Prof. Dr. Jörg Hettel

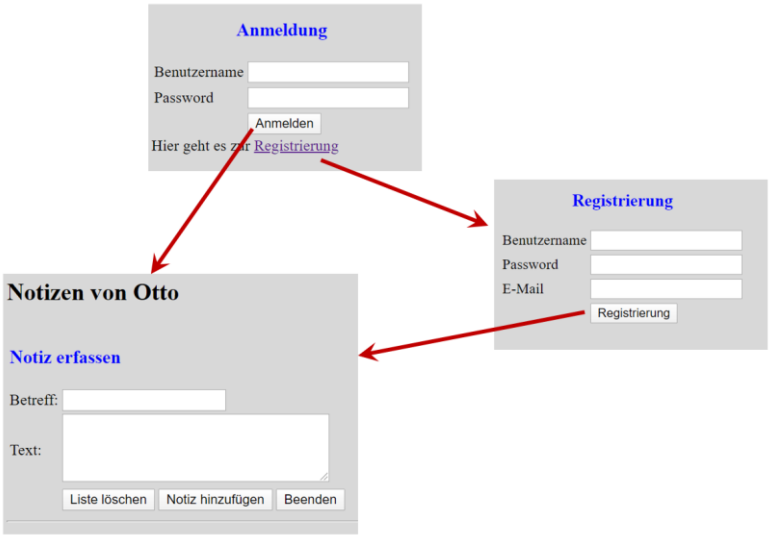
Datenbankprogrammierung
Arbeitspaket 3

Vorbemerkung

In diesem Aufgabenpaket geht es um den Datenbankzugriff innerhalb einer Java-Web-Applikation. Hierzu wechseln Sie am besten in eine Entwicklungsumgebung, die die Entwicklung von Web-Anwendungen erlaubt, wie z.B. die *Eclipse for Java EE Developer* (siehe hierzu auch das Modul Web-Prorammierung).

Auf der Web-Seite finden Sie eine kleine Web-Anwendung für die Verwaltung von Notizen. Laden Sie sich die Anwendung in Ihre Entwicklungsumgebung und machen Sie sich mit der Logik vertraut. Die Anwendung ist lauffähig, wobei eingegebene Daten nur transient gehalten werden.

Benutzer müssen sich zuerst registrieren. Nach erfolgreicher Registrierung wird der Benutzer gleich angemeldet und kann seine Notizen erfassen. Registrierte Benutzer können sich direkt einlaggen.



Aufgabe 1 – Einrichten eines Schemas

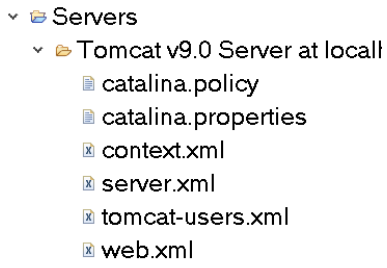
Die Anwendung enthält die Modellklassen **`User`** und **`Note`** und eine Verwaltungsklasse für die **`User`**-Objekte. Legen Sie nun zuerst eine neue Datenbank **`usernotes`** in der mySQL-Datenbank in Ihrem Docker-Container an:

```
DROP DATABASE  IF EXISTS usernotes;

CREATE DATABASE usernotes CHARACTER SET utf8mb4;

USE usernotes;

CREATE TABLE Users (id INT  PRIMARY KEY AUTO_INCREMENT,
                    username VARCHAR(50),
                    password VARCHAR(50),
                    email  VARCHAR(50) );
CREATE TABLE Notes (id INT  PRIMARY KEY AUTO_INCREMENT,
                    subject VARCHAR(100),
                    content VARCHAR(1024),
                    date  VARCHAR(100) ,
                    userId INT  );
```

Datenbankprogrammierung
Arbeitspaket 3

Aufgabe 2 – Einrichten einer DataSource

Vergewissern Sie sich, dass in der Entwicklungsumgebung der Tomcat-Server registriert ist. Der Zugriff auf eine Datenbank sollte in einem Application-Server immer über eine *DataSource* erfolgen. Eine solche *DataSource* muss global beim *Application Server* eingerichtet werden. Bei Tomcat erfolgt dies in entsprechenden xml-Dateien:



Fügen Sie in der **`server.xml`** Datei bei den **`GlobalNamingResources`** den Eintrag hinzu:

```
<Resource auth="Container" driverClassName="com.mysql.cj.jdbc.Driver"
maxActive="10" maxIdle="3" maxWait="10000"
name="jdbc/Notes" type="javax.sql.DataSource"
url="jdbc:mysql://localhost/usernotes?useUnicode=true&amp;useLegacyDatetimeCode=false&amp;useJDBCCompliantTimezoneShift=true&amp;useLegacyDatetimeCode=false&amp;serverTimezone=UTC"
username="root"/>
```

In der **`context.xml`** Datei (entspricht einem lokalen *Name-Service*) muss dann noch folgender Eintrag gemacht werden:

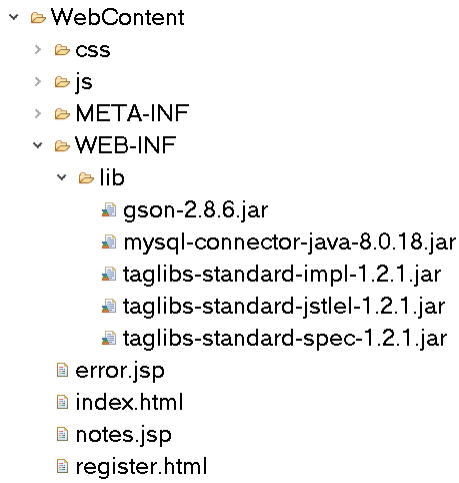
```
<ResourceLink name="jdbc/Notes" global="jdbc/Notes" type="javax.sql.DataSource" />
```

Der Zugriff auf die *DataSource* und die Anforderung einer Connection erfolgt über folgendes Code-Pattern:

```
public class DBConnection
{
   // Singleton Pattern
   private static DBConnection instance = new DBConnection();
   public static DBConnection getInstance() { return instance; }

   private DataSource datasource;

   private DBConnection() {
      super();
      Context ctxt;
      try {
         ctxt = new InitialContext();
         this.datasource = (DataSource) ctxt.lookup("java:comp/env/jdbc/Notes");
      }
      catch (NamingException e) {
         e.printStackTrace();
      }
   }

   public Connection getConnection() throws SQLException  {
         return this.datasource.getConnection();
   }
}
```

```
import java.sql.Connection;
import java.sql.SQLException;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;
```

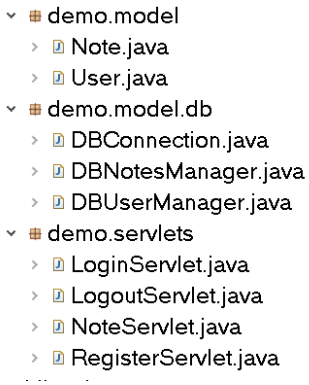
Weiter muss noch der JDBC-Treiber bei Tomcat hinterlegt werden. Da verschiedene Anwendungen, verschiedene Datenbanken nutzen können, empfiehlt es sich, den JDBC-Treiber Anwendungslokal im lib-Verzeichnis der Anwendung zu hinterlegen:



Aufgabe 3 – Entwicklung von Zugriffsklassen und Integration in die Anwendung

Entwickeln Sie entsprechende Zugriffsklassen, die Objekte in die Datenbank speichern bzw. aus der Datenbank restaurieren können. Überlegen Sie sich, wie Sie am besten mit der Beziehung zwischen **`User`** und **`Note`** umgehen. Modifizieren Sie auch die Servlets, so dass jetzt sowohl die registrierten Benutzer als auch deren Notizen in der Datenbank abgelegt werden.

Hier ein Vorschlag für die Paketierung:



Aufgabe 4 – Sichere Speicherung des Passwords

Passwörter sollten nie im Klartext in einer Datenbank abgelegt werden. Ein übliches Verfahren ist, dass von dem Password ein sicherer Hashwert berechnet und dieser in der Datenbank abgelegt wird. Dieses Verfahren hat aber noch den Nachteil, dass gleiche Passwörter immer denselben Hashwert ergeben. Ist somit einmal ein Passwort geknackt, d.h. Password und zugehöriger Hashwert sind bekannt, kann diese Information dazu benutzt werden, bei anderen Angriffen ein Passwort leicht zu identifizieren. Dieser

Angriff ist sehr vielversprechend, da Benutzer oft immer das gleiche Passwort benutzen. (Siehe hierzu auch Attacken mit Hilfe von Rainbow-Tabellen: https://de.wikipedia.org/wiki/Rainbow_Table).

Um solche Attacken zu erschweren (zu versalzen), wird das Hashing „randomisiert“, indem noch ein zufällig erzeugter Wert mit einbezogen wird. So ergibt das Hashing eines Passwords immer einen unterschiedlichen Hashwert. Dieser zufällig erzeugte Wert (salt) muss dann mit dem Passwort zusammen in der Datenbank hinterlegt werden.

Das Schema der User-Tabelle ändert sich somit wie folgt:

```
CREATE TABLE Users (id INT  PRIMARY KEY AUTO_INCREMENT,
                    username VARCHAR(50),
                    pwdhash BLOB,
                    salt BLOB,
                    email  VARCHAR(50) );
```

Die **`User`** Klasse muss nun auch entsprechend angepasst werden. Der BLOB-Datentyp kann auf Byte-Arrays abgebildet werden. Ändern Sie nun die Anwendung, so dass die Passwörter nicht mehr im Klartext in der Datenbank stehen. Beachten Sie hierzu auch die folgenden Code-Patterns.

Die Generierung eines zufälligen Salt-Wertes kann z.B. mit folgender Methode erfolgen:

```
   private static byte[] generateSalt()
   {
      try {
         SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
         byte[] salt = new byte[8];
         random.nextBytes(salt);

         return salt;
      }
      catch (NoSuchAlgorithmException exce) {
         exce.printStackTrace();
         return "42".getBytes();
      }
   }
```

Der Salt-Wert wird dann für die Generierung des Passworts benutzt:

```
   private static byte[] generatePassword(String password, byte[] salt)
   {
      try {
         SecretKeyFactory f = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
         KeySpec ks = new PBEKeySpec(password.toCharArray(), salt, 10000, 160);
         SecretKey s = f.generateSecret(ks);
         Key k = new SecretKeySpec(s.getEncoded(), "AES");
         return k.getEncoded();
      }
      catch (InvalidKeySpecException | NoSuchAlgorithmException exce) {
         exce.printStackTrace();
         return password.getBytes();
      }
   }
```

Bei der Überprüfung eines Passworts, z.B. beim Login, werten Password und Salt aus der Datenbank gelesen. Für das eingegebene (zu überprüfende) Password wird mit Hilfe des für den Benutzer gespeicherten Salt-Werts der Hashwert berechnet und dann mit dem gespeicherten Hashwert verglichen:

```
   public boolean checkPassword(User user, String passwd)
   {
      byte[] pwd = user.getPasswordHash();
      byte[] salt = user.getSalt();

      byte[] pwdToTest = generatePassword(passwd, salt);

      boolean compare = true;

      if (pwd.length != pwdToTest.length)
         return false;

      for (int i = 0; i < pwd.length; i++)
      {
         compare &= (pwd[i] == pwdToTest[i]);
      }

      return compare;
   }
```

Die verwendeten Klassen werden in den folgenden Java-Paketen zur Verfügung gestellt:

```
import java.security.Key;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.KeySpec;


import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;
```