Hochschule München
Fakultät für Elektrotechnik
und Informationstechnik

Bachelorarbeit

von Jakob Schöttl

FBWEBAUTH – Entwurf und Realisierung der Zugriffskontrolle für die inhouse Webanwendung FBWEB

Laufende Nummer:	638
Bearbeitungsbeginn:	10. Oktober 2012
Abgabetermin:	4. April 2013

SIEMENS

Hochschule München
Fakultät für Elektrotechnik
und Informationstechnik

Bachelorarbeit

von Jakob Schöttl

FBWEBAUTH – Entwurf und Realisierung der Zugriffskontrolle für die inhouse Webanwendung FBWEB

FBWEBAUTH – Design and implementation of access control for the in-house web application FBWEB

Laufende Nummer:	638
Betreuer (Hochschule):	Prof. Dr. Manfred Gerstner
Betreuer (Siemens AG):	Bernhard Pottler
Bearbeitungsbeginn:	10. Oktober 2012
Abgabetermin:	4. April 2013

Hochschule München
Fakultät für Elektrotechnik und Informationstechnik

Erklärung des Bearbeiters

Schöttl	Jakob
Name	Vorname

1) Ich erkläre hiermit, dass ich die vorliegende Bachelorarbeit selbständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe.

   Sämtliche benutzte Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate sind als solche gekennzeichnet.

Otterfing, 2013-04-01	
Ort, Datum	Unterschrift

2) Ich erkläre mein Einverständnis, dass die von mir erstellte Bachelorarbeit in die Bibliothek der Hochschule München eingestellt wird. Ich wurde darauf hingewiesen, dass die Hochschule in keiner Weise für die missbräuchliche Verwendung von Inhalten durch Dritte infolge der Lektüre der Arbeit haftet. Insbesondere ist mir bewusst, dass ich für die Anmeldung von Patenten, Warenzeichen oder Geschmacksmuster selbst verantwortlich bin und daraus resultierende Ansprüche selbst verfolgen muss.

Otterfing, 2013-04-01	
Ort, Datum	Unterschrift

Abstract

This paper describes design, implementation and the subsequent integration of a login system with access control for a PHP-based web application. This web application is a in-house software to support workflows in the specialist counselling of Siemens AG. In the beginning, requirements on the login system and the access control will be discussed. Security, useability and reusability play an important role. Especially reuseability is decisive because the system have to endure a planed modernization of the application and will also be integrated in other software of the department. After the requirements section, design and outcome will be reviewed. Here the solution as a whole will be introduced, with a description on how to integrate it in the web application. Furthermore the developed PHP classes and their interfaces will be documented. Finally there is a discussion of some questions and problems which were found during the design and implementation process. These include inter alia the design of the auto login feature, the choice of the password API and remarks on porting the system.

Zusammenfassung

Die vorliegende Arbeit beschreibt Entwurf, Implementierung und die nachträgliche Integration eines Loginsystems mit Zugriffskontrolle für eine bestehende PHP-basierte Webanwendung. Bei der Webanwendung handelt es sich um eine abteilungsinterne Software zur Unterstützung von Arbeitsabläufen bei der Fachberatung der Siemens AG. Zuerst werden die Anforderung an das Loginsystem und die Zugriffskontrolle erörtert. Dabei spielen Sicherheit, Benutzerfreundlichkeit und die einfache Wiederverwendbarkeit in anderen Webanwendungen eine wichtige Rolle. Vor allem die Wiederverwendbarkeit ist entscheidend, da das System die geplante Modernisierung der Anwendung überstehen, und auch in andere Software der Abteilung integriert werden soll. Nach den Anforderungen wird der Entwurf und das Resultat besprochen. Hier wird die Gesamtlösung vorgestellt und beschrieben, wie sie sich in die Webanwendung einbinden lässt. Außerdem werden die entstandenen PHP-Klassen und Schnittstellen dokumentiert. Am Ende werden weitere Frage- und Problemstellungen diskutiert, die sich bei Entwurf und Implementierung aufgetan haben. Dazu zählen unter anderem der Entwurf der Autologin-Funktion, die Wahl der Passwort-API und Anmerkungen zur Portierung des Systems.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Webanwendungen sind Programme, die im Browser dargestellt und bedient werden und größtenteils auf zentralen Servern ablaufen. Sie haben Vorteile wie Plattformunabhängigkeit und günstigere Wartung, da Updates nur an einer Stelle eingespielt werden müssen und nicht bei jedem Client einzeln. Deshalb werden Webanwendungen gerne bei Unternehmen und Institutionen wie Behörden anstatt nativer, fest installierter Anwendungen eingesetzt. So auch bei der Siemens AG.

Im Siemens-Intranet gibt es viele allgemeine Webanwendungen für Aufgaben wie Travel Management, Verwaltung elektronischer Signaturen, Firmenausweisverwaltung und Ähnliches. Viele Abteilungen haben zusätzlich eigene Anwendungen, z. B. zur Inventarisierung, Verwaltung und um Arbeitsabläufe zu unterstützen. Eine solche Anwendung ist FBWeb.

1.1 Die Webanwendung FBWeb

FBWeb ist eine abteilungsinterne Webanwendung der Siemens Fachberatung für Automatisierungstechnik (GER I S BAY TECHCON). Sie basiert auf PHP und läuft auf einer WAMP-Plattform. Sie entstand 2000 als Eigenentwicklung und wurde seitdem von verschiedenen Mitarbeitern der Abteilung und Studenten weiterentwickelt.

Die Entwickler sind keine gelernten Softwareingenieure und es konnte bisher auch niemand vollzeitlich für Wartung und Weiterentwicklung abgestellt werden. Der Leser sollte sich auch vor Augen halten, dass FBWeb ein „Mittel zum Zweck“ ist, und nicht das Produkt einer professionellen Softwarefirma. Seinen Zweck erfüllt es trotzdem bereits sehr gut:

FBWeb unterstützt die Arbeitsabläufe bei der Fachberatung. Die Mitarbeiter verwalten damit Kundendaten, erstellen sogenannte Fälle, bearbeiten diese und erstellen Berichte. Des Weiteren werden Kundenzufriedenheitsumfragen unterstützt.

1.2 Das Projekt FBWebAuth

Mit meiner Bachelorarbeit FBWebAuth trage ich zwei weitere Komponenten zur Software FBWeb bei: Ein Loginsystem (AUTHentication) und eine Zugriffskontrolle (AUTHorization).

Eine primitive Authentifizierung gab es bereits, die zufällige Besucher der Seiten davon abhalten sollte, Daten zu manipulieren. Diese war aber konzeptionell unsicher. Ebenso die Zugriffskontrolle, die kein durchgängiges Konzept realisiert und die Datenbank stark denormalisiert.

- Die Verbindung war komplett unverschlüsselt, u. a. das Passwort wurde also immer im Klartext übertragen.
- Auch in der Datenbank wurde das Passwort im Klartext gespeichert.
- Die Session wurde nicht genutzt, um den momentan angemeldeten Benutzer zu speichern. Stattdessen mussten Benutzer-ID und Passwort bei jeder Anfrage (GET/POST) mitübertragen werden.
- Entsprechend oft musste man sich neu anmelden, wenn diese Parameter bei der Navigation durch die Webanwendung verloren gingen.

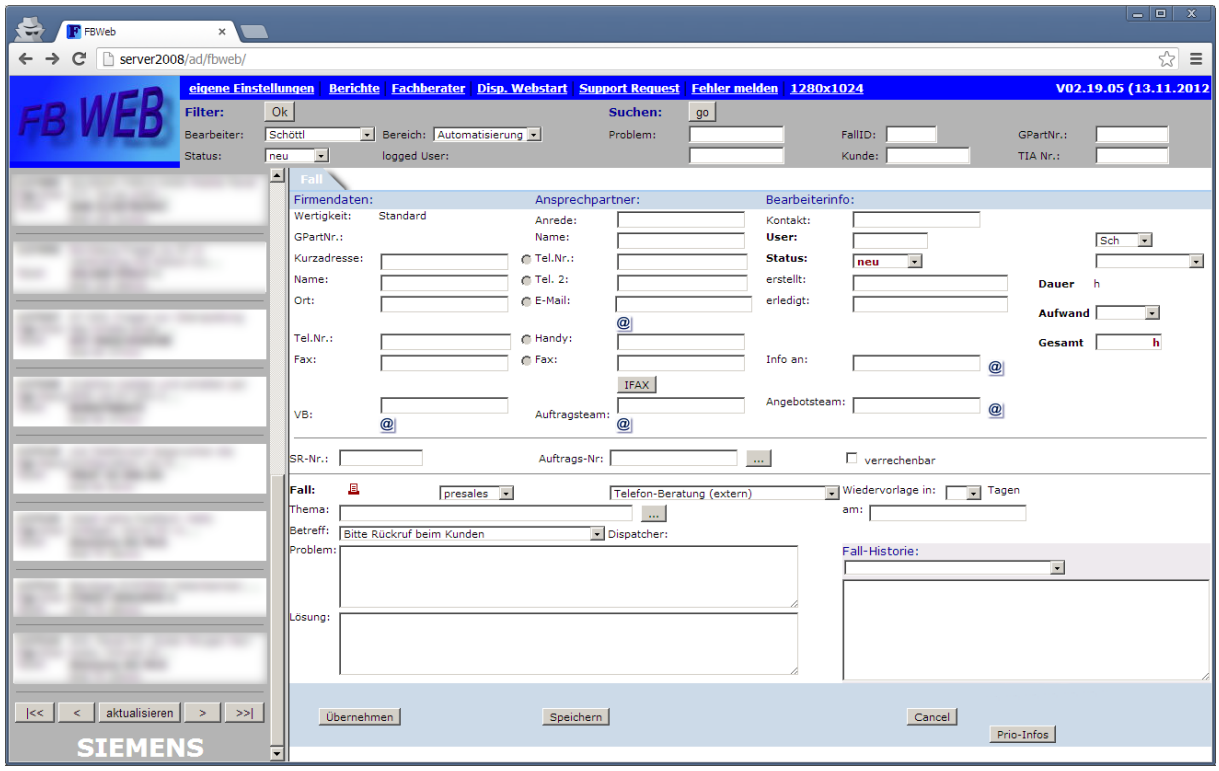

Abbildung 1: Der Screenshot zeigt das Hauptformular zur Bearbeitung von Fällen der Fachberatung. Ist ein bestimmter Fall geladen, können noch sieben weitere Registerseiten angezeigt werden, u. a. „Kundendaten“, „Aktivitäten“ und „Kundenzufriedenheit“.

- Die meisten Seiten konnten nicht durch einen einfachen Hyperlink aufgerufen werden – wegen der Authentifizierung und auch weil sich die URL wegen der Frameset-Architektur nicht ohne weiteres herausfinden ließ.
- Teilweise war das Passwort auch in der URL sichtbar, in der Form `.../index.php?user=1234-&password=password`. Gleichzeitg war eine solche URL als Lesezeichen im Browser für die Mitarbeiter eine willkommene Lösung um den ständigen Login zu umgehen.
- Bei der Zugriffskontrolle wurden diverse Spalten in der Mitarbeitertabelle hinzugefügt, die Angaben zu Rollen oder Berechtigungen enthalten.
- Dadurch mussten wiederum die Datensätze vieler Mitarbeiter dupliziert werden, wenn sich z. B. eine doppelte Rollenzugehörigkeit nicht anders abbilden ließ.

Mit FBWebAuth soll ein session-basiertes Loginsystem mit rollenbasierter Zugriffskontrolle eingeführt werden, das die eben genannten Mängel beseitigt. Dabei soll das System zudem benutzerfreundlicher und die Datenbank normalisiert werden.

2 Anforderungen

Die allgemeinen Anforderungen an FBWebAuth sind folgende:

- Die Webanwendung soll sicherer werden.
- Die Benutzerfreundlichkeit soll nicht unter den Neuerungen leiden.
- Die Lösung soll einfach in *bestehende* PHP-Seiten eingebunden werden können.
- Die Lösung soll auch dann noch passend sein, wenn sich die Architektur der PHP-Seiten[1] ändert.
- Die Lösung soll auch in anderen Webanwendungen möglichst einfach integriert werden können.

Neben diesen Grundlagen müssen Loginsystem und Zugriffskontrolle natürlich das tun, was man von ihnen erwartet. Die konkreten Anforderungen sind in den folgenden beiden Absätzen zusammengefasst.

2.1 Loginsystem

Der Server, auf dem FBWeb läuft, befindet sich im Siemens-Intranet unter der Domäne `siemens.net` und ist damit für jeden Mitarbeiter der Siemens AG sichtbar. Das Loginsystem soll sicherstellen, dass nur berechtigte Personen mit FBWeb arbeiten können. Zu diesem Zweck müssen sich die Mitarbeiter mit Benutzername und Passwort anmelden, um sich gegenüber dem System zu authentifizieren (4.2). Wichtig ist dabei aber auch eine gewisse Benutzerfreundlichkeit: Die Fachberater sollen sich nicht jeden Tag einmal oder gar mehrmals ausweisen müssen, sondern sollen an ihrem PC angemeldet bleiben (4.2.7). Desweiteren sollte es für den Benutzer möglich sein, das Passwort zu ändern bzw. zurückzusetzen, wenn es vergessen wurde. Dabei sollen gewisse Regeln für das Passwort durchgesetzt werden (4.4.6).

[1] Damit ist gemeint: Wird der HTML-Code von einer Methode einer PHP-Klasse erzeugt, oder steht HTML- und PHP-Code direkt in z. B. `index.php`.

2.2 Zugriffskontrolle

Die Zugriffskontrolle baut auf das Loginsystem auf und soll angemeldeten Benutzern bestimmte Aktionen erlauben bzw. verbieten. Dabei lassen sich den Benutzer bestimmte Rollen zuordnen. Erste Überlegungen ergaben die Rollen Fachberater (FB), First-Level Support (FLS), Leiter der Fachberatung (LF), Promotoren (Prom) und Vertriebsbeauftragte/Sonstige (VB/S). Jede Rolle besitzt unterschiedliche Rechte, z. B. das Erfassen, Bearbeiten und Anzeigen von Fällen oder das Ansehen von Berichten. Tabelle 1 zeigt, welche Rechte welchen Rollen zugeordnet sind.

	Erfassen	Bearbeiten	Anzeigen	Berichte
FB	x	x	x	x
FLS	x	x	x	x
LF			x	x
Prom			x	x
VB/S				x

Tabelle 1: Diese Tabelle zeigt grob die Rechte für verschiedene Rollen.

Die Anforderungen noch einmal zusammengefasst:

- Benutzer müssen Rechte haben können.
- Die Rechte müssen den Benutzern bequem über Rollen zuweisbar sein.
- Ein Benutzer soll mehrere Rollen innehaben können.
- Das System soll eine einfache API bieten und damit gut in PHP verwendbar sein.

Diese Punkte führen zu einer rollenbasierten Zugriffskontrolle. Nach weiterer Betrachtung wurde klar, was die Zugriffskontrolle leisten muss, welche Arten von Rechten es geben muss und wie sie sich in etwa in die PHP-Seiten einfügt.

1. Zum einen ist da die Frage, ob der momentan angemeldete Benutzer *grundsätzlich Zugriff* auf eine PHP-Seite hat. Zum Beispiel dürfen sich nur Benutzer mit der Rolle „Administrator“ die Seiten unterhalb von `/admin/` ansehen. Alle anderen Benutzer sollen die Seite nicht anzeigen können, und beim Versuch diese aufzurufen mit einem Hinweis zurückgeleitet werden.
2. Desweiteren gibt es innerhalb einer Seite *seitenspezifische Rechte*. So dürfen auf der Seite zum Anzeigen und Bearbeiten eines Falls z. B. nur FB und FLS tatsächlich die Fälle bearbeiten. Alle anderen, die nicht (mindestens) eine dieser Rollen innehaben, dürfen den Fall wirklich nur anzeigen.
3. Zuletzt soll es noch *allgemeine (d. h. nicht-seitenspezifische) Rechte* geben. Damit wird z. B. auf allen Seiten der Hyperlink „Zur Administration“ nur für Benutzer mit der Rolle „Administrator“ angezeigt.

3 Methodenteil

3.1 Programmierung

Die Webanwendung wird entwickelt in PHP mit Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) und JavaScript. PHP ist eine Skriptsprache, die serverseitig interpretiert und ausgeführt wird. PHP unterstützt prozedurale und objektorientierte Programmierung. Mit PHP werden Geschäftslogik und Datenbankkommunikation realisiert sowie die dynamischen HTML-Seiten erzeugt.

3.1.1 PHPDoc

Zur Dokumentation der PHP-Programme verwenden wir PHPDoc. Damit werden Quelldateien, Funktionen und Klassen (mitsamt Methoden und Eigenschaften) dokumentiert. PHPDoc ist zum einen eine festgelegte Syntax für Kommentare (angelehnt an JavaDoc) und zum anderen ein Werkzeug, das aus den Kommentaren „druckbare“ Dokumentation (HTML, Portable Document Format (PDF), . . .) erzeugt.

3.1.2 PHPUnit

PHPUnit ist ein Framework für PHP-Modultests. Wir verwenden es um einige grundlegende Klassen gründlich zu testen. Der große Vorteil gegenüber experimentellem Testen mit `echo` und `var_dump` etc. ist, dass die Tests jederzeit wiederholt werden können.

3.2 Entwicklungsumgebung

3.2.1 Versionsverwaltung: Git

Mitte 2011 wurden alle abteilungsinternen Softwareprojekte auf Git migriert, ein modernes Version Control System (VCS). Git ist eine kostenlose, quelloffene „Source Code Management“-Software, die vor allem durch ihren „verteilten“ Charakter Vorteile gegenüber vielen anderen VCSs hat. „Verteilt“ bedeutet hier, dass jeder Entwickler ein lokales Repository hat, mit dem er arbeitet. Nur hin und wieder wird mit dem zentralen Repository auf dem Server synchronisiert. Dadurch gibt es bei den allermeisten Arbeiten keine Latenzzeiten.

Ein zentrales Repository für FBWeb liegt auf einem Server im Intranet. Dies ist einfach ein Verzeichnis namens `FBWeb.git`, in dem alle Mitentwickler Lese- und Schreibrechte haben. Der Server ist von CAT-Clients (also den Firmen-PCs) aus erreichbar, im Intranet oder über das Internet via Universal Remote Access (URA) (ein Virtual Private Network von Siemens). Somit können die Entwickler auch von zuhause aus Aktualisierungen abfragen oder hochladen.

3.2.2 Integrierte Entwicklungsumgebung (IDE)

Bei größeren Projekten ist eine IDE unbedingt sinnvoll. Neben Editor und Dateibrowser bietet eine IDE viele Features, die die Entwicklung erheblich erleichtern. Bei der Entwicklung der abteilungsinternen PHP-Anwendungen ist NetBeans IDE die erste Wahl. Mit ein paar Anpassungen der Einstellungen kann aber auch Eclipse for PHP Developers verwendet werden.

NetBeans IDE NetBeans ist eine bekannte IDE – momentan Version 7.3 – mit mehr oder weniger guter Unterstützung für diverse Sprachen wie PHP, HTML/CSS, verschiedene Java-Ausprägungen und C/C++. NetBeans IDE legt im Projektverzeichnis den Ordner `nbproject` an. Darin sind Einstellungen zum Projekt gespeichert, z. B. Zeichensatz der Dateien, und ob zur Einrückung Tabulator- oder Leerzeichen verwendet werden. Benutzerdefinierte Einstellungen befinden sich im Unterordner `private`; dieser unterliegt nicht der Versionskontrolle.

Eclipse for PHP Developers Diese IDE basiert auf der Eclipse Plattform, momentan Version 3.6 (Helios) und wird entwickelt im Eclipse Projekt PHP Development Tools (PDT). Eclipse besitzt einen mächtigen Editor und lässt an Einstellungsmöglichkeiten kaum zu wünschen übrig. Ein Vorteil gegenüber NetBeans – gerade bei der Umstellung auf UTF-8 – ist die Möglichkeit, bestimmte Einstellungen für Resourcen (Ordner, Dateien) auch einzeln festzulegen. Praktisch sind auch die „Save Actions“: Hier kann z. B. eingestellt werden, dass Leerzeichen am Zeilenende vor dem Speichern automatisch entfernt werden. Hinsichtlich Refactoring ist Eclipse NetBeans unterlegen – es kann keine Bezeichner umbenennen.

3.2.3 Datenbankentwicklung

Für die Arbeiten an der Datenbank läuft auf dem Webserver eine phpMyAdmin-Installation. Alternativ kann auch ein clientseitig installiertes Programm verwendet werden wie HeidiSQL oder MySQL Workbench.

phpMyAdmin Diese, in PHP geschriebene Anwendung stellt ein Graphical User Interface (GUI) zum Verwalten von MySQL-Datenbanken bereit. Als Webanwendung ist sie aber weniger performant und bietet keine Kontextmenüs oder Tastenkürzel. Die meisten Arbeiten sind somit umständlicher oder langsamer zu erledigen als mit nativen Anwendungen.

MySQL Workbench MySQL Workbench ist die umfassende Lösung für alle Aufgaben rund um MySQL-Datenbanken, von Entwurf und Entwicklung über Administration bis hin zur Datenmigration. Das Programm gibt es kostenlos als native Anwendung für Windows, Linux und Mac OS [6].

Neben den Vorteilen bezüglich des effizienteren Arbeitens, ist die Modellierungsumgebung ein sehr nützliches Feature. Hier kann die Datenbank als Enhanced Entity-Relationship (EER)-Diagramm entworfen werden. Auch Reverse Engineering, also Erzeugen des Diagramms aus der bestehenden Datenbank, ist möglich.[2]

3.3 Produktivsystem

Produktiv läuft FBWeb auf einer XAMPP-Installation unter Microsoft Windows XP. XAMPP ist eine Distribution von Apache (HTTP-Webserver), MySQL (Datenbank), PHP und Perl. Dabei gibt es Installationspakete für fast alle wichtigen Betriebssysteme. Das praktische an diesen Distributionen ist, dass der Server nach der Installation direkt lauffähig und vorkonfiguriert wird. Es sind nur noch wenige Anpassungen nötig, um den Server für den produktiven Betrieb abzusichern.

[2] Sofern die Datenbank nicht noch auf der älteren Engine MyISAM läuft, welche keine referentielle Integrität unterstützt. Bei solchen Datenbanken können mittels Reverse Engineering keine Beziehungen zwischen den Tabellen ermittelt werden.

4 Ergebnis

4.1 Datenbankverbindung

Beide neuen Komponenten, Loginsystem und Zugriffskontrolle, benötigen eine Datenbankverbindung. Diese wird durch Objekte der Klasse `DBAccess` bereitgestellt. Beim Herstellen der Datenbankverbindung können im Ausnahmefall Fehler auftreten. In diesem Fall ist eine entsprechende Fehlerseite anzuzeigen.

Um diese Fehlerbehandlung nicht auf jeder Seite schreiben zu müssen, ist der Aufbau der Datenbankverbindung in das PHP-Skript `make_dbaccess.php` ausgelagert. Dieses Skript wird per `require`-Anweisung dort eingebunden, wo eine Datenbankverbindung gebraucht wird. Damit steht dann die Variable `$dbh` (Klasse `DBAccess`) zur Verfügung, über die auf die Datenbank zugegriffen werden kann. Ein Beispiel:

```
<?php
require 'make_dbaccess.php';
// Ab jetzt steht die Variable $dbh
// (Klasse DBAccess) zur Verfuegung!
echo $dbh->pquery("SELECT␣name␣FROM␣mitarbeiter␣WHERE␣id␣=␣?", 1234)->fetchColumn
    ();
```

Für mehr Informationen zur Klasse `DBAccess` siehe 4.4.1 auf Seite 20 und Anhang.

4.2 Loginsystem

Die neuen PHP-Seiten sind, wie auch die bestehenden, noch nicht angepasst an das Corporate Design für Webauftritte. Dies wird im Zuge der sowieso anstehenden Aktualisierung realisiert.

4.2.1 Authentifizierung

Eine Überprüfung, ob ein Benutzer angemeldet ist, findet am Anfang jeder geschützen PHP-Seite statt. Dazu wird das Skript `db_authenticate.php` mit der `require`-Anweisung eingebunden. Das Skript bindet seinerseits `make_dbaccess.php` (4.1) ein, welches die benötigte Datenbankverbindung aufbaut. Anschließend wird sichergestellt, dass

1. eine verschlüsselte Verbindung besteht.
2. eine aktive Session besteht
   (a) mit gültigem Benutzernamen und gültiger -ID *und*
   (b) einer unveränderten IP-Adresse (um Session-Hijacking vorzubeugen)
3. *oder* die Session durch einen Autologin-Zugang wieder aufgenommen werden kann.

Andernfalls wird nocheinmal korrekt abgemeldet[3] und auf die Login-Seite weitergeleitet.

Das Skript `db_authenticate.php` definiert eine weitere Variable: `$session`, ein Objekt der Klasse `Session`. Damit kann unter anderem der momentan angemeldete Benutzer abgefragt werden. Ein Beispiel:

[3] Unter Umständen könnte die Session nämlich noch nicht ordnungsgemäß beendet sein.

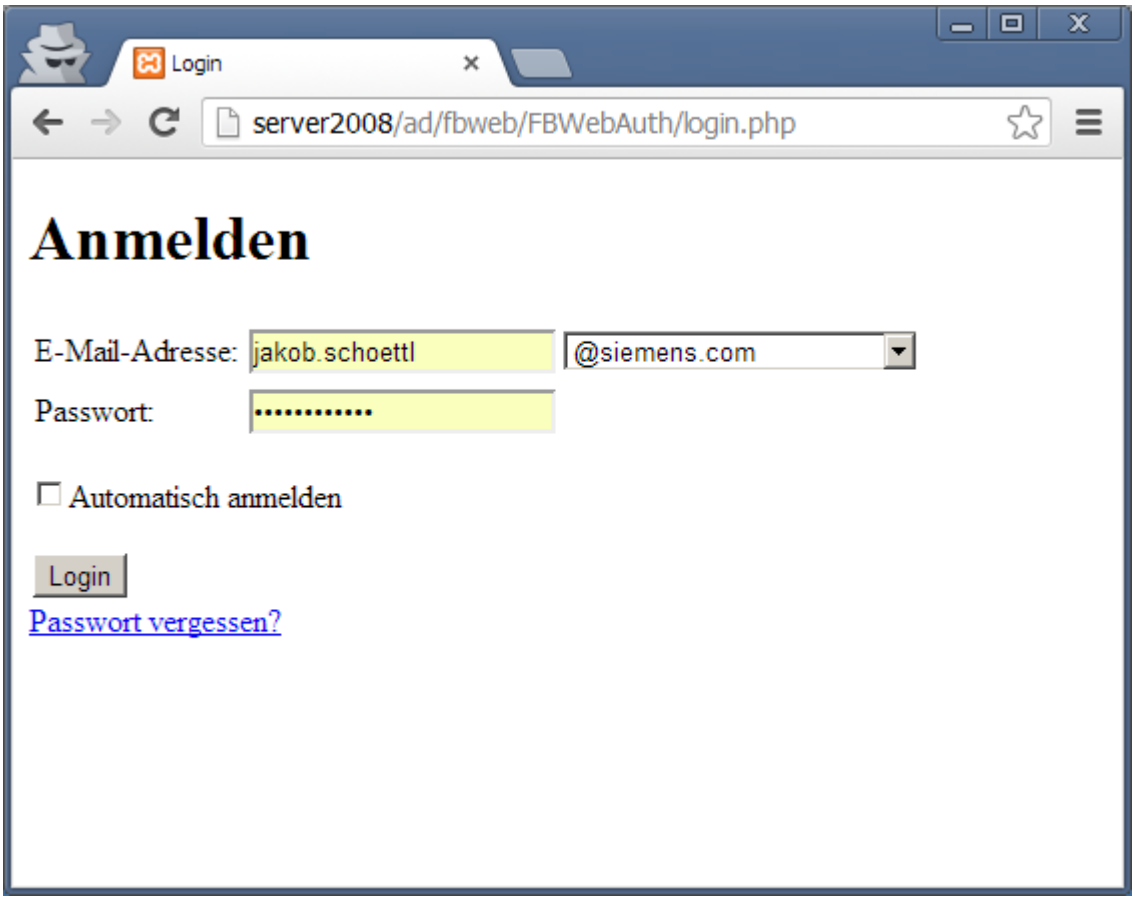

Abbildung 2: Dieser Screenshot zeigt die Anmeldeseite in ihrer Rohform. Sie ist noch nicht in die bestehende Webanwendung integriert bzw. an das Corporate Design angepasst.

```
<?php
require 'db_authenticate.php';
// Ab hier steht die Variable $session zur Verfuegung!
echo $session->getCurrentUser()->getUsername();
```

4.2.2 Login

Auf der Login-Seite meldet sich der Benutzer mit seinem Benutzernamen (E-Mail-Adresse) und seinem Passwort an. Dabei kann er einstellen, ob er die nächste Zeit[4] angemeldet bleiben möchte.

Die Domäne der E-Mail-Adresse lässt sich in einem Kombinationsfeld auswählen. Darin befinden alle möglichen Domänen[5], nach Häufigkeit[6] absteigend sortiert. Dazu gibt es einen leeren Eintrag, falls der Benutzer die komplette E-Mail-Adresse selbst eingeben will. Ungeachtet dessen wird die ausgewählte Domäne

[4] Die Gültigkeitsdauer des Autologin-Zugangs kann im Quellcode über die entsprechende Konstante noch angepasst werden. Momentan sind 30 Tage eingestellt.

[5] Das heißt die (verschiedenen) Domänen der E-Mail-Adressen aller eingetragenen Mitarbeiter.

[6] Aktuell kommen nur drei Domains vor. Bei einer größeren Anzahl wäre eine alphabetische Sortierung wahrscheinlich sinnvoller.

ignoriert, wenn der Benutzer eine komplette E-Mail-Adresse eingibt. Standardmäßig ist die am häufigsten vorkommende Domäne ausgewählt. Die Auswahl wird als Cookie gespeichert, sodass der Benutzer nicht jedes Mal neu auswählen muss.

Meldet sich der Benutzer mit einem Häkchen im Kontrollfeld „Automatisch anmelden“ an, wird ihm ab dann – solange der Autologin-Zugang gültig ist – die Login-Seite erspart. Dieses Feature wird im Folgenden *Autologin* genannt.

Intern wird beim Klick auf die Schaltfläche „Login“ die `logIn`-Methode der Klasse `Session` aufgerufen. Dabei werden drei Parameter übergeben: Benutzername, Passwort und ein Autologin-Flag. Die Methode prüft dann mithilfe der Klasse `Authenticator` ob die Anmeldedaten korrekt sind, und erstellt gegebenenfalls einen Autologin-Zugang.

4.2.3 Logout

Zum Abmelden wird intern die `logOut`-Methode der Klasse `Session` aufgerufen. Damit wird die Session und – falls vorhanden – der Autologin-Zugang zerstört. Dadurch ist sichergestellt, dass sich ein Benutzer anschließend wieder mit Benutzername und Passwort authentifizieren muss.

Das Logout findet statt, wenn

- das Skript `db_authenticate.php` eine Unstimmigkeit feststellt (siehe 4.2.1).
- der Benutzer sich selbst abmeldet, indem er die Seite `logout.php`[7] aufruft. Diese leitet anschließend weiter auf die Seite `login.php`.

4.2.4 Verschlüsselte Verbindung

Vorraussetzung für eine sichere Webanwendung ist eine verschlüsselte Verbindung. Hierfür wird das HTTPS-Protokoll verwendet. Bei XAMPP ist das entsprechende Apache-Modul mod_ssl standardmäßig installiert. Das Zertifikat kann mit dem Programm openssl erstellt werden und muss dann bei allen Clients lokal installiert werden.

Es ist Aufgabe des `db_authenticate.php`-Skripts (4.2.1) zuallererst sicherzustellen, dass die Seite via HTTPS aufgerufen wird. Wenn das Protokoll nicht HTTPS ist, wird automatisch umgeleitet. Zusätzlich kann bei Apache das Modul mod_rewrite so konfiguriert werden, dass es Aufrufe einer URL, die mit `http://` beginnt, automatisch „umschreibt“ (also weiterleitet) auf `https://`.

4.2.5 Benutzernamen

Als Benutzername, der beim Anmelden zusammen mit dem Passwort angegeben werden muss, dient die primäre E-Mail-Adresse. Die Gründe für diese Entscheidung werden auf Seite 28 besprochen.

4.2.6 Passwörter

Die Passwörter werden beim Anmelden verschlüsselt via HTTPS übertragen. In der Datenbank wird das Passwort nicht im Klartext gespeichert, sondern ein gesalzener Hashwert. In FBWebAuth wird die Secure Password Hashing API von PHP 5.5 verwendet. Diese greift auf den Hash-Algorithmus bcrypt zurück.

[7] Im Kopf jeder Seite wird ein entsprechender Hyperlink stehen.

Hash-Algorithmen Die Passwörter der Benutzer dürfen nicht im Klartext in der Datenbank stehen. Stattdessen wird üblicherweise ein Hashwert gespeichert. Bekannte Hash-Algorithmen sind Message Digest 5 (MD5) oder Secure Hash Algorithm (SHA). Diese wurden aber nicht zum Hashing von Passwörtern entwickelt, sondern um Prüfwerte für größere Datenmengen zu erzeugen. Dabei spielt Effizienz eine Rolle, und genau das ist der Nachteil, wenn man diese Algorithmen auf Passwörter anwendet: Bei einer Brute-Force-Attacke (ohne Rainbow-Table) können viele Millionen Passwörter pro Sekunde einfach durchprobiert werden [3].

Als Lösung bieten sich Algorithmen wie bcrypt an. bcrypt wurde speziell für das Hashing von Passwörtern entwickelt. Dabei wird die Hashfunktion nicht nur einmal auf das Klartextpasswort, sondern wiederholt auch auf das Ergebnis angewendet. Für den normalen Anwender macht es keinen Unterschied, ob die Hashfunktion beispielsweise 1000 mal so lange braucht wie MD5 – für Angreifer lohnt sich die Brute-Force-Atacke aber kaum mehr [2].

Salz Zusätzlich sollten die Hashwerte der Passwörter gesalzen (salted) sein. Dies bedeutet, dass zu jedem Passwort ein zusätzlicher zufälliger Wert gespeichert wird, der zusammen mit dem Klartextpasswort der Hashfunktion übergeben wird. Die Hashfunktion berechnet den Hashwert also nicht allein aus dem Passwort, sondern aus der Kombination von Passwort und Salz. Damit fallen auch Wörterbuchangriffe weg: Der große Vorteil dieser Angriffe mit sog. Rainbow-Tables, die Wiederverwendbarkeit, gilt nicht mehr, weil eben jedes Passwort mit einem anderen zufälligen Salz gehasht wurde [7].[8]

4.2.7 Autologin

Unter „Autologin“ oder „Remember Me“ versteht man ein Feature von Webanwendungen, das dem Benutzer die tägliche Eingabe von Benutzername und Passwort ersparen soll. Wie in 4.2.2 erwähnt ist ein solches Feature in FBWebAuth implementiert. Durch ein spezielles Cookie wird der Benutzer wiedererkannt. Das automatische Anmelden eines Benutzeres funktioniert *nicht* parallel in verschiedenen Browsern bzw. auf verschiedenen Computern. Details dazu werden auf Seite 28 diskutiert.

4.3 Zugriffskontrolle

Die Zugriffskontrolle regelt, ob der momentan angemeldete Benutzer auf eine Seite zugreifen darf, welche Rechte er hat und was er auf der Seite sehen darf.

4.3.1 Authorisierung

Um die Zugriffskontrolle auf einer Seite bereitzustellen, wird das Skript `db_authenticate_authorize.php` mit der `require`-Anweisung am Anfang dieser PHP-Seite eingebunden. Dieses Skript bindet seinerseits `db_authenticate.php` ein, übernimmt also die Aufgaben der Authentifizierung (4.2.1) und initiiert zusätzlich die Zugriffskontrolle: Das Skript

1. überprüft, ob der momentan angemeldete Benutzer grundsätzich Zugriff auf die aktuelle Seite hat. Wenn nicht, wird eine entsprechende Fehlerseite angezeigt.

[8] Abgesehen von dem Fall, dass der Zufallsgenerator für zwei Datensätze zufällig das gleiche Salz erzeugt hat.

2. definiert die Variable `$authorizer`, ein Objekt der Klasse `Authorizer`. Damit können Rechte abgefragt werden.

Um zu zeigen, wie die Zugriffskontrolle in PHP-Seiten eingebaut wird, hier ein einfaches Beispiel:

```
<?php
require 'db_authenticate_authorize.php';
// Ab hier stehen folgende Variablen zur Verfügung:
// $dbh        (DBAccess)
// $session    (Session)
// $authorizer (Authorizer)
```

Die Zugriffskontrolle wird allerdings nicht auf jeder Seite benötigt. Zum Beispiel darf jeder angemeldete Benutzer das eigene Benutzerprofil und Passwort ändern. Hier ist keine Unterscheidung nach Rollen nötig. Dementsprechend reicht hier die Authentifizierung, also das Einbinden von `db_authenticate.php`.

4.3.2 Die Authorizer API

Die Programmierschnittstelle für die Zugriffskontrolle wird über die Klasse `Authorizer` zur Verfügung gestellt. Damit gibt es auch zum zweiten Teil von FBWebAuth eine zentrale Klasse. Im Unterschied zur Klasse `Authenticator` wird `Authorizer` aber direkt in den PHP-Seiten verwendet. Hier ist die Klassendefinition ohne Implementierung:

```
class Authorizer {
        function __construct($dbh, $session, $pagePath = null);
        function canCurrentUserAccessPage($pagePath = null);
        function hasCurrentUserRight($subject);
        function hasCurrentUserGeneralRight($subject);
}
```

`canCurrentUserAccessPage` Diese Methode prüft, ob der momentan angemeldete Benutzer grundsätzlich Zugang zur Seite hat.

`hasCurrentUserRight` Diese Methode prüft, ob der momentan angemeldete Benutzer das mit `$subject` angegebene Recht auf der Seite besitzt.

`hasCurrentUserGeneralRight` Diese Methode prüft, ob der momentan angemeldete Benutzer das mit `$subject` angegebene allgemeine (nicht-seitenspezifische) Recht besitzt.

Der Rückgabetyp dieser Funktionen ist `boolean`. Für Details zur Klasse und zu den Methoden und Parametern, siehe 4.4.5. Hier ein Beispiel zur Verwendung der Authorizer API:

```
<?php
require 'db_authenticate_authorize.php';

// Grundsaetzlich Zugang zu anderer Seite?
$r0 = $authorizer->canCurrentUserAccessPage('/editroles.php');
```

```
if ($r0) {
    echo '<a␣href="editroles.php">Rollen␣bearbeiten</a>';
}

// Seitenspezifisches Recht abfragen
$r1 = $authorizer->hasCurrentUserRight('view␣content');

// Allgemeines Recht abfragen z.B. fuer eine Funktion
$r2 = $authorizer->hasCurrentUserGeneralRight('phpinfo()');

if ($r1 && $r2) {
    phpinfo();
} else {
    echo 'Sie␣haben␣nicht␣das␣Recht␣diesen␣Inhalt␣zu␣sehen.';
}
```

4.3.3 Rollenkonzept

Aus den Anforderungen geht hervor, dass zusätzlich zur Tabelle `mitarbeiter` noch zwei Tabellen für Rollen und Rechte benötigt werden. Zwischen Mitarbeitern und Rollen, sowie Rollen und Rechten bestehen m:n-Beziehungen. Eine Vererbungshierarchie bei Rollen (dass also eine „Kindrolle“ alle Rechte der „Vaterrolle“ erbt) ist nicht nötig. Auch das direkte, rollenunabhängige Einräumen bzw. Entziehen von Rechten für Mitarbeiter ist nicht vorgesehen. Die Datenbankstruktur kann jedoch nachträglich einfach erweitert werden, um auch solche Anforderungen zu erfüllen.

Zusätzlich zu diesem Standardschema gibt es zwei Erweiterungen. Dabei spielt die ebenfalls neue Tabelle `webpage`[9] eine wichtige Rolle. Diese Tabelle enthält für jede (nennenswerte) Seite einen Datensatz. Dadurch können Rechte und Rollen mit den Webseiten verknüpft werden:

1. Rechte können an Seiten gebunden sein. Wenn das der Fall ist, gilt das Recht eben nur für die angegebene Seite, nicht allgemein und nicht für andere Seiten.
2. Den Rollen können Seiten zugewiesen werden. Wenn eine Seite a einer Rolle r zugewiesen ist, bedeutet das, dass Mitglieder dieser Rolle r grundsätzlich Zugriff auf die Seite a haben. Andernfalls wird – sofern die Zugriffskontrolle eingebunden ist – auf eine Fehlerseite umgeleitet.

4.3.4 Datenbank

Zur Datenbank kommen also sechs neue Tabellen hinzu. Neben denen für Rollen und Rechte gibt es die Tabelle `webpage`. Dazu kommen drei einfache Tabellen, die lediglich die m:n-Beziehungen zwischen den übrigen Tabellen herstellen. Abbildung 3 stellt dieses Schema dar. Im Folgenden werden die wichtigsten der neuen Tabellen kurz beschrieben.

Rollen (`auth_role`) Diese Tabelle besitzt die Attribute ID, Name, Kürzel und Kommentar/Beschreibung. Die ersten drei sind alle für sich eindeutig. Das Kürzel kann z. B. verwendet werden, um die GUI zur Vergabe von Rollen und Rechten zu vereinfachen.

[9] Diese Tabelle wird auch zum Erzeugen einer Breadcrumbs-Navigationsleiste verwendet.

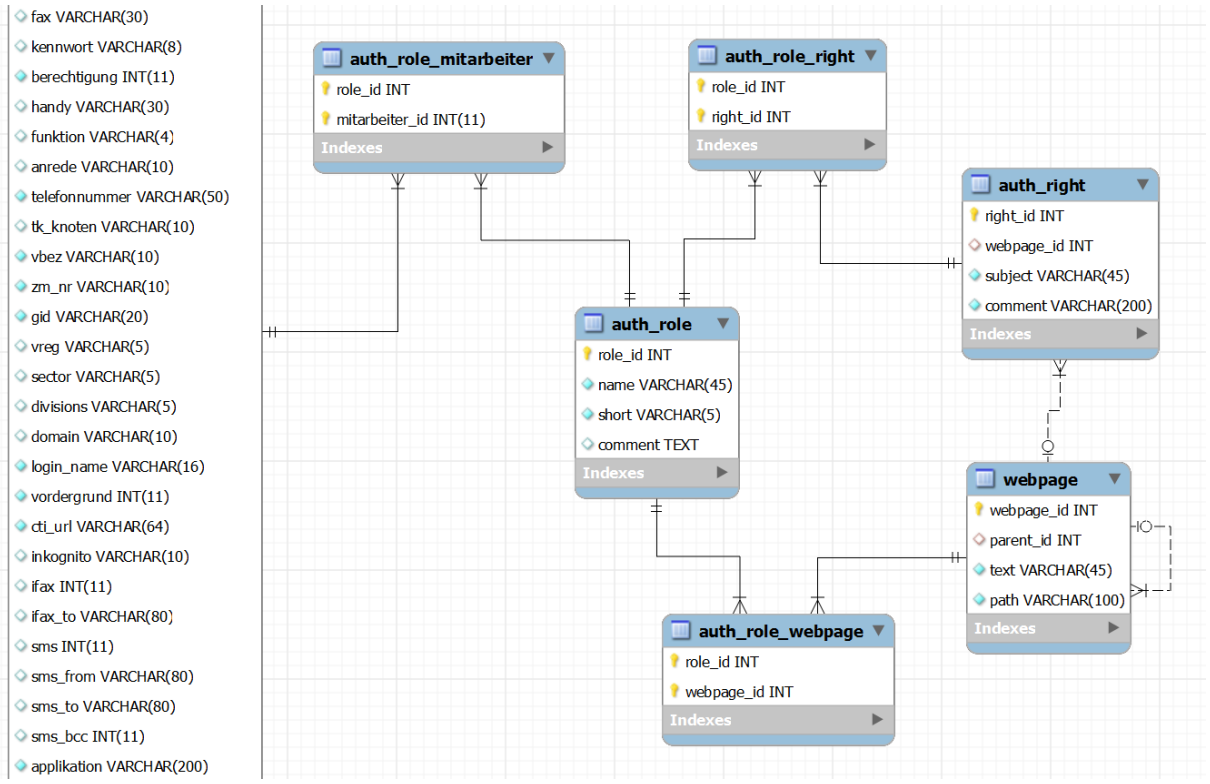

Abbildung 3: Dieses ER-Diagramm zeigt u. a. die Tabellen, die für die Zugriffskontrolle hinzugekommen sind. Diese haben das Präfix `auth_` im Namen. Ganz links ist die Tabelle `mitarbeiter`. Die Tabelle `webpage` wird zusätzlich zum Aufbau der Breadcrumbs-Navigation verwendet.

Rechte (`auth_right`) Diese Tabelle besitzt die Attribute ID, Seiten-ID, Betreff, und Kommentar. Die ID ist für sich eindeutig, Seiten-ID und Betreff sind zusammen eindeutig. Dabei gibt es bei MySQL allerdings eine Schwierigkeit: Wenn bei einem Unique-Index über mehrere Felder `NULL`-Werte enthalten sind, wird die Eindeutigkeit für diese Zeile nicht weiter überprüft.[10] Das Problem kann mit einem Datenbank-Trigger gelöst werden, der das Erstellen oder Ändern einer Zeile verhindert, wenn bereits ein nicht-seitenspezifisches Recht mit gleichem Betreff vorhanden ist.

Seiten (`webpage`) Diese Tabelle besitzt die Attribute ID, Eltern-ID, Text und Pfad. ID und Pfad sind beide für sich eindeutig. Der Pfad ist der absolute Pfad zur Datei (PHP-/HTML-Seite) ab dem „Document Root“ der Webanwendung, beginnend mit einem Schrägstrich (`/`). Dieses Attribut korrespondiert mit dem Parameter `$pagePath`, siehe 4.4.5 auf Seite 22. Die Eltern-ID legt fest, unter welcher anderen Seite die bestimmte Seite in den Breadcrumbs aufgehängt ist. Der Text wird dort als Anzeigetitel verwendet.

[10] Bei einer einzelnen Spalte macht das durchaus Sinn. Beispiel mit den Tabellen `mitarbeiter` und `firmenausweis`: Ein Mitarbeiter *kann* einen Ausweis besitzen. Wenn ein Mitarbeiter einen Ausweis besitzt, ist er der *einzige* Besitzer dieses Ausweises.

4.4 PHP-Klassen

Die Arbeit FBWebAuth hat der Webanwendung FBWeb mehrere neue PHP-Klassen hinzugefügt. Diese Klassen stellen zum einen Hilfsfunktionen bereit und kapseln zum anderen die Hauptfunktionalität des Loginsystems und der Zugriffskontrolle. Im Folgenden werden die wichtigsten Klassen mit ihren Schnittstellen kurz vorgestellt.

4.4.1 DBAccess

`DBAccess` basiert auf PHP Data Objects (PDO) und wird in FBWebAuth als grundlegende Datenbankschnittstelle verwendet. Außerdem wird `DBAccess` auf längere Zeit die gemischte Verwendung der Datenbankschnittstellen `mysql_*` und `mysqli` in FBWeb ersetzen. Die wichtigsten Fragen zur Klasse `DBAccess` und deren Antworten finden sich im Anhang.

4.4.2 Session

Diese Klasse kapselt Erzeugung, Zugriff und Zerstörung der bzw. auf die momentane Session. FBWebAuth bindet die Session außerdem an den momentan angemeldeten Benutzer. Sie steht also grundsätzlich nur angemeldeten Benutzern vollständig zur Verfügung steht. Deshalb ist die Klasse zusätzlich verantwortlich für das An- und Abmelden des Benutzers, sowie das Überprüfen des Status.

4.4.3 User

Ein Objekt der Klasse `User` wird in der Session gespeichert. Es wird beim Anmelden erzeugt und initialisiert mit Benutzername und -ID. Dabei wird auch automatisch die IP-Adresse des Clients gespeichert.

Mit der Methode `isValid` kann abgefragt werden, ob der Benutzer noch gültig angemeldet ist. Dabei wird überprüft, ob

1. ein Benutzer mit der bei der Initialisierung angegebenen Kennung existiert.
2. die Adresse des anfragenden Clients unverändert ist.

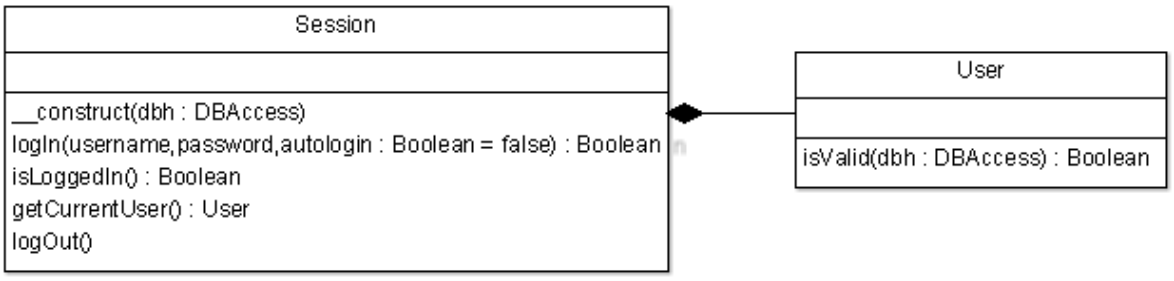

Abbildung 4: Das Klassendiagramm zeigt die Methoden der Klassen `Session` und `User`, und wie diese Klassen zusammenhängen.

4.4.4 Authenticator

Diese Klasse stellt Methoden zum Thema Authentifizierung bereit. Sie greift sie auf die Datenbank zu. In der Regel wird sie nicht direkt verwendet, sondern über das `db_authenticate.php`-Skript (4.2.1) und die Klasse `Session`. Allgemein gilt in dieser Klasse:

1. Der Parameter `$id` ist die Benutzer-ID.
2. Der Parameter `$password` ist das Passwort im Klartext.
3. Der Parameter `$key` ist ein (sogenannter) *Schlüssel* und ist als langes, starkes Passwort zu verstehen, mit dem sich ein Benutzer authentifizieren kann.
4. Alle `make*`- und `update*`-Funktionen geben bei Erfolg einen Schlüssel und ansonsten `false` zurück.
5. Alle `verify*`-Funktionen geben bei Erfolg die Benutzer-ID und ansonsten `false` zurück.

Im Folgenden werden die wichtigsten Funktionen der Klasse vorgestellt.

Festsetzen des Passwortes Ein gesalzener Hashwert des Passwortes wird in der Datenbank gespeichert.

- **function** `setPassword($id, $password)`

Normale Authentifizierung Authentifizierung mit Benutzername und Passwort. Wenn sich die Standards für den Hashwert geändert haben, wird dieser neu erzeugt und gespeichert.

- **function** `verifyLogin($username, $password, $rehashIfNeed =` **true**`)`

Autologin Einen Autologin-Zugang erstellen, prüfen bzw. löschen.

- **function** `makeAutologin($id)`
  Liefert für eine gültige Benutzer-ID einen Schlüssel, der zusammen mit dem Benutzernamen als Cookie gespeichert werden kann, um den Autologin zu ermöglichen.
- **function** `verifyAutologin($username, $key)`
  Prüft, ob der Schlüssel gültig ist.
- **function** `cancelAutologin($id)`
  Entfernt den Autologin-Zugang.
- **function** `updateAutologin($id, $key)`
  Erzeugt einen neuen Schlüssel; damit wird der alte (Parameter `$key`) ungültig.[11]

[11] Prinzip des „One-Time-Tokens“, siehe 5.4 auf Seite 28

Zurücksetzen des Passwortes Das bekannte „Passwort vergessen“. Dazu wird ein Schlüssel erzeugt.[12] Wer diesen kennt, kann das Passwort zurücksetzen bzw. den Vorgang abbrechen.

- **function** `makePasswordReset($username)`
  Liefert für einen gültigen Benutzernamen einen Schlüssel zum Zurücksetzen des Passwortes.
- **function** `verifyPasswordReset($username, $key)`
  Prüft, ob der Schlüssel gültig ist und der Benutzer damit sein Passwort zurücksetzen darf.
- **function** `cancelPasswordReset($username, $key)`
  Entwertet den Schlüssel zum Zurücksetzen des Passwortes und bricht damit den Vorgang ab.

4.4.5 Authorizer

Mit der Klasse `Authorizer` wird die Zugriffskontrolle umgesetzt. Die Klasse greift auf die Datenbank zu. Das Interface ist relativ einfach und besteht (inklusive Konstruktor) nur aus vier Methoden:

- **function** `__construct($dbh, $session, $pagePath = null)`
  Dies ist der Konstruktor der Klasse. Er erwartet ein `DBAccess`- und ein `Session`-Objekt und den Pfad zur aktuellen Webseite. Die Session wird benötigt, um auf den momentan angemeldeten Benutzer zugreifen zu können. Wenn der Pfad angegeben ist, beziehen sich die Methodenaufrufe `canCurrentUserAccessPage(null)` und `hasCurrentUserRight($subject)` auf ebendiese Seite.
- **function** `canCurrentUserAccessPage($pagePath = null)`
  Über diese Methode wird festgestellt, ob der momentan angemeldete Benutzer auf die angegebene Seite grundsätzlich zugreifen darf. Wenn der Parameter nicht angegeben wird (Standardwert `null`) wird die dem Konstruktor übergebene Seite herangezogen. Wenn dort ebenfalls keine Seite angegeben wurde, ist das Ergebnis `false`.
- **function** `hasCurrentUserRight($subject)`
  Damit wird festgestellt, ob der momentan angemeldete Benutzer das mit `$subject` angegebene seitenspezifische Recht besitzt. Die aktuelle Seite muss dazu dem Konstruktor übergebene worden sein – ansonsten ist der Rückgabewert `false`.
- **function** `hasCurrentUserGeneralRight($subject)`
  Damit wird festgestellt, ob der momentan angemeldete Benutzer das mit `$subject` angegebene allgemeine (nicht-seitenspezifische) Recht besitzt.

Die Funktionen geben nur `true` zurück, wenn der momentan angemeldete Benutzer das erforderliche Recht eindeutig besitzt. Im Zweifel wird immer `false` zurückgegeben. Hier noch eine kurze Beschreibung der Parameter `$pagePath` und `$subject`:

- Der Parameter `$pagePath` ist der absolute Pfad der Datei, beginnend mit einem Schrägstrich (`/`, „Root“). Der Pfad gilt ab dem „Document Root“ der Webanwendung und ist innerhalb dieser für jede Ressource eindeutig. Liegt die Webanwendung unter `https://example.com/fbweb` so ist ein Beispiel für einen Pfad `/password/change.php`. Die Konkatenation von URL (der Webanwendung) und Pfad ergibt die vollständige URL der Datei.

[12] Dieser Schlüssel wird in spezielle Links eingebaut und dem Benutzer so per E-Mail geschickt.

– Der Parameter `$subject` ist der Name eines Rechts. Hierfür kann sich der Administrator im Grunde beliebige Namen ausdenken, es sind alle Zeichen erlaubt.

Die Implementierung ist relativ aufwändig: Die Berechtigungen werden aus der Datenbank ermittelt. Dabei wird auf sieben Tabellen (siehe 4.3.4) zurückgegriffen. Dies soll möglichst effizient durchgeführt werden.

Außerdem können die Methoden auf mehrere Weisen fehlerhaft aufgerufen werden. Dabei kann es sich um Konfigurationsfehler (in der Datenbank) oder Implementierungsfehler (des Programmierers, der das `Authorizer`-Objekt verwendet) handeln. Zum Beispiel

– könnte ein Recht abgefragt werden, das in der Datenbank gar nicht existiert.

– könnten seitenspezifische Rechte abgefragt werden, obwohl beim Konstruktoraufruf keine Seite angegeben wurde.

Solche Fehler werden protokolliert, da Tests hierzu sehr aufwändig wären.

4.4.6 PasswordPolicy

Die `PasswordPolicy` fasst die Anforderungen an ein gültiges Passwort zusammen. Auf diese Regeln wird beim neu Festlegen bzw. Ändern des Passwortes zurückgegriffen. Dadurch kann eine gewisse Qualität des Passwortes hinsichtlich Sicherheit erzwungen werden. Neben minimaler und maximaler Länge des Passwortes und der Gültigkeitsdauer in Tagen gibt es eine Beschreibung der Regeln. Diese kann dem Benutzer präsentiert werden, bevor er sein Passwort festlegt. Schließlich wird das Passwort mit der Methode `checkPassword` überprüft. Entspricht es nicht den Anforderungen, können die Fehlermeldungen mit den Methoden `getErrorText` und `getErrorTextArray` abgefragt werden.

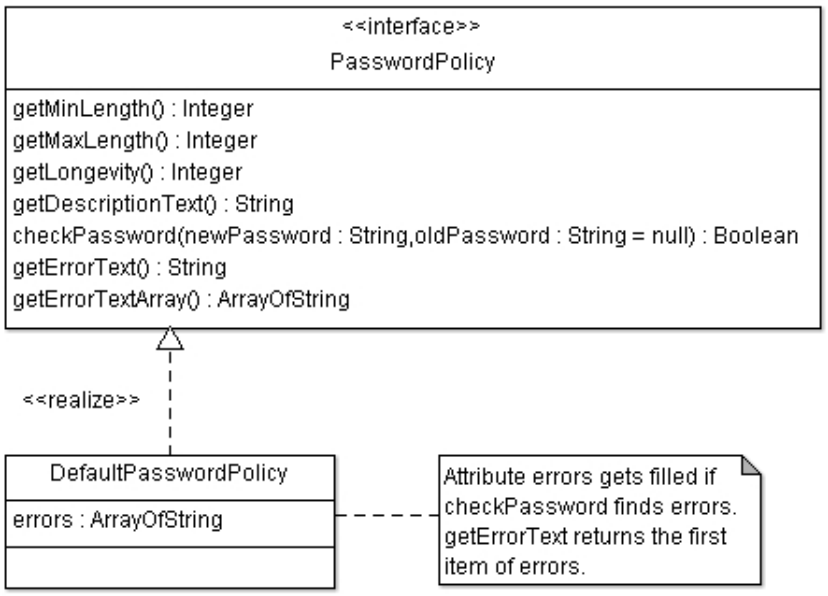

Abbildung 5: Der Ausschnitt aus einem Klassendiagramm zeigt das Interface `PasswordPolicy` und die implementierende Klasse `DefaultPasswordPolicy`.

4.4.7 Redirector

Diese Hilfsklasse bietet u. a. die statische Methode `static function redirect($url, $base = self::AUTO_DETECT_URL);`

Damit kann in einer Zeile korrekt auf eine andere Seite umgeleitet werden. Realisiert wird dies über den HTTP-Header `Location` mit `header('Location: '. $url)` und einem darauffolgendem `exit`. Dabei gibt es aber zwei Schwierigkeiten, welche auch der Grund dafür sind, das Umleiten in eine Klasse auszulagern:

1. Die URL im HTTP-Header `Location` muss gemäß RFC 2616, Abschnitt 14.30 eine vollständiger absoluter URI sein, also im Format `http://server/path/to/page`. Die Funktion `redirect` nimmt dem Entwickler das Zusammensetzen einer vollständigen URL ab.

2. In unterschiedlichen Situationen können unterschiedliche Arten von Werten für den Parameter `$url` angebracht sein. Mal wäre ein relativer Pfad praktisch, mal aber auch ein absoluter Pfad (ab dem Document Root oder ab `/fbweb`, also dem „Document Root“ der Webanwendung) oder doch eine vollständige URL.
   Durch den Parameter `$base` kann eingestellt werden um welchen Typ von URL es sich handelt. Mögliche Werte für diesen Parameter sind die Klassenkonstanten `AUTO_DETECT_URL`, `COMPLETE_URL`, `FROM_HOST_URL`, `FROM_BASE_URL` und die magische Konstante `__DIR__` für relative Pfade.

Einige Beispielaufrufe zeigen das einfache Umleiten mit der Klasse `Redirector`. An dieser Stelle sei noch

einmal darauf hingewiesen, dass `redirect` neben der Funktion `header` auch `exit` aufruft und damit auch das aufrufende Skript beendet.

```
<?php
Redirector::redirect('/login.php');
Redirector::redirect('http://example.com/');
Redirector::redirect('../index.php', __DIR__);
Redirector::redirect('/index.php', Redirector::FROM_BASE_URL); // -> /fbweb/index.
    php
Redirector::redirect('/index.php', Redirector::FROM_HOST_URL); // -> /index.php
?>
```

4.4.8 HTMLCode

`HTMLCode` ist ein Interface, das nur eine Methode deklariert:
`function` `html();` (der Rückgabewert ist ein String)

Klassen, die dieses Interface implementieren, sind dafür gedacht HTML-Code zu erzeugen. Das ist gerade dann sinnvoll, wenn derselbe HTML-Code an mehreren Stellen gebraucht wird, aber auch nützlich um komplexere dynamische Seiten zu vereinfachen. Durch das gemeinsame Interface wird erreicht, dass über die gesamte Webanwendung hinweg ein einheitlicher Standard verwendet wird. Ansonsten könnten leicht viele unterschiedlich benannte und anzuwendende Methoden enstehen, wie `makeHtml()`, `printHTML()`, `echoHTMLCode()`.

Das folgende Listing zeigt, wie das Konzept angewendet wird:

```
<?php
$someHTML = new ConcreteHTMLCode(...);
echo $someHTML->html();
?>
```

Ein Beispiel ist die spezielle Eingabemethode des Benutzernamen, siehe Abbildung 6. Dort implementieren die Klassen `EmailWithDomainInput` und `UsernameInput` das gemeinsame Interface `HTMLCode`. Dabei ist `UsernameInput` eine Spezialisierung, die durch Komposition und Delegation an ein `EmailWithDomainInput` Objekt zustandekommt.

Da nur die von `HTMLCode` abgeleiteten Klassen wissen, was z. B. der Name (`name`-Attribut) eines `input`-Elements ist, ist es sinnvoll eine entsprechende „Getter"-Methode hinzuzufügen. So erhält man die

Abbildung 6: Ein Beispiel für die Verwendung des Interfaces `HTMLCode`: Dieses zusammenhängende Eingabeelement wird dynamisch erzeugt und an mehreren Stellen in der Webanwendung eingebunden.

eingegebene E-Mail-Adresse beim `UsernameInput` Objekt `$input` nach einem POST durch `$email = $input->getUsername($_POST);`

Das Argument `$_POST` muss übergeben werden, weil diese Klasse nur das `input`-Element, ohne das umgebende `form`-Element darstellt. Daher kann sie nicht wissen, ob der Inhalt mit POST oder GET abgeschickt wurde.

5 Diskussion

Im Folgenden werden noch einige Fragen erörtert, die sich beim Entwurf aufdrängten.

5.1 Active Directory Authentication

Alle Mitarbeiter bei Siemens haben einen Eintrag im Active Directory der Firma und damit eine Kennung zum Anmelden am CAT-Client. Nun wäre es natürlich praktisch, wenn die Anmeldung bei der Webanwendung eben mit dieser Kennung möglich wäre. Grundsätzlich gibt es zu diesem Zweck das Lightweight Directory Access Protocol (LDAP), über das auch von PHP auf ein Active Directory zugegriffen werden kann. Bei Siemens ist eine solche direkte Verbindung aber nicht möglich. Stattdessen gibt es – als weitere Abstraktionsschicht – den Siemens Entitlement Service[10].

Mit GetAccess stellt der Siemens Entitlement Service ein Paket bereit, das internen Webanwendungen eine Authentifizierung der Benutzer via Acitve Directory ermöglicht. Es werden zwei Möglichkeiten der Authentifizierung angeboten:

1. Die Authentifizierung per Smart Card PKI (Firmenausweis und Personal Identification Number (PIN)).
2. Automatischer Login über die aktuelle Windows-Benutzersitzung.

Vorteile sind offensichtlich: Der Benutzer muss sich keine neue Kennung merken und kann sich auf bequeme Arten anmelden, wie er es auch von anderen Intranet-Anwendungen gewohnt ist. FBWeb müsste sich weder um die Authentifizierung noch das Speichern der Passwörter kümmern.

Leider sprechen auch einige Gründe gegen die Nutzung des Siemens Entitlement Service. Zum einen ist der der Computer, auf dem FBWeb läuft, kein angemieteter, professionell betriebener Server, sondern ein normaler PC mit einer XAMPP-Installation. Aus diesem Grund ist es unwahrscheinlich, dass der Server die Anforderungen des Siemens Entitlement Service erfüllt und überhaupt zugelassen wird. Desweiteren gibt es einige Nachteile und Risiken:

1. Hoher Installationsaufwand – dazu wäre außerdem Vollzugriff auf den Server nötig.
2. Spärliche Dokumentation.
3. Administrator notwendig – dieser muss mit der Installation und Konfiguration von GetAccess vertraut sein.
4. Eigener Support notwendig, da Siemens Entitlement Service keinen „End-User Support“ bietet.
5. Entwickler notwendig, falls aufgrund von Änderungen in GetAccess Anpassungen in der Webanwendung notwendig werden.

6. Geplante oder ungeplante Ausfallzeiten des Siemens Entitlement Service könnten zu erheblichen Arbeitsausfällen in der Abteilung führen. Bei einem eigenen Loginsystem bleibt die Kontrolle immerhin in der Abteilung.

7. Es könnten bei fortschreitender Integration Probleme auftreten, die aufgrund der knappen Dokumentation vielleicht nicht vorhersehbar waren. Auch der Aufwand lässt sich im Vorhinein schwer abschätzen.

Aus diesen Gründen wurde (zumindest vorläufig) gegen GetAccess des Siemens Entitlement Service und für die PHP-Implementierung eines Loginsystems entschieden.

5.2 Passwort-Hashing API

phpass vs. Secure Password Hashing API (PHP 5.5) – beide bieten Funktionen für sicheres Passwort-Hashing in PHP-Anwendungen und schließen damit eine Lücke, die PHP jahrelang offengelassen hat: Viele Entwickler haben aus Unwissenheit oder Bequemlichkeit zum Passwort-Hashing die Funktion `md5` verwendet – immerhin. Aber einerseits sind dabei die Passwörter nicht gesalzen, andererseits ist MD5 ein effizienter Hash-Algorithmus. Beides begünstigt Brute-Force-Attacken.

phpass Die Public-Domain-Bibliothek phpass der Firma Openwall verwendet vorzugsweise den bcrypt-Algorithmus und erzeugt automatisch gesalzene Passwörter.

Secure Password Hashing API Die Secure Password Hashing API geht auf ein „Request for Comments“ in der PHP-Community zurück [3]. Ziel war es, PHP um eine sichere Passwort-Hashing API zu bereichern. Spezifiziert sind wie bei phpass die Verwendung geeigneter Hashfunktionen und Salz [4]. Die API soll in PHP 5.5 einfließen, welches vorraussichtlich im ersten Quartal 2013 veröffentlicht wird. Eine PHP-Implementierung der API gibt es bereits [9].

Es steht also die Wahl zwischen phpass und der Secure Password Hashing API. Aus folgenden Gründen wurde dann klar für die Secure Password Hashing API von PHP entschieden:

- Verwendung eingebauter Standardfunktionen statt externer Bibliothek
  Aktuell wird bei FBWebnoch die PHP-Implementierung verwendet, sobald aber PHP auf Version 5.5 aktualisiert wird, kann diese Quelldatei einfach gelöscht werden.

- Gute Dokumentation
  Offizielle Dokumenation auf php.net/password und detaillierte Besprechung im „Request for Comments“.

- Einfachere Verwendung
  Nur vier, fast selbsterklärende Funktionen (hash, verify, needs_rehash, get_info) im Gegensatz zu neun Methoden der phpass-Klasse.

- Zukunftssicherer
  Die „Kosten“ der Hashfunktion können per Parameter eingestellt (und damit erhöht) werden.

5.3 Benutzername

Es gibt einige Anforderungen an den Benutzernamen, sowohl von Seiten der Benutzer als auch von Seiten des Systems. Zu diesen Anforderungen zählen: Der Benutzername muss

- einfach zu merken und nicht zu lang sein (Benutzerfreundlichkeit).
- für *jeden* Benutzer definiert sein (auch z. B. für Praktikanten).
- eindeutig sein, wobei aber zwischen Groß- und Kleinschreibung nicht unterschieden werden soll.
- automatisiert aus den Mitarbeiterdaten übernommen oder generiert werden können.

Eine weitere Idee war, durch einen nicht öffentlich zugänglichen Benutzernamen die Sicherheit des Benutzerkontos zu erhöhen. Durch Social Engineering bekäme ein Angreifer einzelne *Benutzernamen* zwar recht einfach heraus, aber eine Hürde wäre es dennoch.

Bei Siemens kommen für solche nicht-öffentlichen Benutzernamen grundsätzlich Personalnummer und Windows-Anmeldename in Frage. Während die Personalnummer für jeden Mitarbeiter definiert ist, hat ein Praktikant aber nicht unbedingt einen eigenen Windows-Anmeldenamen. Ein weiterer Nachteil ist, dass viele Mitarbeiter diese Daten nicht auswendig wissen und sie sich unter Umständen auch nicht leicht herausfinden lassen (z. B. steht die Personalnummer auf der letzten Gehaltsabrechnung, die aber zu Hause liegt).

Ein frei wählbarer Benutzername wird auch nicht von jedem Mitarbeiter als benutzerfreundlich empfunden: Zuerst muss sich im Zuge der Prozedur „Benutzerkonto anlegen“ für einen Namen entschieden werden. Anschließend hat der Benutzer eine weitere Kennung, die er sich merken muss. Zudem lohnt es nicht, viel Arbeit in ein „Benutzerkonto anlegen“-Feature zu investieren, da vergleichsweise selten neue Benutzer hinzukommen. Wegen der Nachteile entfällt also hier die Idee der nicht-öffentlichen Benutzernamen. Als weitere Möglichkeiten bleibt damit nur noch die E-Mail-Adresse.

Die E-Mail-Adresse als Benutzername hat mehrere Vorteile: Sie ist dem Benutzer bekannt, sie ist eindeutig und ist in den Mitarbeiterdaten enthalten. Für Praktikanten, die keine siemens.com-Adresse haben, kann auch eine private E-Mail-Adresse verwendet werden. Dass die Adresse mitunter etwas länger sein kann fällt nicht ins Gewicht, da Benutzeroberfläche, Browser und die Autologin-Funktion dem Benutzer fast alles abnehmen (siehe Login und Abbildung 2). Dem Nachteil an Sicherheit kann mit einer geeigneten Passwort Policy begegnet werden.

Nach diesen Überlegungen wurde für die E-Mail-Adresse als Benutzername entschieden, was nun auch die benutzerfreundlichste Lösung darstellt.

5.4 Autologin

Ein möglichst sicheres und benutzerfreundliches Autologin-Feature zu implementieren ist nicht trivial. Es gibt viele unterschiedliche Möglichkeiten, den verschiedenen Anforderungen gerecht werden. Als „Autologin-Zugang“ wird im Folgenden die aktivierte automatische Anmeldung in einem bestimmten Browser bezeichnet, wenn also nicht mehr nach Benutzername und Passwort gefragt wird.

5.4.1 Allgemeines

Technisch wird ein Autologin-Feature üblicherweise realisiert, indem die Webanwendung den Browser veranlasst, ein „One-Time-Token“ als Cookie zu speichern [8]. Anhand dieses Schlüssels (Autologin-Key) kann die Webanwendung den Client wiedererkennen.

Grundsätzlich ist eine solche Funktion ein gewisses Sicherheitsrisiko: Ein Angreifer, der es schafft das Cookie zu stehlen, kann eventuell Zugang zur Webanwendung bekommen, ohne die tatsächlichen Anmeldedaten zu kennen. Man muss also abwägen. Bei Banken z. B. wird man diese Funktion nie antreffen. Bei FBWeb wurde aber entschieden, den Autologin zuzulassen. Die Mitarbeiter würden sich zu sehr ärgern, wenn sie täglich morgens und womöglich auch nach dem Mittagessen Benutzername und Passwort eingeben müssten[13]. Um Autologin dennoch möglichst sicher zu gestalten hat man rein technisch schon einige Möglichkeiten:

1. Zugriff auf das Cookie nur über verschlüsselte Verbindungen (HTTPS) zulassen.
2. Zugriff auf das Cookie nur per Hypertext Transfer Protocol (HTTP) zulassen und nicht per Skriptsprachen wie JavaScript.
   Diese beiden Beschränkungen werden durch die (aktuell) letzten Parameter der PHP-Funktion `setcookie` festgelegt: `secure` und `httponly`
3. Das Token (eine Zeichenkette, im Folgenden auch Autologin-Key) sollte ein Zufallswert sein und genügend Stellen besitzen, um Brute-Force-Attacken vorzubeugen.
4. Der Autologin-Key sollte bei jeder Anmeldung automatisch geändert werden.
5. Neben dem Autologin-Key können serverseitig weitere Client-Daten gespeichert werden, die beim Autologin ebenfalls übereinstimmen müssen:
   (a) IP-Adresse des Clients
   (b) Internet-Hostname des Clients: `gethostbyaddr($ip_addr)`
   (c) Weitere Informationen über Browser/Client aus dem „Usager Agent“-Header[14]: `$_SERVER['HTTP_USER_AGENT']`

Die Vorschläge von Punkt 5 können sich aber auch negativ auf die Benutzerfreundlichkeit auswirken. Wenn sich im Firmennetzwerk z. B. IP-Adresse oder sogar Name des Clients häufig ändern, hat das Autologin-Feature seinen Zweck verfehlt. Das selbe gilt, wenn es häufig Aktualisierungen für Browser oder andere Softwarekomponenten gibt, die sich in einem veränderten „User Agent“-String widerspiegeln.

Der Entwurf eines Autologins ist noch deutlich mehr ist, als die Umsetzung der oben genannten technischen Details. Es folgen einige, aus Benutzersicht wichtige Fragen. Diese schlagen sich auch alle im Entwurf nieder.

1. Wie lange bleibt der Autologin-Zugang bestehen?
2. Was ist, wenn ich mich an mehreren verschiedenen Computern/Browsern anmelde?

[13] Beim CAT-Client lässt sich das Speichern von Benutzernamen und Kennwörtern im Internet Explorer nicht aktivieren.
[14] Eine Zeichenkette, die per HTTP im Header (`User-Agent:`) mitübertragen wird und Informationen zum Client und Browser enthält.

   (a) Bleiben dann die anderen Autologin-Zugänge bestehen?

   (b) Laufen die Autologin-Zugänge alle gemeinsam ab, oder jeder einzeln?

   (c) Wenn ich das Passwort ändere, werde ich dann überall aufgefordert, mich neu anzumelden (oder überleben das die Autologin-Zugänge)?

   (d) Kann ich sehen, welche anderen Computer/Browser einen Autologin-Zugang haben? Und kann ich diese deaktivieren?

3. Was ist, wenn jemand meinen Autologin-Zugang klaut?

   (a) Geht das überhaupt?

   (b) Bekomme ich das mit?

   (c) Werde ich dann noch automatisch angemeldet?

   (d) Was wird im Nachhinein dagegen getan?

Um alle diese Fragen positiv beantworten zu können, wäre einiges an Entwurfs- und Implementierungsarbeit notwendig. Zum Beispiel sollte (in einer extra Datenbanktabelle) für jeden Autologin-Zugang nicht nur das aktuelle, sondern auch die nicht mehr gültigen Token gespeichert werden. Dazu folgendes Szenario:

Ein Dieb klaut den aktuellen Autologin-Key (d. h. kopiert die Cookies) und meldet sich damit von einem anderen Computer aus an. Anschließend will sich der echte Benutzer an seinem Computer wieder anmelden. Er wird aufgefordert, Benutzername und Passwort einzugeben, da sein Schlüssel ungültig ist. Dieser wurde geändert, als sich der Dieb damit anmeldete.

Wenn also nicht mehr gültige Schlüssel gespeichert bleiben, kann die Webanwendung die Unstimmigkeit feststellen und auch den Dieb auffordern, sich mit Benutzername und Passwort zu authentifizieren. Damit kann der Dieb den Zugang vielleicht nur ein paar Stunden nutzen, statt einem Monat (letzteres ist abhängig von der Gültigkeitsdauer des Autologins).

5.4.2 Die Lösung in FBWebAuth

Im konkreten Fall wurde für eine einfache Lösung entschieden: Pro Benutzer gibt es immer höchstens einen Autologin-Zugang. Das heißt, der Benutzer kann sich nur an einem Computer mit einem bestimmten Browser automatisch anmelden lassen. Meldet sich der Benutzer in einem anderen Browser mit seiner Kennung und einem Häkchen bei „Remember me“ an, wird er von da an in diesem Browser automatisch angemeldet – der Autologin-Zugang von vorher gilt dann nicht mehr.

Realisiert ist dies durch zwei Spalten in der Tabelle `mitarbeiter`: `autologin_key` enthält den aktuellen Schlüssel, der auch clientseitig in einem Cookie gespeichert ist. In `autologin_expire` steht das feste Ablaufdatum des Autologin-Zugangs. Dieses wird beim Erstellen des Zugangs z. B. auf „Jetzt plus 30 Tage“ gesetzt.

Das Konzept reicht bei FBWeb in der Praxis vollkommen aus. Mitarbeiter haben in der Regeln ein Firmennotebook und einen Standardbrowser. Wenn sie von Zuhause arbeiten, verwenden sie das selbe Notebook. Manche Mitarbeiter haben weitere Computer z. B. ein SIMATIC Field PG (ein sogenanntes Programmiergerät). Diese werden aber nicht für die Fachberatung, sondern für Anwendungen im Automatisierungsumfeld verwendet.

Ein weiterer Vorteil der Lösung mit maximal einem Autologin-Zugang pro Benutzer, ist neben der Einfachheit auch die relativ hohe Sicherheit. Wenn der Zugang kompromittiert, also das Cookie gestohlen wurde, muss sich der wahre Benutzer neu anmelden. An dieser Stelle kann ihm ein Warnhinweis angezeigt werden. Durch das Anmelden mit Häkchen bei „Remember me“ wird außerdem der Autologin-Zugang neu angelegt, was dazu führt, dass der Angreifer dann keinen Zugang mehr hat.

5.5 Portierung in andere Webanwendung

Ziel von FBWebAuth war es, ohne gravierende Änderungen auch in anderen abteilungsinternen Webanwendungen eingebaut werden zu können. Dass trotzdem an vielen Stellen kleinere Anpassungen nötig sind lässt sich nicht verhindern. Zum Beispiel müssen Pfadangaben in `require`-Anweisungen geändert werden, wenn die Klassen in einer neuen Verzeichnisstruktur angeordnet werden. Außerdem müssen SQL-Abfragen ausgebessert werden, wenn sich in den Datenbanken die „User“-Tabellen unterscheiden. Durch die folgende Liste werden die wichtigsten Arbeiten bei der Portierung identifiziert.

- Pfadanpassungen in `require`-Anweisungen. Dies betrifft also die Strings hinter den `require_once`-Befehlen am Anfang aller Dateien.
- Anpassungen in den SQL-Abfragen. Hierzu ein Beispiel:
  `"SELECT␣id,␣pw_hash␣FROM␣mitarbeiter␣WHERE␣msmail␣=␣?"`
  Diese Abfrage bezieht sich auf die Spalten `id`, `pw_hash` und `msmail` der Tabelle `mitarbeiter`. Für eine andere Webanwendung muss die Abfrage wahrscheinlich ausgebessert werden und z. B. wie folgt lauten:
  `"SELECT␣user_id,␣pw_hash␣FROM␣user␣WHERE␣email␣=␣?"`
- Abhängigkeit von Konfigurationsdatei (`/Config.inc.php`). Hier sind einige Konstanten definiert, die von der Zugriffskontrolle verwendet werden. Dabei handelt es sich allerdings nur um triviale Werte wie „Name der Webanwendung“, „Verzeichnis, in dem die Webanwendung liegt“, ...
- Abhängigkeiten von Bibliotheksfunktionen aus FBWeb, speziell `html_lib.php`. Darin befinden sich allgemeinen Funktionen, z. B. zur Cross-Site-Scripting (XSS)-sicheren Ausgabe von Variablen. Diese Datei oder ihr Quellcode muss also übernommen werden, wobei eventuell weitere Pfadanpassungen nötig sind.
- Abhängigkeit von der Passwort-API – entweder ist diese ist schon nativ in PHP integriert, oder es muss eine PHP-Implementierung eingebunden werden. Für Details hierzu, siehe 5.2 auf Seite 27.

Möglicherweise treten bei der tatsächlichen Portierung auch nicht vorhergesagte Probleme auf – ohne diesen Vorgang einmal durchgespielt zu haben, lässt sich kaum das Gegenteil behaupten. Lösbar werden solche Schwierigkeiten auf jeden Fall sein.

6 Schluss

Der aktuelle Stand des Projekts ist nun wie folgt: Loginsystem und Zugriffskontrolle sind bereit, in FBWeb integriert zu werden. Features, wie die Aufforderung zum Ändern des Passwortes (bei Erreichen des Ablaufdatums) oder eine Benutzeroberfläche zum Verwalten von Rollen und Rechten müssen noch programmiert

werden. Diese Arbeiten haben jedoch eine geringere Priorität. Gerade die GUI zur Verwaltung der Zugriffskontrolle wird anfangs nicht benötigt, da die Rollen und Rechte nicht einzeln festgelegt werden, sondern stapelweise vom bestehenden System übernommen und per SQL-Anweisungen eingespielt werden. Zudem können diese Features auch gut implementiert werden, wenn das restliche System schon produktiv läuft.

Da die interne Architektur von FBWeb nun aber sowieso grundlegend überarbeitet wird und die Seiten an das Corporate Design der Siemens AG angepasst werden, wird FBWebAuth jetzt nicht mehr in die bestehenden alten PHP-Seiten integriert. Stattdessen wird die Zugriffskontrolle von Anfang an in die neuen Seiten eingebaut.

Vorher sind aber noch ein paar Arbeiten am Produktivsystem notwendig. Das Zertifikat für den HTTPS-Betrieb muss am Server erstellt werden und in der MySQL-Datenbank muss die Zeichenkodierung auf UTF-8 und die Engine von MyISAM auf die modernere InnoDB (die u. a. referentielle Integrität bietet) umgestellt werden.

Abschließend lässt sich sagen, dass das Projekt FBWebAuth bisher wie geplant und erfolgreich verlief. Nach Abschluss der Modernisierung von FBWeb liegt damit die Webanwendung nicht mehr ungeschützt im Intranet, sondern ist dank Authentifizierung und Autorisierung nur noch für berechtigte Mitarbeiter bedienbar.

Glossar

Autologin-Zugang
: Die aktivierte automatische Anmeldung in einem bestimmten Browser, wenn also nicht mehr nach Benutzername und Passwort gefragt wird.

bcrypt
: Eine kryptologische Hashfunktion, speziell für das Hashing von Passwörtern.

CAT-Client
: Firmenrechner der Siemens AG mit vorinstalliertem Microsoft Windows und bestimmten Dienstprogrammen.

HTTPS
: Hypertext Transfer Protocol Secure, d. h. also verschlüsseltes HTTP.

Passwort Policy
: Anforderungen an Passwörter hinsichtlich Stärke und Lebensdauer.

PHP
: ,,PHP: Hypertext Preprocessor“ (rekursives Akronym), ursprünglich ,,Personal Homepage Tools“.

PHPDoc
: Eine Möglichkeit zur Dokumentation von PHP-Quellcode; eine Anpassung von Javadoc für PHP.

PHPUnit
: Ein Framework für PHP-Modultests.

Siemens Entitlement Service
: Firmeninfrastruktur für webbasierte Authentifizierung von Mitarbeitern, Geschäftspartner und Kunden der Siemens AG.

UTF-8
: Unicode Transformation Format, ein Standard zur Kodierung von Unicode Zeichen.

WAMP
: Serverinstallation, bestehend aus Windows, Apache, MySQL, PHP/Perl/Python.

XAMPP
: Eine Distribution von Apache, MySQL, PHP und Pearl – verfügbar für verschiedene Betriebssysteme.

Abkürzungsverzeichnis

API	Application Programming Interface.
CSS	Cascading Style Sheets.
EER	Enhanced Entity-Relationship.
ER	Entity-Relationship.
GUI	Graphical User Interface.
HTML	Hypertext Markup Language.
HTTP	Hypertext Transfer Protocol.
ID	Identifikator.
IDE	Integrated Development Environment.
IP	Internet Protocol.
LDAP	Lightweight Directory Access Protocol.
MD5	Message Digest 5.
PDF	Portable Document Format.
PDO	PHP Data Objects.
PDT	PHP Development Tools.
PIN	Personal Identification Number.
PKI	Public-Key-Infrastruktur.
SHA	Secure Hash Algorithm.
SQL	Structured Query Language.
URA	Universal Remote Access.
URL	Uniform Resource Locater.
VCS	Version Control System.
VPN	Virtual Private Network.
XSS	Cross-Site-Scripting.

Literatur

[1] heise Security : PHP 5.5 soll Passwort-Schlamperei eindämmen
http://heise.de/-1707355, 2013-02-19

[2] heise Security : Cracker-Bremse
http://heise.de/-1253931, 2013-02-19

[3] The PHP Group : Request for Comments: Adding simple password hashing API
https://wiki.php.net/rfc/password_hash, 2013-02-19

[4] nikic : The new Secure Password Hashing API in PHP 5.5
https://gist.github.com/nikic/3707231#file-password_hashing_api-md, 2013-02-19

[5] Openwall : Portable PHP password hashing framework
http://openwall.com/phpass/, 2013-02-19

[6] MySQL : MySQL Workbench – Design, Develop, Administrator
http://www.mysql.de/products/workbench/, 2013-02-20

[7] Niels Provos and David Mazières, The OpenBSD Project : A Future-Adaptable Password Scheme
http://static.usenix.org/events/usenix99/provos.html, 2013-03-07

[8] The Open Web Application Security Project : PHP Security Cheat Sheet
https://www.owasp.org/index.php/PHP_Security_Cheat_Sheet, 2013-03-14

[9] Anthony Ferrara : password_compat
https://github.com/ircmaxell/password_compat, 2013-03-15

[10] Siemens AG : Siemens Entitlement Service
https://entitlement.siemens.com/, 2013-04-01 (nur Siemens-Intranet)

Anhang

PDO/DBAccess als neue Datenbankschnittstelle

Was ist PDO?

PHP Data Objects (PDO) ist eine Datenbankschnittstelle für PHP, genau wie die `mysql_*`-Funktionen und `mysqli`. PDO ist für Webanwendungen „State of the Art“, weil es die modernste DB-Schnittstelle ist, eine schöne und konsistente API bietet. PDO ist voll objektorientiert. PDO ist nicht auf MySQL-Datenbanken beschränkt, sondern dient als Schnittstelle zu beliebigen Datenbanken, wobei aber nicht auf datenbank-spezifische Features verzichtet werden muss.

Wie ist das Design von diesem objektorientierten PDO?

Es gibt zwei Klassen: `PDO` und `PDOStatement`. Ähnlich wie bei `mysqli` repräsentieren Instanzen der Klasse `PDO` eine Datenbankverbindung. Durch Methoden wie `query` und `exec` bzw. `prepare` können dann Datenbankabfragen abgesetzt bzw. vorbereitet (→ Prepared Statements) werden.
Objekte der Klasse `PDOStatement` werden von `query` und `prepare` zurückgegeben. Sie repräsentieren den „Result Set“ der Abfrage. Bei Prepared Statements müssen – bevor die Ergebnisse bereitstehen – natürlich noch die Parameter gesetzt und die Abfrage ausgeführt werden. Auch dafür hat PDOStatement entsprechende Methoden: `bindParam`, `bindValue`, `execute`.

Was ist DBAccess?

DBAccess ist eine eigene Klasse (von uns), die von PDO abgeleitet ist (OOP). Das heißt sie kann alles, was PDO kann aber noch mehr. Dabei ging es aber nicht darum viele Convencience Methoden hinzuzufügen! Denn auch DBAccess soll nicht mehr sein, als unsere grundlegende DB-Schnittstelle.

Was kann DBAccess mehr als PDO?

Zum einen fügt DBAccess eine Methode `count` hinzu. Damit lässt sich die Anzahl von Datensätzen bestimmen, die bestimmte Kriterien erfüllen.
Des Weiteren werden folgende Methoden „überladen“ um Variablen sicher in SQL-Statements einzusetzen. Damit wird SQL-Injection verhindert.[1]

1. `query` → `pquery`
2. `exec` → `pexec`
3. `count` → `pcount`

Das „p“ steht für „parameterized“ und bedeutet eben, dass die SQL-Statements in der Form `SELECT * FROM tbl WHERE id = ?` eingegeben werden. Die Variablen, die für die Fragezeichen (?) eingesetzt werden sollen, werden einfach als weitere Parameter übergeben.

Wo ist die Dokumentation zu DBAccess?

Die Dokumentation ist direkt im Quellcode DBAccess.php als PHPDoc. Diese Dokumentation bezieht sich häufig auf die Dokumentation von PDO: http://php.net/pdo

[1] SQL-Statements dürfen nie so aussehen: `'SELECT * FROM tbl WHERE id = ' . $id`

Darf ich der Klasse DBAcces noch eine Methode hinzufügen?

Nein. Die Idee ist, dass DBAccess wie PDO eine **grundlegende** DB-Schnittstelle ist und bleibt.[2]

Wie benutzt man DBAccess?

Ganz einfach! Beispiel:

```
function daysUntilPasswordExpires($id)
{
    $dbh = new DBAccess();
    $result = $dbh->pquery("SELECT
        (TO_DAYS(pw_expire) - TO_DAYS(NOW())) AS days
        FROM mitarbeiter WHERE id = ?", $id);
    return $result->fetchColumn();
}
```

Für weitere Beispiele kann man einfach im Internet nach PDO suchen.

Wie kann ich DBAccess in einer anderen Webanwendung verwenden?

Abgesehen vom entsprechenden PDO-Treiber (standardmäßig richtig installiert, siehe `phpinfo()`): Das einzige was DBAccess braucht um zu funktionieren sind die Logindaten zur Datenbank. Dazu muss ganz oben beim `include` der Pfad angepasst werden.[3] In der `include`-PHP-Datei müssen folgende Konstanten definiert sein: `DB_HOST`, `DB_NAME`, `DB_USER`, `DB_PASSWORD` und zu guter Letzt:

```
define('PDO_DSN', 'mysql:host='.DB_HOST.';dbname='.DB_NAME);
```

[2] Na gut, wenn du z.B. eine **super API** anzubieten hättest um **ganz allgemein und datenbankunabhängig** Stored Procedures/Functions aufzurufen, dann kann man vielleicht darüber reden.

[3] Es ist am sichersten, die PHP-Datei mit den Logindaten außerhalb des Document Root zu platzieren.