

Praktikum aus Softwareentwicklung 2		WS 2008 / 09
Übung 3: JDBC und Networking		
Abzugeben schriftlich: alle Klassen elektronisch: Projekt		Abgabetermin: 27. 10. 2008, 17:00
Team	Name 1:	Matr.Nr. 1:
	Name 2:	Matr.Nr. 1:
	SVN Repository:	
Tutor:		Punkte:

Mail Server

In dieser Übung soll ein Mail-Server implementiert werden, der SMTP-Nachrichten empfangen kann und diese in einer Datenbank verwaltet.

Die Datenbank enthalte 2 Tabellen, eine Tabelle `Users` für die Verwaltung der Benutzer und eine Tabelle `Mails` für die Speicherung der Mails der registrierten Benutzer. Die Tabellen sind folgend aufgebaut (siehe auch Abbildung 1)

Users:

- id: automatisch generierter Primärschlüssel
- email: Mail-Adresse (eindeutig für alle Users)
- password: Passwort (oder besser Digest des Passworts, siehe Hinweise)

Mails:

- id: automatisch generierter Primärschlüssel
- userId: Fremdschlüssel des Users
- fromAdr: Adresse des Senders
- toAdr: Adresse des Empfängers (gleich der Email des Users mit UserId)
- mail: Inhalt der Mail

Users

id	email	password
1	franz@gmx.at	dlkfahselkf
2	hugo@gmail.com	adfaskfhaö

Mails

id	userId	fromAdr	toAdr	mail
1	2	foo@gmx.at	hugo@gmail.com	...

Abbildung 1: Tabellen Users und Mails

Die SQL-Kommandos für die Erzeugung der Tabellen in der Datenbank sind wie folgt (siehe Download von der Homepage):

```
CREATE TABLE Users (
  id INTEGER NOT NULL PRIMARY KEY GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
  email VARCHAR(320) NOT NULL UNIQUE,
  password VARCHAR(32)
);
```

```

CREATE TABLE Mails (
  id INTEGER NOT NULL PRIMARY KEY GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
  userId INTEGER NOT NULL,
  fromAdr VARCHAR(320),
  toAdr VARCHAR(320),
  mail CLOB,
  FOREIGN KEY (userId) REFERENCES Users (id) ON DELETE CASCADE ON UPDATE RESTRICT
);

```

Komponenten MailDatabase:

Realisieren Sie eine Zugriffskomponente für die Datenbank (MailDatabase). Diese soll folgende Operationen unterstützen:

- Benutzerverwaltung:
 - Anfügen eines neuen Benutzers mit Email-Adresse und Passwort
 - Löschen eines Benutzers mit bestimmter Email-Adresse
 - Zugriff auf Benutzer mit bestimmter Email-Adresse
 - Authentifikation eines Benutzer mit bestimmter Email-Adresse über Passwort
- Mails
 - Speichern einer Mail für einen Benutzer
 - Laden aller Mails für einen Benutzer
 - Laden einer Mail mit einer bestimmten Id
 - Löschen einer Mail

Implementieren Sie des Weiteren einen Ereignismechanismus (MailListener, MailEvent), mit dem Änderungen in der Datenbank gemeldet werden.

Komponente DatabaseClient

Implementieren Sie eine einfache Konsolenanwendung, mit der die Datenbank getestet werden kann. die Konsolenanwendung soll folgendes unterstützen:

- neue Benutzer anlegen
- Benutzer löschen
- alle Benutzer auf der Konsole ausgeben
- Passwort eines Benutzers prüfen
- Mails sichern
- Mails ausgeben
- Mails löschen

Komponente SmtServer

Implementieren Sie dann eine Komponente SmtServer, die auf Port 25 auf eingehende SMTP-Requests von Clients horcht, Mails empfangen kann und diese dann für registrierte Empfänger in der Datenbank speichert. Es sollen gleichzeitig mehrere Clients bedient werden können, d.h. jeder Request ist in einem eigenen Thread abzuwickeln.

Es sollen folgende SMTP-Kommandos unterstützt werden: HELO, RSET, MAIL, RCPT, DATA, NOOP und QUIT.

Auf die Kommandos, die nicht unterstützt werden, soll mit *not implemented* (502) geantwortet werden.

Komponente SmtClient

Implementieren Sie für Testzwecke einen einfachen SMTP-Client, der Mails an den SmtServer schicken kann. Es reicht eine einfache Konsolenanwendung.

Übung 3, insgesamt 100 Pkte, abzugeben bis 24. 11. 2008:

- MailDatabase 40 Pkte
- DatabaseClient 10 Pkte
- SmtServer 40 Pkte
- SmtClient 10 Pkte

Hinweise:

- Die Wahl der Datenbank steht Ihnen frei. Am einfachsten ist aber die Verwendung der Java-Datenbank Derby.
- Für das SMTP-Protokoll siehe Folien zu Networkung und SMTP-Spezifikation (<http://tools.ietf.org/html/rfc2821>)
- Die Architektur des Systems ist wie in Abbildung 2 gezeigt:

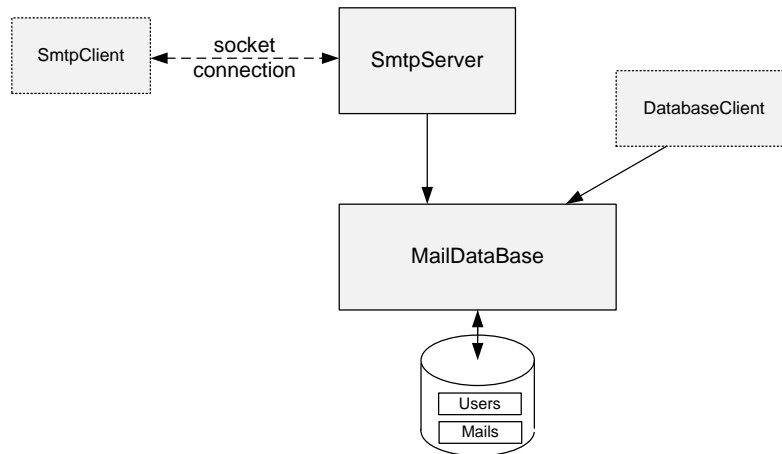


Abbildung 2: Architektur des Systems

- In der Datenbank sollten eigentlich nicht die Passwörter selbst sondern nur ein Digest des Passwortes gespeichert werden. Den Digest kann man mit `MessageDigest` folgend erzeugen:

```
MessageDigest md = MessageDigest.getInstance("MD5");
private String hash(String password) {
    String passHash = "";
    try {
        md.reset();
        md.update(password.getBytes("UTF8"));
        BigInteger bigInt = new BigInteger(md.digest());
        if (bigInt.signum() == -1) {
            bigInt = bigInt.negate();
        }
        passHash = bigInt.toString(16);
    } catch (UnsupportedEncodingException e) {
        Hermes.Log("Missing encoding to has password.", e);
    }
    return passHash;
}
```

Literatur:

SMTP-Specification: <http://tools.ietf.org/html/rfc2821>

Überblick Reply-Codes: <http://www.greenend.org.uk/rjk/2000/05/21/smtp-replies.html>

Derby Database: <http://db.apache.org/derby/>