="manual.overview.active.tooltip">
    ?
</help:tooltip>
```

Emailserver

Wie arbeitet der Email-Server und wie ist er aufgebaut?

ANT Script

Das Erstellen der Dokumentation erfordert einige Schritte. Es werden mehrere Dateien gelöscht, kopiert und erstellt. Ebenso müssen verschiedene XSLT-Prozesse angestossen werden. Damit das nicht alles „von Hand“ ausgeführt werden muss, ist ein ANT Script hilfreich.

Gesteuert wird Ant durch eine XML-Datei, die so genannte Build-Datei. In dieser wird zunächst ein Projekt definiert, welches *targets* enthält. Diese sind vergleichbar mit Funktionen in Programmiersprachen und enthalten u. a. Aufrufe von Tasks. Ein Task ist ein untrennbarer Arbeitsschritt. Zwischen den Targets sollten Abhängigkeiten definiert werden. Beim Aufrufen eines Targets löst Ant diese Abhängigkeiten auf und arbeitet die Targets entsprechend ab. Wenn man ein einzelnes Target definiert hat, welches direkt oder indirekt Abhängigkeiten zu allen anderen Targets hat, so genügt es, dieses aufzurufen und Ant führt dann alle notwendigen Arbeitsschritte in der richtigen Reihenfolge aus.

Es können *targets* öffentlich (public) oder privat (private) definiert werden. Zur Verwendung sind nur die öffentlichen *targets* relevant.

Die folgenden öffentlichen *targets* stellten wir zur Verfügung:

- generate-docu
- generate-tooltips
- all
- default

generate-docu. Dieses *target* erzeugt die Dokumentationen für das Spamgourmet Projekt inklusive des Handbuchs, sowie die Dokumentation des Tooltip-Imports in die Spamgourmet Anwendung. Es erstellt zuerst die Verzeichnisse für die HTML und PDF Dateien. Zuerst werden die HTML Seiten generiert, danach folgt die PDF Dokumente. Am Ende wird alles in eine „WAR“-Datei gepackt. Das ist eine Datei im JAR- bzw. ZIP-Format, die eine vollständige Webanwendung enthält.

generate-tooltips. Dieses *target* erstellt die Tooltips. Nach dem Generieren wird das Ergebnis auf Fehler mit Hilfe eines XML Schemas überprüft.

all. Dieses *target* führt alles aus. Es werden also die Dokumentation und die Tooltips erstellt.

default. Das *default-target* führt ebenfalls alles aus.

TODO

Pflicht

was muss noch gemacht werden?

Der **Service.**

Optionales

was sollte noch gemacht werden? (nice to have)

- Mappingdateien generieren. XDoclet funktioniert nicht mit Java5... Lösungsansatz mit Annotations...
- Email nach bestimmten Schlüsselwörtern überprüfen, die vorab vom User definiert und mit der temporären Email Adresse abgelegt wurden. Dementsprechend reagieren: Weiterleitung an den HDM - Mailserver oder verwerfen.
- Plugin für Mozilla Firefox mit dem neue temporäre Emailadressen komfortabel angelegt werden können. Context Menü, Basic (TempMail wird mit default count angelegt) und Extended User (wie im Webinterface)
- XML Interface (Soap), das den UserController ersetzt.
- Reply mit der Tempmail Adresse.
- Handy Client.

Verbesserungen, Veränderungen

Wie könnte die Anwendung verbessert werden? Was könnte oder müsste verändert werden?

- Geplant war, die Möglichkeit anzubieten, dass eine noch nicht existierende temporäre Adresse bei Erhalt einer Email neu angelegt wird. Allerdings besteht dann die Gefahr, dass sobald ein Namespace bekannt wird von den Spammern selbst Adressen angelegt werden können. Wir haben uns deshalb entschlossen, diese Funktion nicht zu implementieren. Ein möglicher Lösungsansatz hierfür wäre ein Firefox-Plugin zu implementieren (siehe "Optionales - nice to have").

Tools

Verwendete Tools und Techniken.

Subversion

Subversion (SVN) ist eine Open-Source-Software zur Versionsverwaltung.

Da es viele Schwächen des in Entwicklerkreisen sehr beliebten Programms CVS behebt, wird Subversion oft als dessen Nachfolger bezeichnet, obwohl es sich um ein eigenständiges Projekt handelt. Obwohl Subversion noch nicht lange existiert, gibt es bereits ausgereifte graphische Benutzeroberflächen. Dazu zählt beispielsweise TortoiseSVN (wird in den Windows-Explorer integriert). TortoiseSVN macht es den Benutzern des zugehörigen TortoiseCVS besonders leicht, ohne großen Einarbeitungsaufwand auf Subversion zu wechseln. Weiterhin sind Plugins für Eclipse verfügbar: Subclipse und Subversive SVN.

Java

Java ist eine objektorientierte Programmiersprache und als solche ein eingetragenes Warenzeichen der Firma Sun Microsystems. Sie ist eine Komponente der Java-Technologie. Java-Programme werden in einer speziellen Umgebung, der Java-Laufzeitumgebung oder Java-Plattform ausgeführt, deren wichtigster Bestandteil die Java Virtual Machine ist. Dazu werden Java-Programme in Bytecode übersetzt, der von der virtuellen Maschine ausgeführt wird. Java-Programme laufen in aller Regel ohne weitere Anpassungen auf verschiedenen Computern und Betriebssystemen. Sun bietet neben dem eigenen UNIX-Derivat Solaris auch Java-VMs für Linux und Windows an. Andere Hersteller, wie zum Beispiel Apple, lassen ihre Java-VM für ihre Plattform (Mac OS X) zertifizieren.

Eclipse

Eclipse ist eine in Java implementierte Open-Source-Entwicklungsumgebung.

Eclipse ist das englische Wort für Verfinsterung. Laut Erich Gamma, einem der Entwickler, soll der Name darauf hinweisen, dass Eclipse proprietäre Entwicklungsumgebungen in den Schatten stelle. Ein besonderer Seitenhieb auf IBMs Konkurrenten Sun Microsystems und dessen Entwicklungsumgebung NetBeans ergibt sich aus der Assoziation mit Sonnenfinsternis (engl. solar eclipse), da sun das englische Wort für Sonne ist.

Hibernate

Hibernate ist ein Open-Source-Persistenz-Framework für Java.

Hibernate ist ein Framework, das es ermöglicht, den Zustand eines Objekts in einer relationalen Datenbank zu speichern und aus entsprechenden Datensätzen wiederum Objekte zu erzeugen (OR-Mapping). Beziehungen zwischen Objekten werden auf entsprechende Datenbank-Relationen abgebildet.

MySQL

MySQL ist ein SQL-Datenbankverwaltungssystem der schwedischen Firma MySQL AB. MySQL ist als Open-Source-Software für verschiedene Betriebssysteme verfügbar und bietet damit die Grundlage vieler Webauftritte. Neben den meisten Unix-Varianten, Mac OS X und Linux läuft MySQL auch auf Windows und OS/2 (nur Version 3.x) und vielen weiteren Betriebssystemen. MySQL wird sehr häufig zusammen mit dem Webserver Apache und PHP eingesetzt.

Struts

Warum Struts hilfreich ist? Trennung von Dialogen und Geschäftslogik (Funktionalität). Manche Leuten entwickeln eine Webanwendung mit Scriptsprachen wie Perl oder PHP und integrieren Ihre SQL Abfragen und Geschäftslogik direkt in das HTML Dokument. So kann man natürlich auch entwickeln, wenn man Java Servlets oder JSP benutzt. Sinnvoll ist das nur in Miniprojekten. Stellen wir uns vor, man hat über 70 Dialoge, sehr viele Datenbankabfragen in allen Dialogen und man möchte jetzt mit einem Statusfeld definieren, ob ein Buch gelöscht ist oder nicht. Aus diesem Grund griffen wir auf das MVC-Pattern zurück. Auf der Suche nach einem geeigneten Framework, stießen wir auf Cocoon, Struts und Spring. Diese Frameworks kannte bis zu diesem Zeitpunkt keiner von uns im Detail mit all ihren Vorteilen und Nachteilen. Da Struts realtiv bekannt ist, entschlossen wir uns dieses Framework für Spamgourmet einzusetzen. Vorteile der Trennung:

- Änderungen in der Funktionalität, ohne in den Dialogen arbeiten zu müssen.
- Bessere Überschaubarkeit, Funktionalität nicht vermischt mit Dialogen.
- Anwedungen lassen sich besser Warten.
- Unterschiedliche Dialoge, aber gleiche Funktionalität.

Man unterscheidet bei Struts zwischen drei Teilen

- Model (Geschäftslogik / Geschäftsprozesse - Java Beans)
- View (Dialoge - JavaServer Pages)
- Controller (Zentrale Steuereinheit - Java Servlets)

Javadocs

Javadoc ist ein Software-Dokumentationswerkzeug, das aus Java-Quelltexten Dokumentationsdateien im HTML-Format generiert. Für zusätzliche Informationen können spezielle Kommentare im Quelltext eingefügt werden. Kommentare können dabei für Interfaces, Klassen, Methoden und Felder über spezielle Doclet-Tags definiert werden. Javadoc wurde ebenso wie Java von Sun Microsystems entwickelt und ist Bestandteil des Java 2 Software Development Kits (J2SDK). Bsp. http://java.sun.com/j2se/1.5.0/docs/api/

```
/**
 * Ein HelloWorld-Programm in Java.
 * Dies ist ein Javadoc-Kommentar,
 * der die Klasse beschreibt.
 * @author Donald Duck
 * @version 1.0
 */
public class HelloWorld {
    /**
     * Die main()-Funktion.
     * @param args Kommandozeilenparameter
     */
    public static void main(String[] args) {
        System.out.println("Hallo, Welt!");
    }
}
```

Checkstyle

Checkstyle erlaubt es dem Java-Programmierer u.a., auch größere Mengen an Quellcode schnell auf die Einhaltung bestimmter Coding Conventions zu überprüfen. Hierzu steht eine Vielzahl unterschiedlicher Prüfoptionen (Checks) zur Verfügung, die in einer XML-Datei definiert sind. Wesentliche Strukturmerkmale sind dabei sogenannte Module und Properties. Über Module werden die gewünschten Checks eingebunden und über Properties die Arbeitsweise bzw. die Parameter eines Moduls festgelegt. Die XML-Datei wird schließlich beim Aufruf des Tools als Konsolenanwendung oder als ANT-Task integriert. Durch das Setzen spezieller Schalter können zudem mehrere Java-Dateien oder Dateien in Unterverzeichnissen einer Prüfung unterzogen werden. Der generierte zum Teil deutschsprachige Fehlerbericht lässt sich als Text und/oder im XML-Format ausgeben. Dies ermöglicht eine Darstellung desselbigen in jedem Browser, der über einen XML-Parser verfügt. Checkstyle kann ferner als Plugin für Java-Entwicklungsumgebungen fungieren (z.B. Eclipse, JEdit, JBuilder), was die Fehlersuche erleichtert und die Einsatzmöglichkeiten erhöht. Eine Besonderheit stellt zudem die Erweiterbarkeit dar: So lassen sich mit Hilfe von selbst definierten Checks beispielsweise Coding Standards erstellen. Hierzu steht dem Entwickler mit dem Checkstyle SDK Gui eine graphische Oberfläche zur Verfügung. Die Nutzung des Tools erfordert eine installierte Java-Entwicklungsumgebung sowie grundlegende XML-Kenntnisse.

ANT

Ant ist ein überaus hilfreiches Werkzeug, das im Rahmen des Projekts zum Übersetzen der Quelltexte sowie zum Testen der Quelltextqualität eingesetzt werden kann (und sollte). Normalerweise muss man als Java-Programmierer darauf achten, dass alle Verzeichnisse von zu übersetzenden oder einfach nur benutzten Klassen vom javac auffindbar sind. So ist etwa daran zu denken, dass alle verwendeten jar-Dateien explizit im CLASSPATH aufgeführt werden, da andernfalls die im jar enthaltenen Klassen nicht gefunden werden können. Zusätzlich ist eine Übersetzung von Klassen in der Regel nur im aktuellen Verzeichnis gängig (und teilweise auch nur dort möglich), so dass bei einer sauberen Aufteilung der Quelltexte auf verschiedene Verzeichnisse jedesmal in ein anderes Verzeichnis gewechselt werden muss. Um diesen Prozess zu vereinfachen, schreiben Unix-Experten in der Regel sogenannte makefiles, die (grob gesprochen) durch Aufruf von make ziel die Befehle ausführen, die für die Erstellung von ziel erforderlich sind. Dummerweise hat make eine etwas kryptische Notation und ist zudem an einigen Stellen sehr pingelig. Zu allem Überfluß ist make zwar standardmäßig unter Unix-/Linux-Umgebungen installiert, aber nicht unter Windows. An dieser Stelle setzt ant mit einem etwas moderneren Ansatz als make an. Ant ist in Java implementiert und daher im Prinzip überall lauffähig. Gleichzeitig basiert ant auf XML, was eine sehr einfache Einbindung auch in Produktionssysteme erlaubt sowie die Nutzung eines etablierten Standards bietet.

Sonstiges

was es sonst zu sagen gibt...

Links & Quellen

- **Temporäre Emailadressen**
  - **Spamgourmet**

    http://www.spamgourmet.com/

    http://sourceforge.net/projects/spamgourmet
  - **Use spam.la email addresses for throw-away site registrations.**

    http://www.spam.la
- **Database**
  - **MySQL**

    http://www.mysql.com
  - **Hibernate**

    http://www.hibernate.org
  - **Hibernate API.** Javadoc Dokumentation der Hibernate Klassen.

    http://www.hibernate.org/hib_docs/v3/api/
  - **Hibernate Gesamt-Doku.** Vollständige Dokumentation zu Hibernate auf einer Seite (englisch).

    http://www.hibernate.org/hib_docs/reference/en/html_single
  - **DBDesigner von fabForce**

    http://fabforce.net/dbdesigner4/
  - **XAMPP** ist eine Distribution von Apache, MySQL, PHP und Perl, die es ermöglicht diese Programme auf sehr einfache Weise zu installieren.

    http://www.apachefriends.org/de/xampp.html
- **STRUTS Framework**
  - **Struts Tutorial**

    http://www.torsten-horn.de/techdocs/jsp-struts.htm
  - **Struts Documentation**

    http://struts.apache.org/1.2.9/userGuide/index.html
  - **Struts Tutorial (with Tiles)**

    http://www.laliluna.de/tutorial/first-tiles/first_struts_tiles_tutorial_de.pdf
- **Eclipse Plugins**
  - **Checkstyle - Source Code Checker** Prüft den Sourcecode.
    - Eclipse-CS - Eclipse Checkstyle Plugin

http://eclipse-cs.sourceforge.net/update

- CheckClipse - Eclipse Checkstyle Plugin

  http://sourceforge.net/projects/checkclipse

- **SQL Explorer** ist ein Plugin zum komfortablen Zugriff auf eine Datenbank.

  http://sourceforge.net/projects/eclipsesql

- **Subclipse** is an Eclipse plugin that adds Subversion integration to the Eclipse IDE.

  http://subclipse.tigris.org/

- **Versionierung**
  - **Das Subversion Buch**

    http://svnbook.red-bean.com/

  - **TortoiseSVN**

    http://tortoisesvn.tigris.org/

- **JAVA Mail and JNDI**
  - **Java Enterprise Mail Server (a.k.a. Apache James)**

    http://james.apache.org/

  - **Java Email Server (JES)**

    http://www.ericdaugherty.com/java/mailserver/index.html

  - **Java and JNDI/LDAP**

    http://java.sun.com/products/jndi/tutorial/trailmap.html

  - **Java Mail Beispiel**

    http://www.torsten-horn.de/techdocs/java-smtp.htm

  - **Java Mail API**

    http://java.sun.com/products/javamail

  - **E-Mail-Header lesen und verstehen**

    http://www.th-h.de/faq/headerfaq.php

  - **SMTP reply codes**

    http://www.supermailer.de/smtp_reply_codes.htm

  - **RFC 2821 - Simple Mail Transfer Protocol**

    http://www.ietf.org/rfc/rfc2821.txt

Teil II. Handbuch

Inhaltsverzeichnis

Spamgourmet Handbuch

Zusammenfassung

Das Spamgourmet Handbuch beschreibt die Spamgourmet Applikation und deren Verwendung. Hier werden die Bedienung der Weboberfläche und ihre Einstellmöglichkeiten erklärt. Es dient als Benutzerhandbuch zur Verwendung der Spamgourmet Applikation und ist hilfreich beim Verstehen, aus welchen Gründen die festgelegten Vereinbarungen und Definitionen so definiert wurden.

Handbuch

Einführung

Der Anbieter http://www.spamgourmet.com bietet die Möglichkeit zur Definition temporärer E-Mailadressen, welche mit einer nur dem Benutzer bekannten, persönlichen E-Mail Adresse verbunden sind. Auf diese Weise kann man beispielsweise für die License Key Zusendung einer Software Teststellung anstelle der eigenen E-Mailadresse eine solche Temporäradresse verwenden und auf diese Weise die spätere Zusendung von Werbemüll vermeiden.

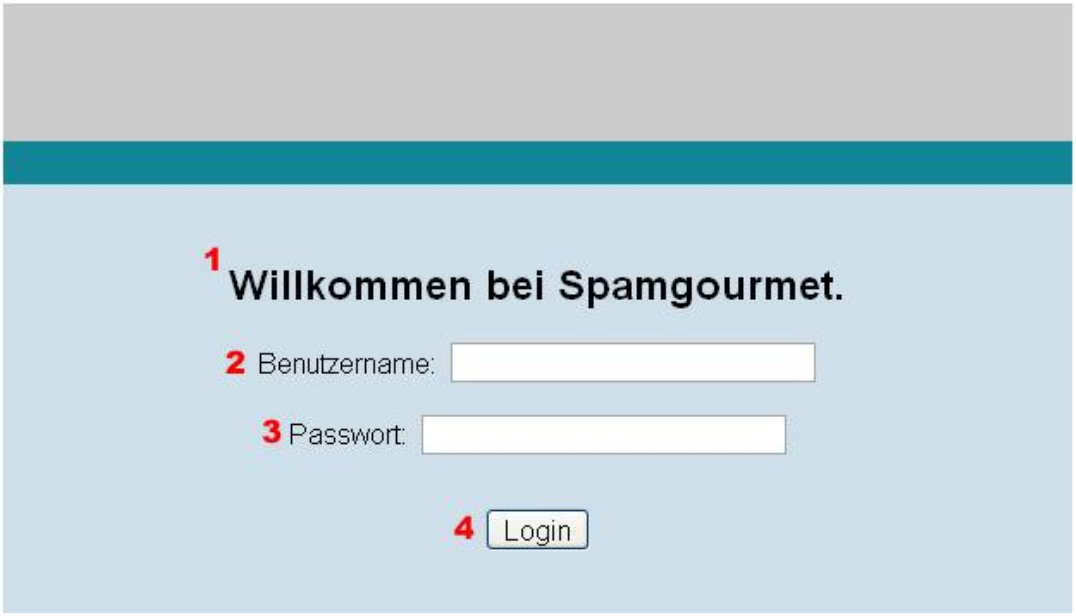
Beim Surfen im Internet stößt man häufig auf Webdienste, die es erfordern sich zu registrieren. Unter anderem wird meist eine Emailadresse verlangt, an die ein Registrierungs-Passwort geschickt wird. Oftmals werden die Daten hierbei offensichtlich nicht als vertraulich behandelt. Nach kurzer Zeit wird das Emailpostfach mit Newslettern des Dienstes oder gar fremden Werbemails befüllt. Eigentlich wollte der Benutzer doch nur einem kostenlosen Forum beitreten, um eine Frage zu stellen. Nun bezahlt er die Rechnung damit, sein Postfach von diesem Ärger zu befreien. Das festlegen der Regeln der angebotenen Spamfilter der Anbieter des Emailaccounts oder des Emailprogrammes scheint kein Ende zu nehmen.

Ein weiterer Lösungsansatz als diese endlosen Regeldefinitionen besteht in der Möglichkeit, temporäre Emailadressen zu vergeben. Die eigentliche Adresse bleibt hier im Hintergrund und sollte nicht weitergegeben werden. Die Emails werden an einen Empfängeralias gesendet, der von einem Webservice in die „richtige“ Adresse übersetzt werden muss, bevor er an diese weitergeleitet wird. Dabei können für jede Adresse Regeln festgelegt werden, die bestimmen, ob diese Email angenommen werden darf, oder nicht. Auf den ersten Blick erscheint es, als ob wir nichts gewonnen haben. Wieder Regeln definieren. Der Vorteil besteht aber darin, dass alle Emails, die aufgrund einer einzigen Registrierung versendet werden (sei es von dem Webseitenbetreiber selbst oder von einem Dritten, an den die Adresse weitergegeben wurde), nur durch das Deaktivieren dieser Adresse ausgefiltert werden. Die Handhabung ist dabei sehr einfach und unkompliziert.

In den folgenden Kapiteln wird die Anwendung des Spamgourmet-Services beschrieben. Dabei steht die Weboberfläche und ihre Einstellungsmöglichkeiten im Vordergrund.

Login

Für die SpamGourmet Anwendung ist ein Login notwendig, um eine Benutzerverwaltung (bzw. das Auseinanderhalten der verschiedenen Benutzer) zu ermöglichen. Dieser erfolgt über eine Login Maske wie folgt: Benutzername und Passwort eingeben und abschicken.



1. Auf der Login-Seite kann sich der Benutzer bei der Spamgourmet Applikation anmelden.
2. Geben Sie in das Username-Eingabefeld ihren Spamgourmet-Benutzernamen ein, mit dem Sie sich registriert haben.
3. Geben Sie in das Password-Eingabefeld ihr HdM-Passwort ein.
4. Klicken Sie den Login-Button, nachdem Sie Ihre Benutzerdaten eingegeben haben, um sich anzumelden.

Erster Login

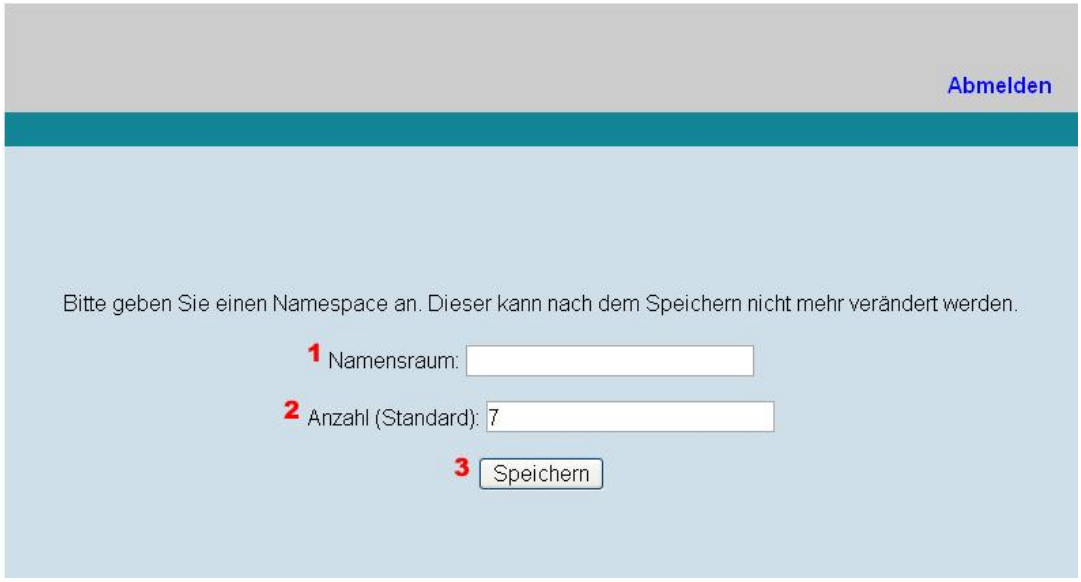
Beim ersten Login muss ein Namespace definiert werden. Zusätzlich kann noch ein Default Count für die Number Rules gesetzt werden.

Der Namespace dient der Unterscheidung der verschiedenen Benutzer. Jeder Namespace wird nur einmal vergeben. Der Vorteil dabei ist, dass jeder Benutzer jede ihm beliebige Adresse anlegen kann.

Wählt beispielsweise ein Benutzer den Namespace „ab“ und ein anderer „cd“, so kann jeder eine Adresse mit seinem Vornamen „Peter“ anlegen. Die beiden vollständigen Adressen lauten dann:

peter.ab@temp.hdm-stuttgart.de
peter.cd@temp.hdm-stuttgart.de

Obwohl beide Adressen mit dem gleichen Namen angelegt wurden können sie trotzdem durch die Verwendung von Namespaces voneinander unterschieden werden.



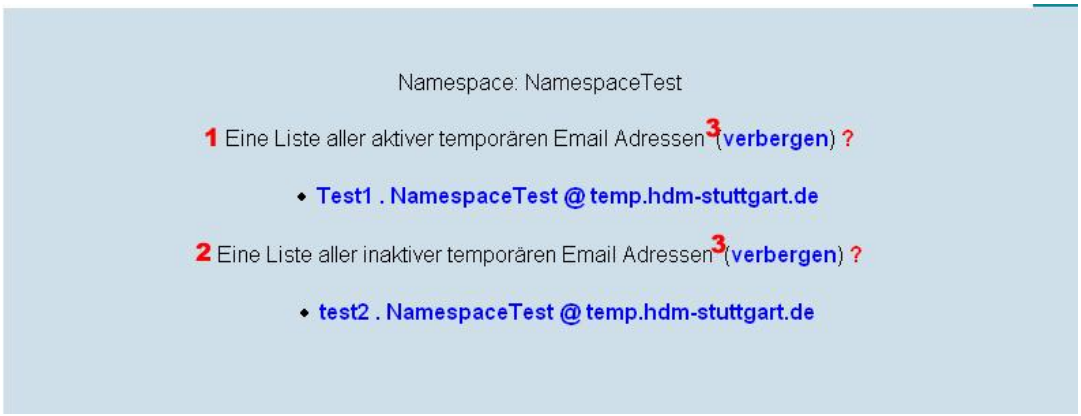
1. Geben Sie in das Namespace-Eingabefeld ihren gewünschten Namespace ein. Innerhalb dieses Namespaces können Sie temporäre Adressen frei nach Belieben anlegen. ACHTUNG: Der Namespace kann später nicht mehr verändert werden.

2. Geben Sie in das DefaultCount-Eingabefeld die Anzahl der Emails an, die sie maximal empfangen wollen. Bei Neuanlegen einer temporären Adresse wird diese Anzahl automatisch festgelegt.

3. Klicken Sie die Speichern-Schaltfläche, nachdem Sie Ihren gewünschten Namespace und die maximale Anzahl erwünschter Emails eingegeben haben, um die Daten zu sichern.

Übersicht

Die Übersichtsseite gibt einen Überblick aller angelegter TempEmail Adressen. Dabei wird unterschieden zwischen aktiven und inaktiven Adressen. Als aktive Adressen werden diejenigen Adressen aufgeführt, welche über Regeln verfügen die noch nicht abgelaufen sind. Abgelaufen heißt zum Beispiel an Hand einer TimeRule: das aktuelle Datum ist entweder vor dem StartDate oder nach dem EndDate. Oder bei der NumberRule: es wurden bereits mehr Emails empfangen als die Regel zulässt.



1. Diese Liste enthält alle aktiven temporären Email Adressen. Emails welche an diese Adressen gesendet und von den jeweiligen Regeln durchgelassen werden, werden empfangen.

2. Diese Liste enthält alle inaktiven temporären Email Adressen. Diese Adressen wurden entweder manuell deaktiviert oder deren Regeln sind abgelaufen.

3. Mit einem Klick auf "verbergen" kann die Liste der temporären Adressen versteckt, d.h. zusammengeklappt werden. Durch einen Klick auf "anzeigen" wird die Liste der temporären Adressen wieder angezeigt.

Head Navigation

Die Head Navigation ist die Hauptnavigationsmöglichkeit. Von hier aus können die einzelnen Grundfunktionen eingestellt werden.

In der Head Navigation kann zwischen dem Basic- und dem Extendedmodus gewechselt werden. Desweiteren gelangt man von hier aus zu den Einstellungen. Falls man sich im Extendedmodus befindet taucht hier noch der Navigationspunkt "Neue TempEmail" auf um manuell eine neue temporäre Emailadresse zu erstellen.

1. Klicken Sie auf Basic um in den Standardmodus zu wechseln.

   Klicken Sie auf Extended um in den Expertenmodus zu wechseln.

2. Klicken Sie auf Einstellungen um auf die Einstellungsseite zu kommen.

3. Klicken Sie auf Neue TempEmail um eine neue temporäre Adresse anzulegen.

4. Klicken Sie auf Abmelden um sich von der Weboberfläche abzumelden.

Content Navigation

Die Content Navigation dient der Übersicht. Sie gibt an, in welcher Einstellungsebene sich der Benutzer gerade befindet. Alle übergeordneten Ebenen können direkt über diese Leiste angesprungen werden.

Mit Hilfe dieser Navigationsleiste kann zu jeder Zeit zwischen der Übersichtsseite und der gerade bearbeiteten temporären Emailadresse navigiert werden.

1. Auf der Übersichtsseite werden alle angelegten TempEmail Adressen angezeigt. Über diesen Link gelangen sie zu jeder Zeit dorthin.

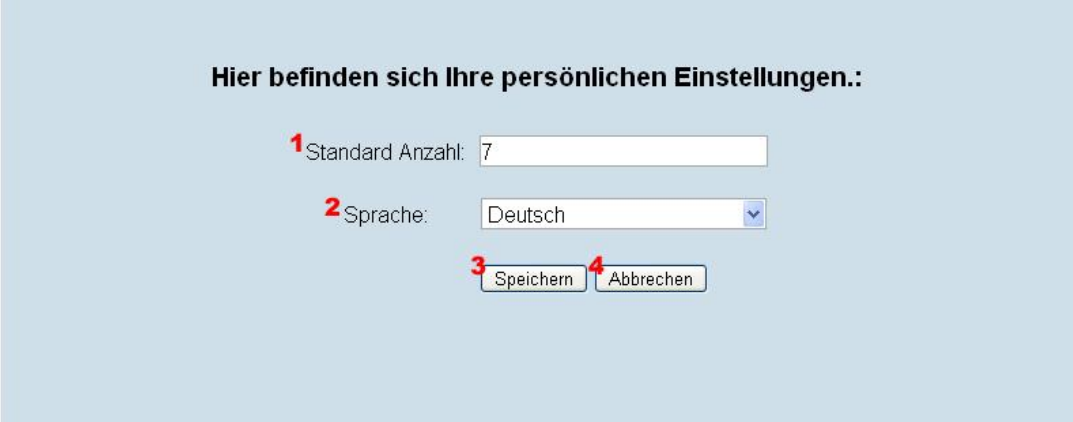
2. Über diesen Link gelangen sie zu der aktuell bearbeiteten temporären Email.

3. Die vollständige Leiste zeigt die Ebenen an, in der sich der Benutzer gerade befindet.

Einstellungen

Auch das Ziel dieses Projektes unter anderem die einfache Handhabung ist: ganz ohne Einstellungen geht es nicht. Auf der Einstellungen-Seite können die notwendigsten Angaben für den Service eingegeben werden.

Das ist zum einen die bevorzugte Sprache des Benutzers, die auf der Weboberfläche verwendet wird und zum anderen die Standardanzahl der Emails, die beim Neuanlegen einer neuen temporären Empfängeradresse akzeptiert werden.



1. Diese Zahl definiert die Standardanzahl der Number Rule. Automatisch angelegte temporäre Emailadressen werden mit dieser Anzahl der zu empfangenden Emails angelegt.

2. Hier können Sie die bevorzugte Sprache der Weboberfläche einstellen.

3. Klicken Sie die Speichern-Schaltfläche, nachdem Sie Ihren gewünschten Namespace und die maximale Anzahl erwünschter Emails eingegeben haben, um die Daten zu sichern.

4. Klicken Sie auf Abbrechen um den Vorgang abzubrechen.

Erstellen einer neue temporären Adresse

Im Expertenmodus können beliebig viele temporäre Emailadressen von Hand angelegt werden. Dazu muss erst der Name der temporären Adresse definiert werden. Ist dies erledigt können Empfangsregeln festgelegt werden.

1. Geben Sie in das Name-Eingabefeld einen Namen für die neue Emailadresse ein.

2. Klicken Sie auf Hinzufügen um die neue temporäre Adresse anzulegen.

1. Der Name gibt an, welche Emailadresse gerade bearbeitet wird.
2. Klicken Sie auf die Schaltfläche aktivieren bzw. deaktivieren um diese Adresse zu aktivieren bzw. deaktivieren.
3. Legen Sie fest, ob die Regeln mit und oder oder verknüpft werden. Bei einer Verknüpfung mit "und" müssen alle Regeln zutreffen damit die Email weitergeleitet wird. Bei einer Verknüpfung mit oder muss nur eine Regel zutreffen damit die Email weitergeleitet wird.
4. Wählen Sie im Listenfeld eine Regel aus, die sie gerne anlegen würden. Klicken Sie anschliessend auf anlegen um eine neue Regel zu erstellen.

   Die Regeln legen fest wer, wann, wieviele Emails und von wem empfangen darf.
5. Klicken Sie auf löschen um die ausgewählte temporäre Emailadresse zu löschen.

Anlegen von Regeln

Der Empfang durch die temporären Emailadressen kann durch drei verschiedene Regeln gesteuert werden.

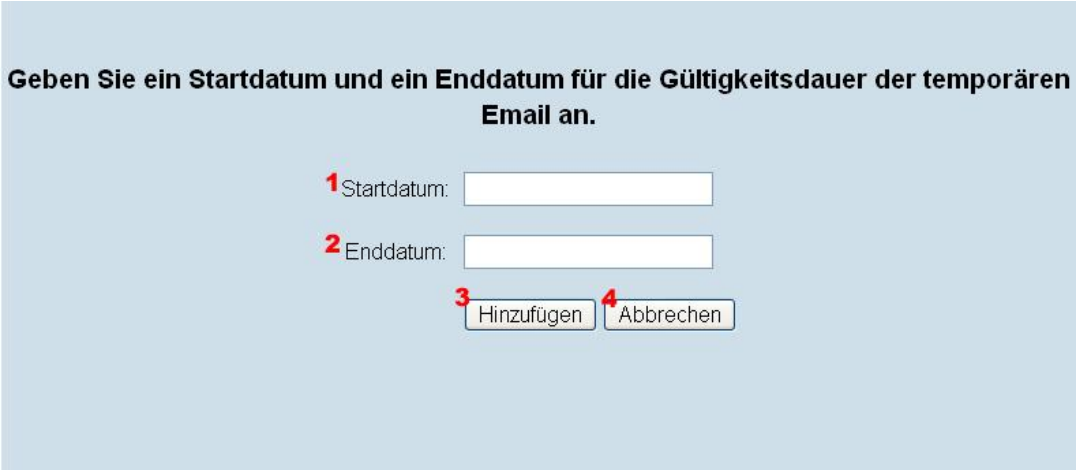
- Number Rule
- Time Rule
- Domain Rule

Number Rule

Die Number Rule definiert eine Anzahl von Emails die empfangen werden sollen. Wurde die definierte Anzahl von Emails empfangen, dann werden alle weiteren an diese Adresse gesendeten Emails verworfen.

Time Rule

Mit der Time Rule wird ermöglicht das eine temporäre Emailadresse nur zu einer bestimmten Zeit gültig ist. Dazu muss ein Startdatum und ein Enddatum definiert werden.



1. Klicken Sie auf das Eingabefeld Startdatum und wählen Sie am Kalender ihren Zeitpunkt aus. Wenn Sie JavaScript deaktiviert haben, schreiben Sie das Datum in der Form: 03-07-2006 21:20 (Deutsche Einstellung), bzw. 2006-07-04 21:23 (Englische Einstellung) in das Textfeld.

2. Klicken Sie auf das Eingabefeld Enddatum und wählen Sie am Kalender ihren Zeitpunkt aus. Wenn Sie JavaScript deaktiviert haben, schreiben Sie das Datum in der Form: 03-07-2006 21:20 (Deutsche Einstellung), bzw. 2006-07-04 21:23 (Englische Einstellung) in das Textfeld.

3. Klicken Sie auf Hinzufügen, bzw Save um die Regel hinzuzufügen.

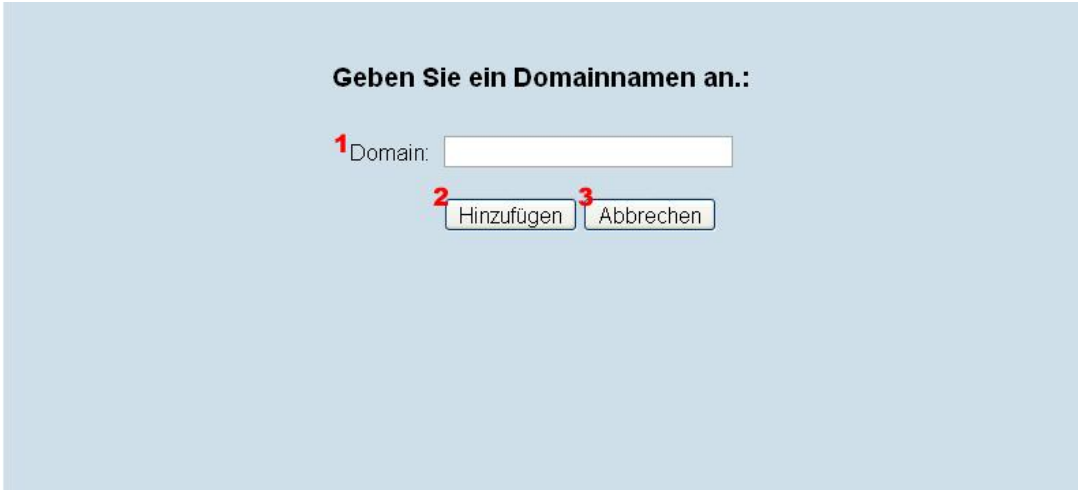
4. Klicken Sie auf Abbrechen um den Vorgang abzubrechen.

1. Mit Hilfe dieses Kalenders, der durch Klicken auf das Textfeld erscheint, kann der Termin komfortabel ausgewählt werden.

Domain Rule

Die Domain Regel erlaubt es die zu empfangenden Emails nach deren Absender-Domains zu filtern. Zum Beispiel ist es möglich nur Emails einer bestimmten Domain zu empfangen

oder umgekehrt alle Emails ausser denen die von der definierten Domain kommen.



1. Geben Sie in das Eingabefeld Domain den Domainnamen, von dem Sie Emails akzeptieren bzw. verweigern wollen, ein.
2. Klicken Sie auf Hinzufügen, bzw. Save um die neu erstellte Domain Regel hinzuzufügen.
3. Klicken Sie auf Abbrechen um den Vorgang abzubrechen.

Teil III. Install Guide

Inhaltsverzeichnis

Installation Guide

Zusammenfassung

Die Installationsanleitung beschreibt den kompletten Installationsvorgang der Spamgourmet Applikation inklusive aller notwendigen Anwendungen unter verschiedenen Betriebssystemen.

Installationsanleitung

Diese Anleitung beschreibt die Vorgehensweise zur Installation und Konfiguration der Spamgourmet Applikation. Da die Anwendung in Java geschrieben ist besteht die Grundvorraussetzung für ein Java Runtime Environment (JRE) beziehungsweise ein J2SE Development Kit (JDK). Die Wahl zwischen diesen hängt vom verwendeten Servlet Container ab. Empfohlen wird allerdings ein JDK, da dies bereits ein JRE enthält und die Wahl des Servlet Containers etwas flexibler gestaltet. Die Anwendung besteht aus einer Webanwendung mit Datenbankanbindung und einem Emailserver, ebenso mit Datenbankanbindung.

Bei dem für die Webanwendung erforderlichen Servlet Container entschieden wir uns für den Apache Tomcat Servlet Container. Dieser ist frei erhältlich und weit verbreitet. Als Datenbank wird die OpenSource Datenbank MySQL verwendet. Nachfolgend wird kurz erklärt wo diese Software erhältlich ist und wie diese installiert werden kann.

Vorbereitungen unter Windows

MySQL installieren

Die Installation der Datenbank kann auf verschiedene Arten durchgeführt werden. Hier werden zwei davon erläutert. Die erste Version verwendet die Distribution von MySQL direkt und die zweite verwendet eine Zusammenstellung verschiedener Software der Apache Friends.

MySQL.com

Die von uns bevorzugte OpenSource Datenbank MySQL ist unter folgender Adresse erhältlich:

- http://dev.mysql.com/downloads/mysql/5.0.html

Diese Anleitung beschreibt die Installation der Datenbank mit Hilfe der MySQL Community Edition. Diese enthält den Datenbank Server und eventuell benötigte Clients.

Nach dem Herunterladen und Entpacken der Zip Datei steht eine Setup.exe zur Verfügung. Nach der Ausführung der Datei erscheint der Installationswizard.

Durch klicken auf „Next“ kommt man zur Auswahl des Setup Types. Hier im Beispiel wurde der Typ „Typical“ verwendet.

Nach der Bestätigung durch „Next“ erhält man eine Übersicht über die ausgewählten Parameter. Hier der Setup Type und das Ziel Verzeichnis in das der Server installiert wird. Nach der weiteren Bestätigung durch Klick auf „Install“ wird der Server installiert.

Nach erfolgreicher Installation wird man dazu aufgefordert sich bei MySQL zu registrieren

oder durch eine bereits erfolgte Registrierung erhaltene Daten anzugeben. Dies ist nicht zwingend notwendig (wie im Beispiel durch „Skip Sign-Up“ angedeutet).

Als Abschluss der Installation erhält man die Möglichkeit den Server umgehend zu konfigurieren. Es sind nur wenige Schritte notwendig die im folgenden beschreiben werden.

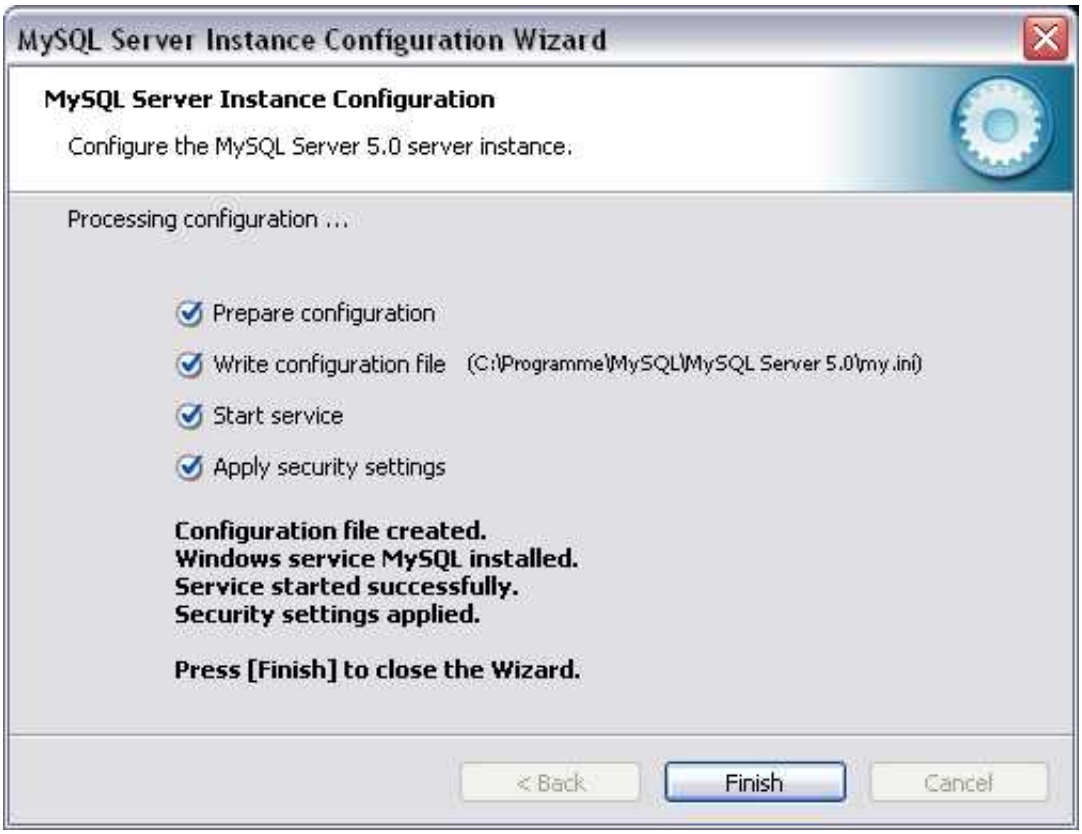
Nach Start der Konfiguration erscheint wieder ein Startbildschirm der durch Klick auf „Next“ bestätigt wird. Als Konfiguration wird empfohlen die „Standard Configuration“ zu wählen.

Als nächstes wird falls der Server als Service laufen soll der Name des Services erfragt und ob der Service automatisch gestartet werden soll. Damit man mit Hilfe der

Kommandozeilen Tools auf den Server zugreifen kann empfielt es sich den Haken bei „Include Bin Directory in Windows PATH“ zu setzen. Diese Kommando Zeilen Tools werden später verwendet um die Datenbank zu initialisieren.

Nun muss noch ein Passwort für den Superuser der Datenbank Instanz definiert werden. Dieses Passwort wird etwas später in dieser Anleitung zum Initialisieren der Spamgourmet Datenbank noch gebraucht.

Nach dem Klick auf „Next“ erscheint der Dialog zum Konfigurieren und Starten des Servers. Nach der Aktivierung durch Klick auf „Execute“ wird die Konfiguration geschrieben und der Datenbank Server mit der neuen Konfiguration gestartet. Ein abschließendes „Finish“ beendet den Wizard.



Nun ist der Server fertig eingerichtet. Es kann nun mit der Initialisierung der Spamgourmet Datenbank fortgesetzt werden. Dies erfolgt mit Hilfe des Kommandozeilen Clients. Dieser ist durch den Shortcut im erstellten Startmenü Eintrag „ MySQL - MySQL Server 5.0 - MySQL Command Line Client “ erreichbar. Nach Aufruf dessen wird man nach dem Superuser Passwort gefragt das vorhin definiert wurde. Nach der erfolgreichen Eingabe dessen müssen folgende SQL Statements abgesetzt werden um die Datenbank und einen speziellen User mit dem auf diese zugegriffen wird, anzulegen.

```
CREATE DATABASE `spamgourmet`;
CREATE USER 'spamgourmet'@'localhost' identified BY 'spamgourmet';
GRANT ALL PRIVILEGES ON spamgourmet.* TO 'spamgourmet'@'localhost';
```

Die Anforderungen an die Datenbank sind nun erfüllt.

XAMPP

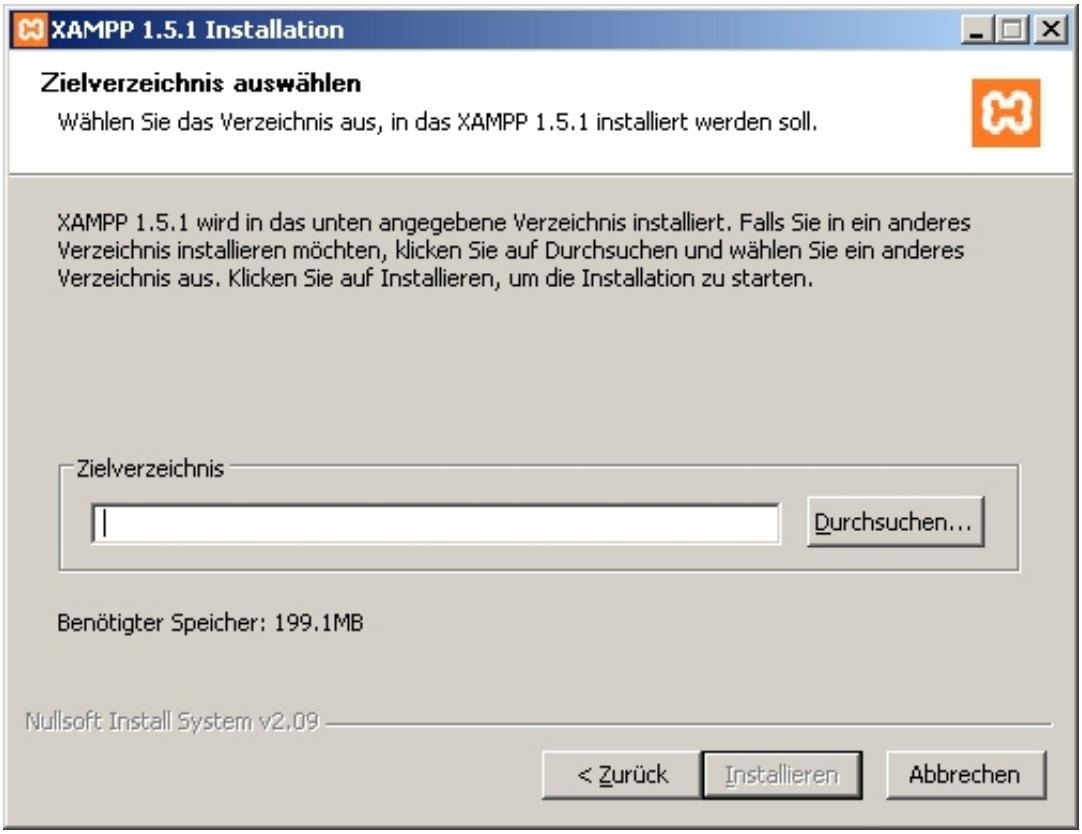
Als Alternative zur Installation der MySQL-Datenbank existiert ein sehr komfortables und einfaches Tool „XAMPP“ . Es ist unter folgender Adresse erhältlich:

- http://www.apachefriends.org/de/xampp.html

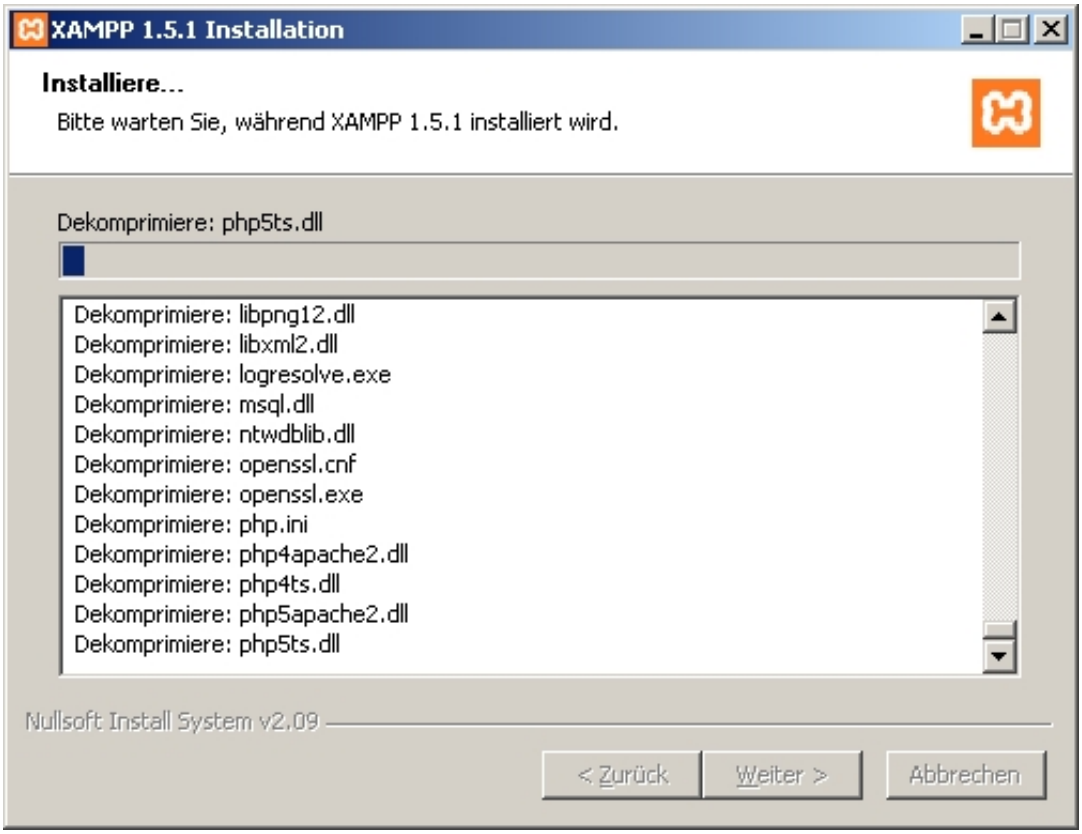
Das erste Fenster nach dem Starten des Installationsvorgangs durch Ausführen der heruntergeladenen exe-Datei von der XAMPP-Webseite fragt nach der bevorzugten Sprache. In dieser Anleitung wird „Deutsch“ ausgewählt.

Das Begrüssungsfenster erscheint.

Nach Klicken auf „Weiter“ fragt der Installer nach dem Zielverzeichnis. Mit Hilfe der Schaltfläche „Durchsuchen“ wird ein Dateisystembrowser gestartet, mit dem das Verzeichnis ausgewählt und gegebenenfalls auch ein neues angelegt werden kann.



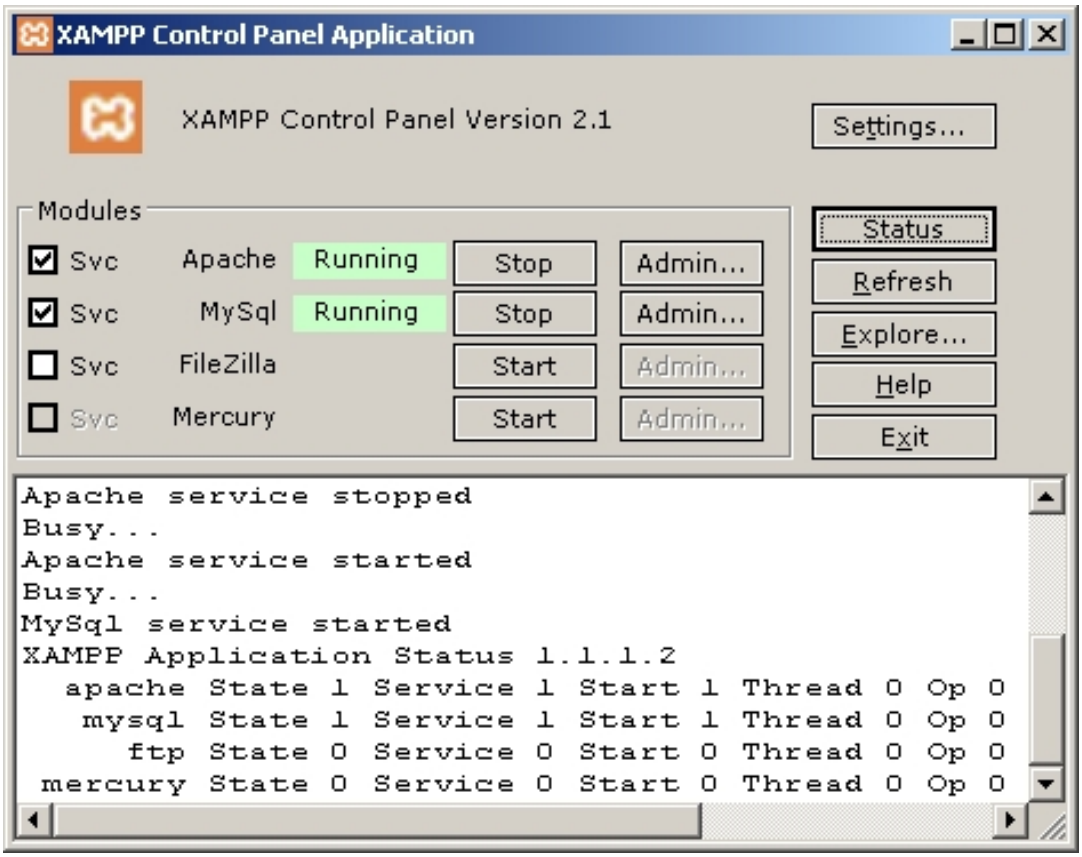
Danach startet der Kopiervorgang.



Bei der Installation als Hintergrunddienst laufen die Anwendungen ständig im Hintergrund. Auf die Frage, ob die Anwendungen als Dienst installiert werden sollen, wurde in dieser Anleitung mit „Nein“ geantwortet. Das hat den Grund, dass sich diese Installationsanleitung auf eine Entwickler-Workstation bezieht, auf der die Dienste nicht konstant laufen müssen und sollen. Sollte die Spamgourmet-Applikation bzw. die Datenbank auf einem Windows-Server installiert werden ist es ratsam die Datenbank als Dienst zu installieren.

Die Installation war erfolgreich.

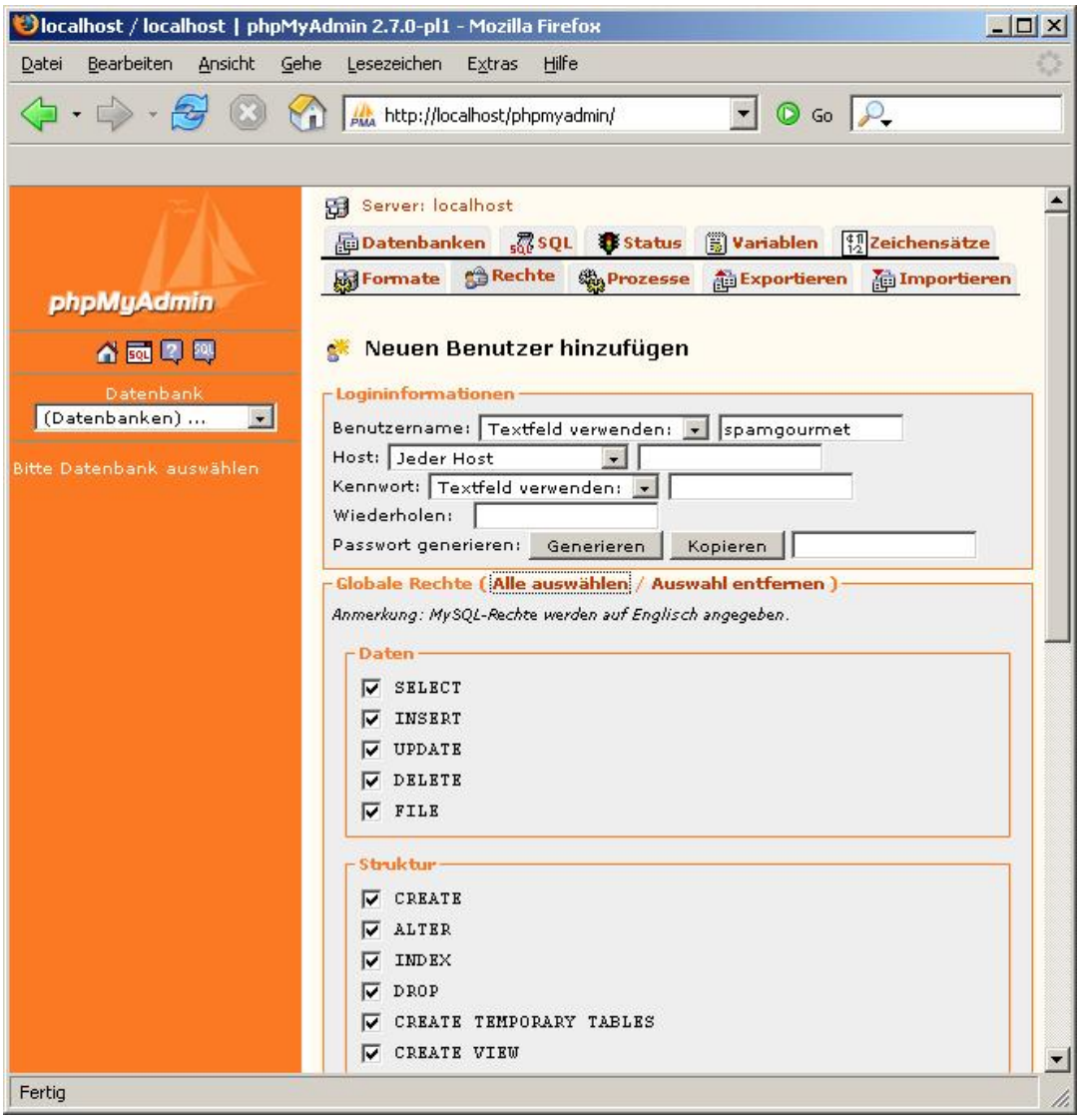
Über das Control Panel können die Server gestartet und gestoppt werden.



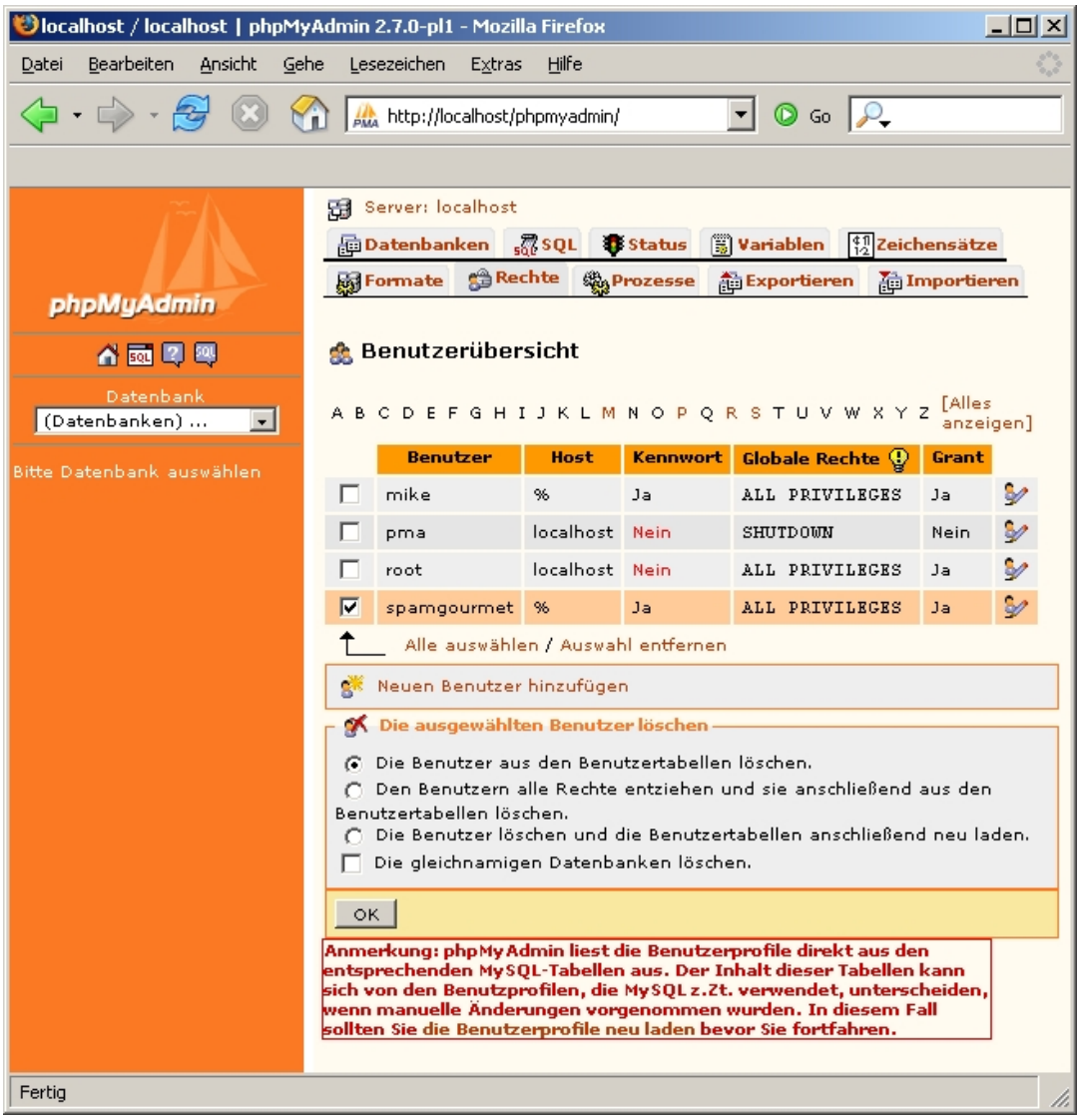
Durch Klicken auf den Apache Admin - Button wird die Konfigurationsapplikation im Webbrowser gestartet.

Die Oberfläche von phpMyAdmin kann über den gleichnamigen Link in der Navigationsleiste errreicht werden.

Zuerst muss ein neuer Benutzer „spamgourmet“ angelegt werden. Der selbe Benutzername wird bei Tomcat ebenfalls angelegt um den Zugriff der Weboberfläche auf den Datenbankserver zu ermöglichen.



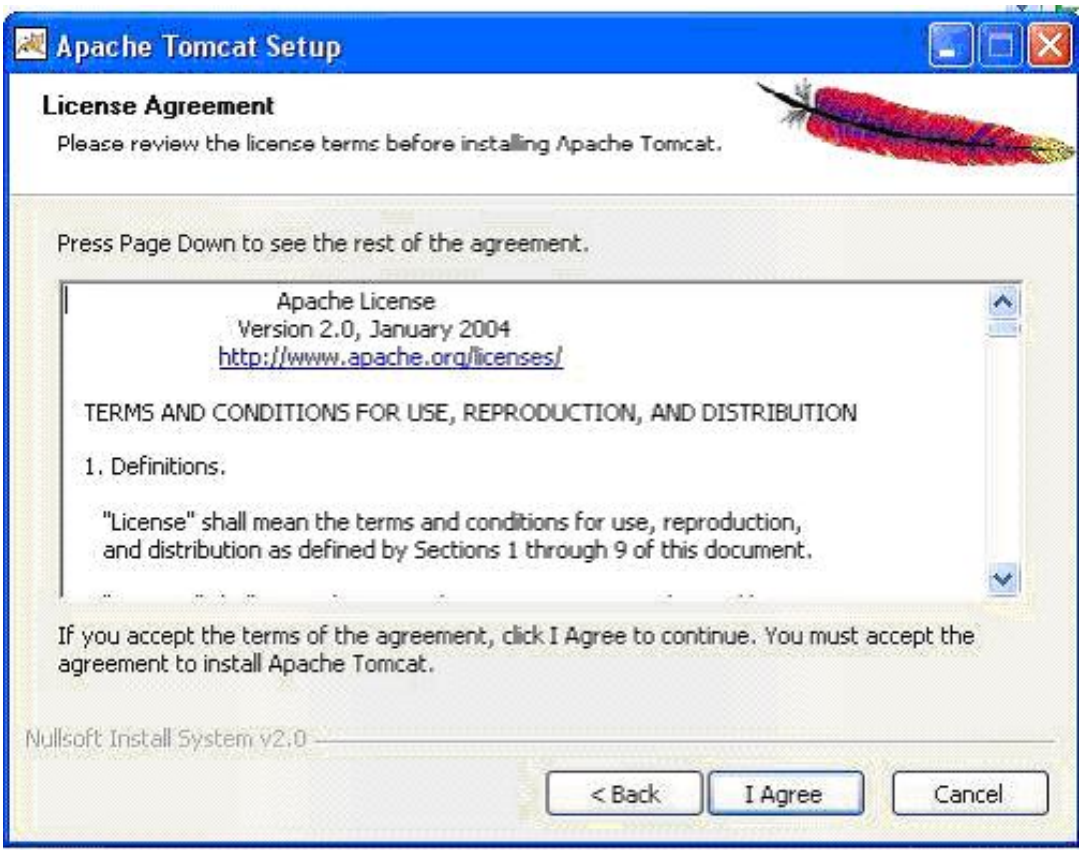
Der Benutzer bekommt alle Rechte.



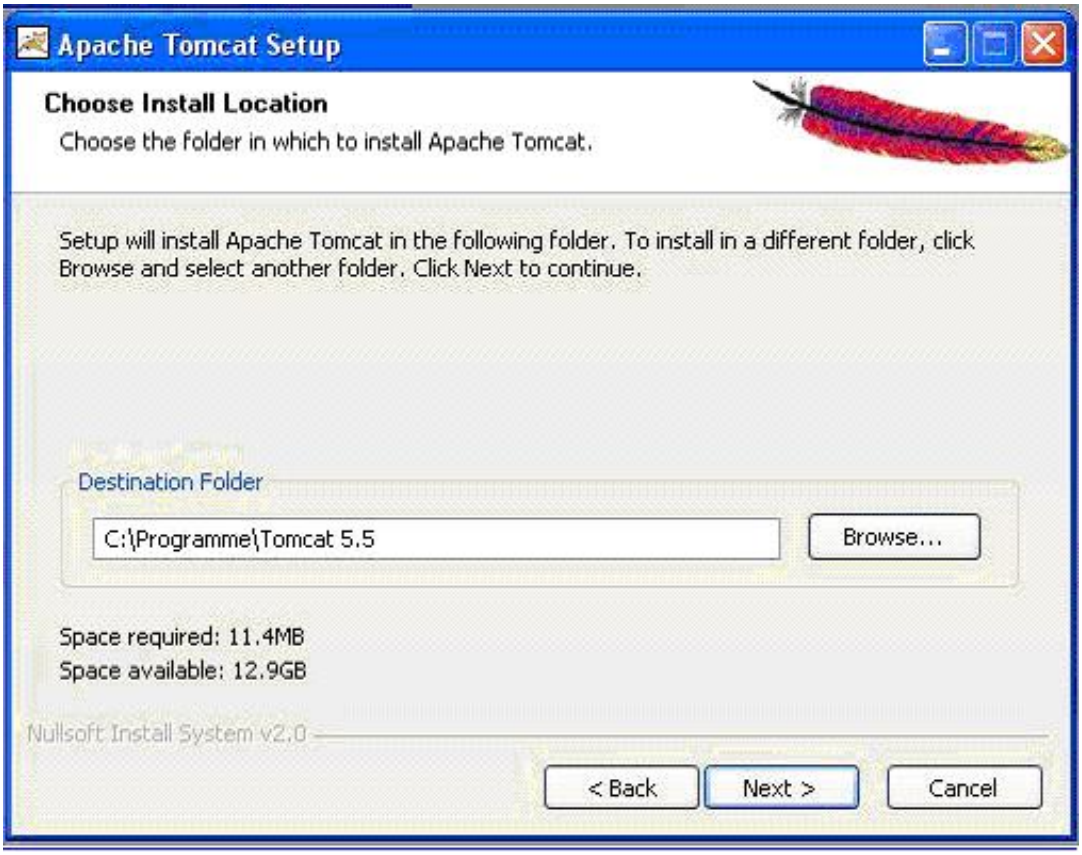
Tomcat installieren

Die Download-Dateien der neusten Versionen sind auf der Apache-Webseite unter dem Link http://tomcat.apache.org/download-55.cgi zu finden.

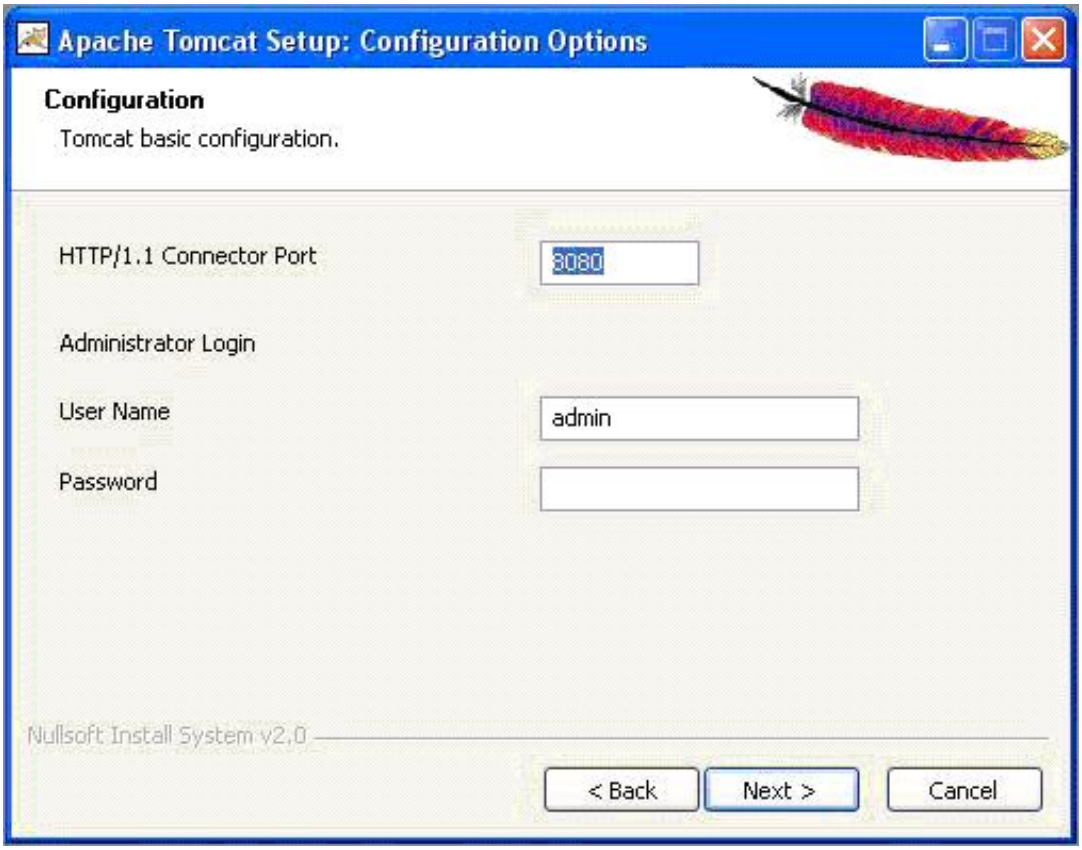
Zu Beginn der Installation wird zunächst um das Einverständnis der Lizenzvereinbarungen gebeten, bevor man die Installationseinstellungen vornehmen kann.



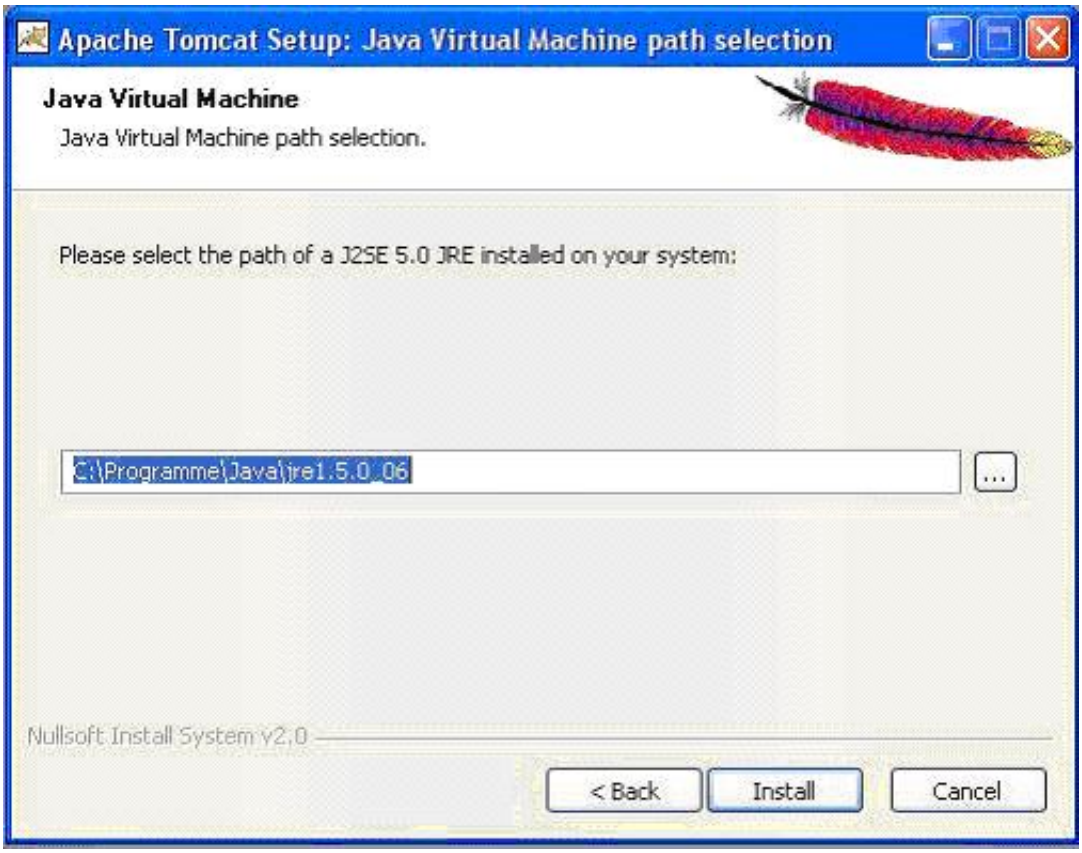
Im zweiten Schritt ist der Pfad zu wählen, in dem der Tomcat installiert werden soll. In diesem Fall „C:\Programme\Tomcat 5.5“.



Als nächstes muss der Port angegeben, unter dem der Webserver erreichbar sein soll. Zudem ist das Administrator Passwort zu vergeben. Wird nichts angegeben, bzw der „next“ Button betätigt, ist das default-Passwort: „password“.



Im letzen Schritt ist der Pfad anzugeben, in dem die „Java-Laufzeitumgebung“ installiert ist.



Daraufhin ist die Grundkonfiguration des Servers abgeschlossen. Nach der Installation kann über einen Browser die Adresse `http://localhost:8080/` abgefragt werden, um die Startseite auf dem neu installierten Server einzusehen.

Das unter `http://localhost:8080/` einsehbare Verzeichnis nennt sich webapp Verzeichnis. Standardmäßig ist es im lokalen Dateisystem im Tomcat-Verzeichnis zu finden. Dieses Verzeichnis wäre dann nach unserer Beispielinstallation an der folgenden Position zu finden:

„C:\Programme\Tomcat 5.5\webapps“

In der Datei <TOMCAT_HOME>/conf/tomcat-users.xml müssen die folgenden zwei Zeilen hinzugefügt werden.

```
<role rolename="spamgourmet"/>
<user username="spamgourmet" password="spamgourmet"
      roles="spamgourmet"/>
```

Damit wird ein neuer Benutzer „spamgourmet“ angelegt. Er wird der ebenfalls neu erzeugten Rolle (Gruppe) hinzugefügt.

Im Verzeichnis <TOMCAT_HOME>conf\Catalina\localhost\ muss eine neue Datei mit dem Namen der Anwendung angelegt werden. Wir nennen sie „spamgourmet.xml“. Darin wird eine Zeile ans Ende eingefügt:

```
<Context docBase="C:\swt1Spamgourmet\trunk\app" path="spamgourmet"/>
```

Die Spamgourmet-Applikation liegt in diesem Beispiel im Verzeichnis „C:\swt1Spamgourmet\trunk\app“

Email Server installieren

Wie wird der Email Server unter Windows installiert?

Vorbereitungen unter Linux

MySQL installieren

Unter Linux gehört die MySQL Datenbank oftmals bereits zur Grundausstattung. Sollte dies nicht der Fall sein kann die Datenbank meist bequem über den verwendeten Paketmanager installiert werden.

Beispiele hierfür wären:

- YUM für die Linux-Distributionen Fedora Linux und Yellow Dog Linux
  - http://fedora.redhat.com/
  - http://linux.duke.edu/projects/yum/
- Advanced Packaging Tool (APT) für deb oder RPM basierte Linux-Distributionen
  - http://www.debian.org/doc/manuals/apt-howto/
- User RedHat Package Manager von Mandriva Linux, RPM-basiert
  - http://www.mandriva.com/
  - http://easyurpmi.zarb.org/
- YaST Yet Another Setup Tool von SUSE Linux
  - http://www.suse.com/

Eine weitere Möglichkeit besteht darin ein fertig gepacktes RPM direkt von MySQL herunterzuladen und dies dann manuell mit rpm zu installieren. Oder aber man besorgt sich die Quellen von MySQL und compiliert die Binaries selbst.

Unter folgender Adresse können die RPMs bzw. die Quellen heruntergeladen werden:

- http://dev.mysql.com/downloads/

Unter diesen Adressen steht ein HowTo das beschreibt wie man dabei vorgeht. Für die Installation per rpm Package:

- http://dev.mysql.com/doc/refman/5.0/en/linux-rpm.html

Und für die Installation mit Hilfe der Quellen:

- http://dev.mysql.com/doc/refman/5.0/en/installing-source.html

Tomcat installieren

Der Servlet Container Jakarta Tomcat gehört ebenso wie die MySQL Datenbank unter

Linux zu den Dingen die sehr einfach installiert werden können da auch dieser in den Distributionen eventuell schon vorhanden oder per Paket Manager auf einfache Weise nachinstalliert werden kann. Hierbei ist sogar zu empfehlen dies über diese Möglichkeit zu erledigen da dabei die startup Scripts gleich mitgeliefert werden. Somit kann der Tomcat Application Server als Daemon laufen und muss nicht bei jedem Reboot neu gestartet werden.

Um den Tomcat Server der Versionen 5.x erfolgreich zu betreiben ist mindestens ein Java Runtime Environment (JRE) notwendig. Bis Jakarta Tomcat 4.x ist mindestens ein J2SE Software Development Kit (SDK) notwendig. Es wird allerdings empfohlen auf die neueren Versionen des Servlet Containers zurückzugreifen (> 5.0.x) und ein J2SE Development Kit (JDK) der Version 5.0 zu verwenden. Die Webanwendung wurde erfolgreich mit den Versionen 5.5.x des Servlet Containers getestet.

Eine weitere Möglichkeit besteht darin sich eine „Binary Distribution“ zu besorgen und diese zu verwenden. Diese ist unter folgender Adresse erhältlich:

- http://tomcat.apache.org/index.html

Für eine ausfühliche Beschreibung des Setup des Apache Tomcat 5.5.x Servlet Container (diese Version des Servlet Containers wird empfohlen) findet man unter folgender URL eine Anleitung des Entwicklers:

- http://tomcat.apache.org/tomcat-5.5-doc/setup.html

Auf den Seiten von Apache Tomcat findet man ebenfalls ein Setup des Apache Tomcat 5.0.x Servlet Container:

- http://tomcat.apache.org/tomcat-5.0-doc/setup.html

Unter dieser Adresse findet man ein HowTo, wie man den Jakarta Tomcat 4.1.x Servlet Container installiert (dieser wird allerdings nicht empfohlen):

- http://tomcat.apache.org/tomcat-4.1-doc/RUNNING.txt

Email Server installieren

Wie wird der Email Server unter Linux installiert?

Spamgourmet Web-Applikation installieren

An dieser Stelle wird vorausgesetzt, dass eine MySQL Datenbank, der Apache Tomcat und der Emailserver James bereits installiert sind. Die Datenbank muss nicht unbedingt auf dem selben Host laufen. Es reicht auch eine Verbindung zu einer entfernten Datenbank.

Build

Über das ANT Script kann ein *Web Archiv* erzeugt werden. Ein Web Archiv (WAR-Datei) ist eine Datei im JAR- bzw. ZIP-Format, die eine vollständige Webanwendung enthält. Darin enthalten sind die bereits vorkompilierten Java Server Pages, die Servlets, statische Files, etc.

Um die WAR-Datei zu erzeugen muss das Target *all* der *build.xml* ausgeführt werden. Das

Ergebnis steht dann im Verzeichnis

{SPAMGOURMET_HOME}/app/dist/

„SPAMGOURMET_HOME“ steht dabei für das Basis Verzeichnis der Spamgourmet Quellen.

Installation

Es gibt mehrere Möglichkeiten, die Spamgourmet Applikation zu installieren. Zwei davon werden nachfolgend beschrieben.

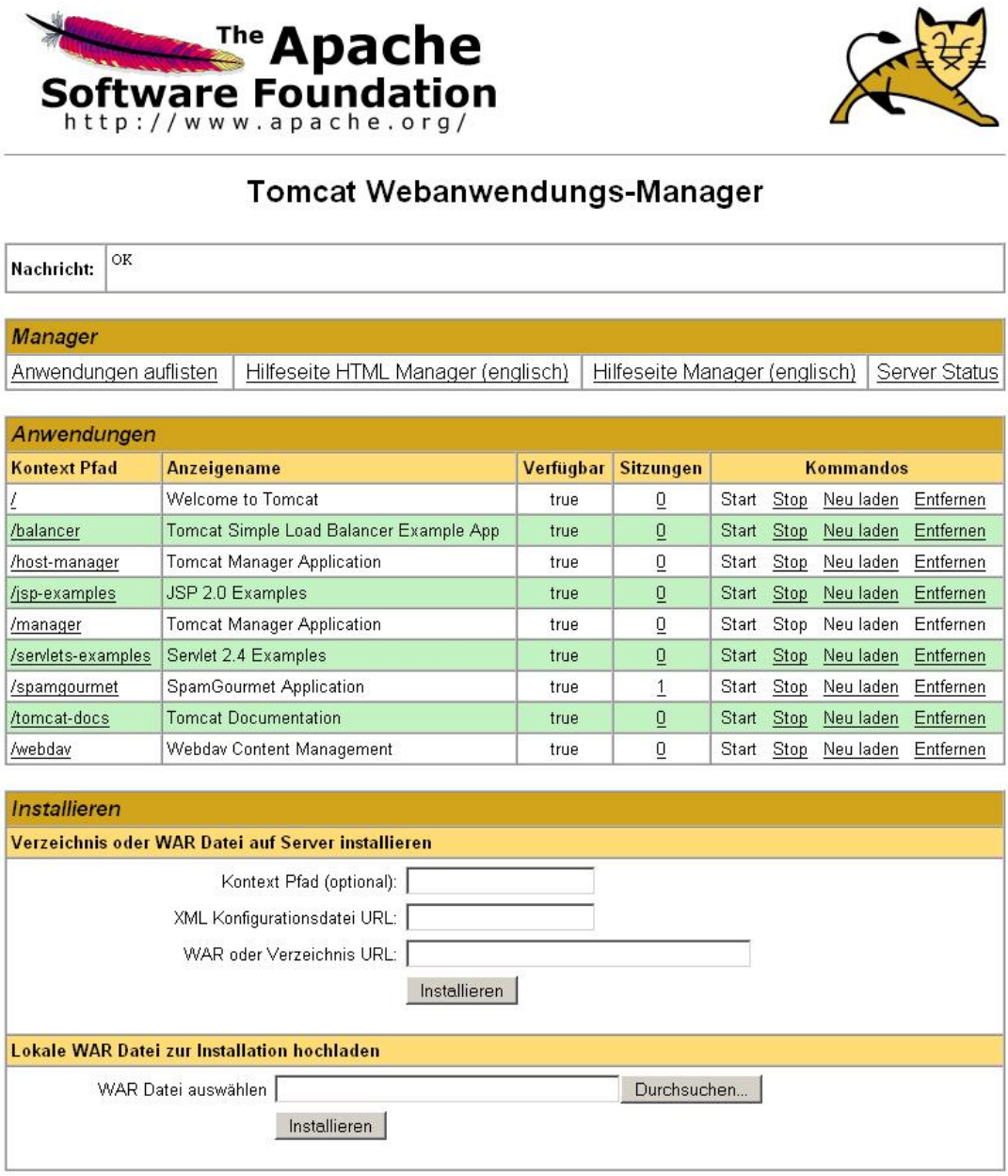
Die WAR Datei kann einerseits einfach in das Verzeichnis der Webanwendungen des Tomcats kopiert werden.

{TOMCAT_HOME}/webapps/

Der Name des Contextes wird dem Dateinamen entnommen. Heisst die Datei zum Beispiel spamgourmet.war ist diese unter <host>/spamgourmet erreichbar.

An Stelle von „TOMCAT_HOME“ steht der Pfade zum jeweiligen Tomcat Installations Verzeichnis.

Die zweite Möglichkeit besteht darin, über die Tomcat-Manager-Oberfläche einen neuen Kontext zu installieren. Die Einstellungen können über den Browser vorgenommen werden.



Im unteren Bereich der Oberfläche kann über den „Durchsuchen“-Button ein Dateisystem-Browser gestartet werden. Damit kann komfortabel zur WAR-Datei navigiert werden. Nachdem sie ausgewählt ist wird der Installationsvorgang über den „Installieren“-Button gestartet.

Konfiguration der Anwendung

Datenbankverbindung

Die Anwendung erfordert eine Datenbank in welcher Daten wie Benutzerdaten und Temporären Emailadressen gespeichert werden. Wie bereits erwähnt wird als Datenbankserver ein MySQL Server angenommen. Auf diesem Server ist nun eine Datenbank zu erstellen und ein User der auf diese zugreifen darf. Wir gehen nun weiter davon aus das der Datenbankserver auf dem localhost läuft. Ansonsten muss anstelle von localhost der Host der Datenbank angegeben werden. Mit Hilfe des Kommandozeilen Clients ist es nun möglich die Datenbank zu initialisieren. Unter Linux wird er z.B. folgendermaßen gerufen:

```
mysql -h localhost -u username -p
```

Dieser Aufruf startet den Kommandozeilen Client als Benutzer „username“ und fordert auf zur Eingabe eines Passwortes. Nun können die SQL Statements abgesetzt werden:

```
CREATE DATABASE `spamgourmet`;
GRANT ALL PRIVILEGES ON spamgourmet.* TO 'username'@'localhost' IDENTIFIED BY '
```

Das erste Statement erstellt die Datenbank mit Namen „spamgourmet“ und das zweite Statement erstellt einen Benutzer mit Namen „username“ der sich durch das Passwort „password“ identifiziert und gibt diesem im gleichen Schritt alle Rechte auf der Datenbank „spamgourmet“.

Die selben Einstellungen müssen in der Konfigurationsdatei für Hibernate ebenfalls vorgenommen werden. Die Datei ist in WEB-INF/classes unter dem Namen „hibernate.cfg.xml“ zu finden.

```
connection.url = jdbc:mysql://localhost:3306/spamgourmet
connection.username = username
connection.password = password
```

Falls keine MySQL-Datenbank verwendet wird kann die Klasse des Datenbanktreibers angegeben und der entsprechende Dialekt verwendet werden. Der Wert in *pool_size* gibt an, wieviele Sessions zur Datenbank konstant geöffnet bleiben sollen.

```
connection.driver_class = com.mysql.jdbc.Driver
dialect = org.hibernate.dialect.MySQLDialect
connection.pool_size = 50
```

Webanwendung

Für die Webanwendung existiert eine separate Konfigurationsdatei mit der Bezeichnung „Application.properties“. Diese liegt ebenfalls in WEB-INF/classes.

```
# Konfiguration
tempmail.domain = temp.hdm-stuttgart.de
namespace.separator = .
numberrule.defaultcount = 7
numberrule.maxcount = 120
```

- *tempemail.domain* gibt den Domainnamen an, unter der der Emailserver läuft.
- *namespace.separator* definiert das Zeichen, das zur Abtrennung des Namespaces vom Rest der temporären Emailadresse verwendet wird.
- *numberrule.defaultcount* ist Anzahl an Emails, die von neuangelegten temporären Adressen angenommen werden dürfen.

```
# Validierung
namespace.validate.not.regex = ^[a-zA-Z]{2}[0-9]{3,4}$
tempemail.validate.regex = ^[a-zA-Z0-9._%-]{5,}$
```

- *namespace.validate.not.regex* ist ein regulärer Ausdruck, der festlegt, welche Form die Namespaces haben sollen. in diesem Fall wird geprüft das der Namespace nicht die Form eines Kürzels der HdM hat (z.B. yx001).
- *tempemail.validate.regex* ist ein regulärer Ausdruck, der auf den Hostanteil der temporären Emailadresse zutreffen muß. Im Beispiel wird auf in einer Email Adresse gültige Zeichen geprüft und ob der Host Anteil länger als 5 Zeichen ist.

```
# Handbuch
manual.base.url = http://localhost/spamgourmet_docu/handbuch/manual
```

- *manual.base.url* gibt an unter welcher URL das Handbuch zur Anwendung erreichbar ist.