University of Applied Sciences and Arts Northwestern Switzerland
School of Engineering

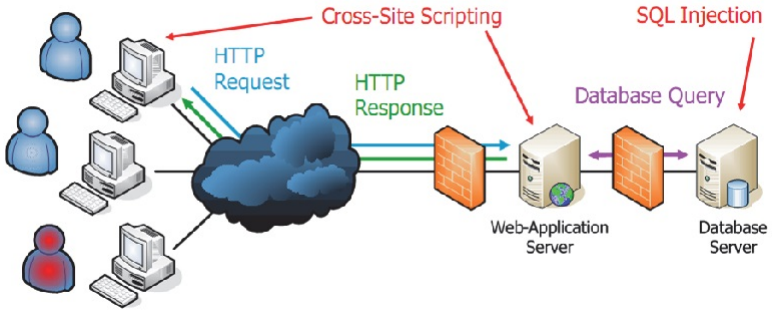
Application Security
Lab 2

Stefan Eggenschwiler & Daniel Gürber

18. Dezember 2013

Inhaltsverzeichnis

1 Aufgabenstellung



2 Softwareaufbau

Die Applikation wurde so programmiert, dass sie auf einem Tomcat lauffähig ist und dabei auf eine MySql Datenbank zugreift. Die Applikation ist aufgeteils in Servlets, Controller, Model, JSP-Pages und MailService.

2.1 Servlets

Die Servlets reagieren jeweils auf ein URL-Pattern und leiten die Anfragen zu den entsprechenden Methoden auf dem Controller.

- IndexServlet: Leitet GET-Anfragen auf /Index auf den Controller weiter, um die Startseite anzuzeigen.
- LoginServlet: Leitet GET- und POST-Anfragen auf /Login auf den Controller weiter um die Login Seite anzuzeigen bzw. die Login Anfrage zu bearbeiten.
- OverviewServlet: Leitet GET- und POST-Anfragen auf /Overview auf den Controller weiter um die Übersichts Seite anzuzeigen bzw. die Passwortänderungs Anfrage zu bearbeiten.
- OverviewServlet: Leitet GET- und POST-Anfragen auf /Register auf den Controller weiter um die Registrierungs Seite anzuzeigen bzw. die Registrierungs Anfrage zu bearbeiten.

2.2 Controller

Die Controller-Klasse kann nicht instanziert werden, da der Konstruktor auf private gesetzt ist, sie bietet statische Methoden an, um eingehende HttpServletRequests zu behandeln. Die einzelnen Methoden analysieren und validieren den Request, rufen entsprechende Methoden auf dem Model auf um Daten zu laden oder zu ändern und zeigen als HttpServletResponse JSP-Pages als View an oder leiten den Benutzer auf eine andere URL um.

2.3 Model

Die Model-Klassen enthalten alle Logik, um das Datenmodel auszulesen, zu validieren und zu ändern.

2.3.1 Comapny

Die Klasse Company hat zwei Anwendungen: Einerseits kann sie mit Feldwerten instanziert werden und diese Werte mit Instanzmethoden zu Validieren und in die Datenbank zu speichern, alle Felder sind final. Andererseits bietet die Klasse auch statische Methoden um Daten zu laden, zu validieren und zu ändern, diese werden verwendet um dem Login durchzuführen, das Passwort zu ändern oder um ein Zitat zu laden.

2.3.2 ConnectionHandler

Diese Klasse kann nicht instanziert werden, sie enthält eine statische Methode um eine Verbindung auf die Datenbank zu erhalten.

2.3.3 Utils

Diese Klasse kann nicht instanziert werden, sie enthält eine statische Methode um eine einen String für Html sicher zu parsen, und würde um weitere allgemeine Methoden erweitert werden.

2.4 JSP-Files

Die JSP-Files stellen Templates dar, die mit Informationen vom Controller gefüllt als Html an den Browser gesendet werden.

2.5 MailService

Diese Klasse kann nicht instanziert werden, sie bietet eine statische Methode um ein E-Mail für die Registrierung einer Firma zu senden.

3 Datenbank

Die Datenbank läuft unter einem MySql-Server.

3.1 Aufbau

Die einzige Tabelle auf der Datenbank ist company, da der username für das Login eindeutig sein muss, ist er als Primärschlüssel geeignet und definiert.

Listing 1: Database Script

```
CREATE TABLE IF NOT EXISTS `company` (
  `username` varchar(64) COLLATE utf8_bin NOT NULL,
  `password` varchar(64) COLLATE utf8_bin NOT NULL,
  `name` varchar(20) COLLATE utf8_bin NOT NULL,
  `address` varchar(255) COLLATE utf8_bin NOT NULL,
  `zip` int(4) NOT NULL,
  `town` varchar(255) COLLATE utf8_bin NOT NULL,
  `mail` varchar(255) COLLATE utf8_bin NOT NULL,
  PRIMARY KEY (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
```

3.2 Sicherheit

Für den Zugriff über die Applikation wurde ein user apsilab02 erstellt, dessen Rechte auf CRUD-Operationen auf der Tabelle company limitiert sind. Für den Root-User wird ein Passwort festgelegt und die Datenbank wird so eingestellt, das nur von localhost darauf zugegriffen werden kann (oder als IP-Limitation wenn die Applikation nicht auf dem selben Server ausgeführt wird).

4 Validierung

Im folgenden gehen wir auf die Validierung der Inputdaten ein. Angegeben werden jeweils die Voraussetzungen aus der Aufgabenstellung und wie wir diese implementiert haben.

4.1 Firmenname

Nur Gross- und Klein-Buchstaben und Leerzeichen (max. 20).

Listing 2: Validierung Firmenname

```
if (name != null) {
  if (name.trim().isEmpty()) {
    errors.add("Firmenname eingeben.");
  } else if (name.trim().length() > 20) {
    errors.add("Firmenname zu lang (max. 20 Zeichen).");
  } else if (!name.matches("([èéÈÉäöüÄÖÜßa-zA-Z\\s]+)")) {
    errors.add("Ung&uuml;ltige Zeichen im Firmennamen");
  }
}
```

4.2 Strasse & Strassennummer

Gross- und Klein-Buchstaben, Zahlen, Punkt, Bindestrich, Leerzeichen.

Listing 3: Validierung Adresse

```
if (address != null) {
  if (address.trim().isEmpty()) {
    errors.add("Keine Adresse.");
  } else if (!address.matches("[èéÈÉäöüÄÖÜß\\w\\s\\.\\-]+")) {
    errors.add("Ung&uuml;ltige Adresse.");
  }
}
```

4.3 Postleitzahl

Nur Zahlen, richtige PLZ für die Schweiz, nachkontrolliert mit einem Web-Dienst wie z. B.: http://www.postleitzahlen.ch.

Listing 4: Validierung PLZ 1

```
if (zip >= 1000 && zip <= 9999) {
  if(!validatePlz(zip))
    errors.add("Ung&uuml;ltige Postleitzahl.");
} else {
  errors.add("Ung&uuml;ltige Postleitzahl.");
  }
```

Listing 5: Validierung PLZ 2

```
/**
 * Validates the zip code with the post.ch service.
 * @param zip zip to validate
 * @return true if correct code
 */
@CheckReturnValue
private static final boolean validatePlz(int zip) {
  URL url;
  HttpURLConnection conn;

  String line;
  try {
    url = new URL("http://www.post.ch/db/owa/pv_plz_pack/
       pr_check_data?p_language=de&p_nap="+zip+"&p_localita=&
       p_cantone=&p_tipo=luogo");
    conn = (HttpURLConnection) url.openConnection();
    conn.setRequestMethod("GET");
    String encoding = conn.getContentEncoding();
    InputStreamReader reader = new InputStreamReader(conn.
       getInputStream(), encoding == null ? "UTF-8" : encoding);
    BufferedReader rd = new BufferedReader(reader);
    try {
      while ((line = rd.readLine()) != null) {
        if(line.contains("Keine PLZ gefunden"))
          return false;
      }
    } finally {
      rd.close();
    }


          return true;
  } catch (IOException e) {
    System.err.println(e.getMessage());
  }
  return false;
}
```

Die Validierung der Postleitzahl führen wir, wie im Code erkennbar, über die Seite der Schweizerischen Post (post.ch) durch. Wir senden einen HTML-Request und parsen das erhaltene XML-Ergebnis. Wir bestimmen dadurch lediglich, ob die angegebene Postleitzahl in der Schweiz existiert oder nicht. Ob diese zur angegebenen Stadt gehört, testen wir nicht.

4.4 Stadt

Erlaubte Zeichen: Gross- und Klein-Buchstaben, Punkt, Bindestrich, Leerzeichen.

Listing 6: Validierung Stadt

```
if (town != null) {
  if (town.trim().isEmpty()) {
    errors.add("Keine Stadt.");
  } else if (!town.matches("[èéÈÉäöüÄÖÜßa-zA-Z\\-\\.\\s]+")) {
    errors.add("Ung&uuml;ltige Stadt.");
  }
}
```

4.5 E-Mail Adresse

Sie senden aber erst, wenn Sie sicher sind, dass die Email-Adresse auch existiert (no bouncing).

Listing 7: Validierung E-Mail

```
if (mail != null && !mail.trim().isEmpty()) {
    if (!mail.matches("//RFC-5322")) {
        errors.add("Ung&uuml;ltige Email-Adresse.");
    } else if(!mxLookup(mail)) {
      errors.add("Ung&uuml;ltige Email-Adresse.");
    }
} else {
        errors.add("Keine Email-Adresse");
}
```

Die Regex-Validierung der E-Mail Adresse führen wir mit dem internationalen Standard der IETF, RFC-822, durch. Weitere Informationen und eine genaue Beschreibung des Standards können unter folgendem Link gefunden werden:
http://www.ietf.org/rfc/rfc0822.txt

Listing 8: Validierung Mail-Server

```
/**
 * Checks if a Mailserver is registered for the specified
 *  email adress.
 * @param mail email adress to check
 * @return true if mailserver is registered
 */
@CheckReturnValue
private static final boolean mxLookup(@Nonnull String mail) {
  String[] temp = mail.split("@");
  String hostname = temp[1];
  Hashtable<String, String> env = new Hashtable<String, String>();

  env.put("java.naming.factory.initial",
      "com.sun.jndi.dns.DnsContextFactory");
  try {
    DirContext ictx = new InitialDirContext(env);
    Attributes attrs = ictx.getAttributes(hostname, new String[] {
        "MX"});
    Attribute attr = attrs.get("MX");
    if (attr == null) {
      return false;
    } else {
      return true;
    }
  } catch (NamingException e) {
    return false;
  }
}
```

Nach einigem Rechercheaufwand haben wir festgestellt, dass es keine Möglichkeit gibt sicher festzustellen, ob eine angegebene E-Mail Adresse wirklich existiert oder nicht. Auch kann es trotz einer existierendem Konto auf dem Mail-Server immernoch zu Bouncing kommen. Wenn beispielsweise das Postfach voll ist, der Benutzer eine Öut of OfficeMessage gesetzt hat oder das Konto aus irgendeinem Grund inaktiv sein sollte.

Deshalb haben wir uns nach einigen Überlegungen dazu entschieden, lediglich abzufragen, ob der Mail-Server, den der Benutzer in seiner E-Mail-Adresse angibt, existiert oder nicht. Diesen Check machen wir über eine MxLookup-Methode.

4.6 Benutzername

min. 4 Zeichen, max. 64 Zeichen; Erlaubte Zeichen: Zahlen, Gross- und Klein-Buchstaben mit Umlauten, Punkte, Bindestriche, Unterstriche.

Eine Validierung des Benutzernamens müssen wir nicht durchführen, da dieser nicht manuell vom Benutzer geändert werden kann. Der Benutzername wird über den Firmennamen generiert. Falls bereits eine Firma mit diesem Namen bestehen sollte, wird eine 1 usw. an den Namen gehängt.

Listing 9: Generierung Benutzername

```
/**
 * Creates a new Username with the name of the company as a base.
 * @return new username
 * @throws SQLException thrown on database error
 */
@CheckReturnValue
private final String createUsername() throws SQLException {
  String usernameBase = name != null ? name.replace(" ", "") : "";
  int tries = 0;

  String newUsername;
  boolean collision;
  try (Connection con = ConnectionHandler.getConnection()) {
    do {
      newUsername = usernameBase + (tries > 0 ? tries : "");
      tries++;

      try (PreparedStatement stm = con.prepareStatement("SELECT `
          username` FROM `company` WHERE `username` = ? ")) {
        stm.setString(1, newUsername);
        collision = stm.executeQuery().next();
      }
    } while (collision);

    return newUsername;
  }
}
```

4.7 Passwort

min. 8 Zeichen, max. 64 Zeichen; Erlaubte Zeichen: Zahlen, Gross- und Klein-Buchstaben mit Umlauten, Punkten, Bindestrichen, Unterstrichen.

Listing 10: Validierung Passwort

```
/**
 * Validates the given password.
 * @param pw password
 * @return true if password is valid
 */
@CheckReturnValue
public static final String validatePassword(@CheckForNull String
    pw) {
  String error = null;
  if (pw != null) {
    if (pw.trim().length() < 8) {
      error = "Passwort zu kurz (min. 8 Zeichen).";
    } else if (pw.trim().length() > 64) {
      error ="Passwort zu lang (max. 64 Zeichen).";
    } else if (!pw.matches("[èéÈÉäöüÄÖÜß\\-\\_\\.\\w]+")) {
        error = "Ungültige Zeichen im Passwort.";
      }
  } else {
    error = "Passwort zu kurz (min. 8 Zeichen).";
  }
  return error;
}
```

5 Sicherheit

5.1 SQL-Injection

Die Validierung der Werte vor dem ausführen der Speicher-Funktion sollte schon verhindern, dass gefährliche Zeichen in den Feldwerten vorhanden sind. Trotz dieser Absicherung werden die SQL-Statements fest mit Platzhaltern als PreparedStatement definiert und die Feldwerte werden darüber gesetzt, wodurch sie nur als Daten und nicht als Befehle interpretiert werden können.

Listing 11: PreparedStatement

```
try (Connection con = ConnectionHandler.getConnection()) {
  try (PreparedStatement stm =
      con.prepareStatement("UPDATE `company` SET `password`= ?"
        + "WHERE `username`= ? AND `password` = ?")) {
    stm.setString(1, hash(username, newPassword));
    stm.setString(2, username);
    stm.setString(3, hash(username, oldPassword));

    stm.execute();
    return stm.getUpdateCount() > 0;
  }
}
```

5.2 Cross-site Scripting

Um Cross-site Scripting zu verhindern werden in den JSP-Files nur dynamische Inhalte dargestellt, welche vom Controller gesetzt wurden und keine Daten aus den Request-Parameter. Der Controller ruft für alle Ausgaben, welche nicht fest im Programm definiert sind, an die JSP-Files zuerst die Funktion encodeHtml auf, welche alle potentiell gefährlichen Zeichen durch Html-Platzhalter ersetzt.

Listing 12: encodeHtml

```
    @Nonnull
    public static String encodeHTML(@Nonnull String s)
3   {
        StringBuffer out = new StringBuffer();
        for(int i=0; i<s.length(); i++)
        {
            char c = s.charAt(i);
8           if(c > 127 || c=='"' || c=='<' || c=='>')
            {
               out.append("&#"+(int)c+";");
            }
            else
13          {
                out.append(c);
            }
        }
        return out.toString();
18  }
```

5.3 Error Handling

Damit keine internen Informationen über die Implementierung nach aussen dringen, wurden im web.xml zwei Seiten definiert, welche bei Fehlern angezeigt werden. 404.jsp wird dem User angezeigt, wenn unter der eingegebenen URL keine Ressource registriert ist, bei allen anderen Fehlern wird error.jsp angezeigt.

Listing 13: Error Handling

```
  <error-page>
2  <error-code>404</error-code>
     <location>/RattleBits/404.jsp</location>
  </error-page>
  <error-page>
     <location>/RattleBits/error.jsp</location>
7 </error-page>
```

5.4 Passwortspeicherung

Damit bei unerlaubtem Zugriff auf die Datenbank nicht die Userpasswörter ausgelesen werden können wird ein SHA-256 wert berechnet und in der Datenbank gespeichert. Damit dieser Hash nicht einfach mit einer tabelle verglichen und somit zum Passwort zurückgerechnet werden kann, wird der username vorangestellt und mit dem Passwort gehasht, da die Applikation keine änderung des Usernamens zulässt, ist dies nicht so problematisch.

6 SSL

Für die Verschlüsselung der Applikation mit SSL wurde die SSL Funktionalität von Tomcat verwendet. In einem ersten Schritt wurde mit dem keytool von Java ein keystore mit einem 2048 bit RSA Schlüssel generiert. Dieser keystore wurde über das server.xml im Tomcat eingebunden, so dass SSL-Verbindungen auf den Tomcat Server möglich sind.

Listing 14: server.xml Auszug

```
<Connector port="8443" maxThreads="150" scheme="https" secure="true"
    SSLEnabled="true"
    keystoreFile="conf/RattleBitsKeyStore" keystorePass="apsilab"
       clientAuth="false"
    keyAlias="RattleBits" sslProtocol="TLS"/>
```

Danach wurde im web.xml definiert, dass die URL-Patterns /Login und /Overview nur über SSL aufgerufen werden sollen, so dass automatisch umgeschalten wird.

Listing 15: web.xml Auszug

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>ApsiUebung2</web-resource-name>
        <url-pattern>/RattleBits/Login</url-pattern>
        <url-pattern>/RattleBits/Overview</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```