2017

# Projektdokumentation W2: Serverseitige Anwendungen

## Wintersemester 2016/17

# BM Hotel-Club

DOZENT HERR SASCHA MAHLER
VON MORITZ MÜLLER MTR: 669005 UND
TIM-ALEXANDER BIBA MTR: 670220

# Inhaltsverzeichnis

# Vorwort: Projektidee

Es geht bei unserer Projektidee um eine Hotel-Club-Webanwendung für Worms. Es werden verschiedene Hotels auf der Seite angezeigt die in Worms ansässig sind. Die einzelnen Hotels werden in Bild und Schrift vorgestellt. Zusätzlich gibt es eine Tipp-Funktion auf der Website in der die Benutzer sich über die Hotels austauschen können,
Die Webanwendung ist als Club geplant für Mitglieder die sich die exklusiven Hotels ansehen können.

Eine Ansicht der Hotels ist erst nach einer Registrierung möglich. Zusätzlich bietet die Seite verschiedene Benutzereinstellungen sowie einen Login und einen Logout.

# Kapitel 1: Verwendete Sprachen und Hilfsmittel

## Entwicklungsumgebung und Editoren

Es wurden verschiedene Entwicklungsumgebungen benutzt. Dies ist von der verwendeten Hardware abhängig die benutzt wurde. Als gemeinsamer unterbau einer gleichen Entwicklungsumgebung wurde Ubuntu Desktop 16.10 genutzt.

Für die Entwicklungsumgebung wurde folgende Editoren und IDEs verwendet:

- Jet Brains PyCharm 2016.3 Professional
- Atom Text Editor 1.13.1
- Nano Version 2.5.3
- SublimeText3 Build 3126

## Verwendete Sprachen

Folgende Sprachen wurde im Projekt verwendet:

- Python 3.5.2
- JavaScript 1.11.2
- HTML5
- CSS3
- Jinja2 Templating Language
- PHP 7.0.13

Python 3.5.2 ist einer der aktuellsten Ableger von Python. Die aktuelle Version ist 3.6.
Folgende zusätzliche Pakete wurden installiert:

- Django 1.10.5
- Flask 0.12

- Flask-Login 0.4.0
- Flask-SQLAlchemy 2.1
- SQLAlchemy 1.1.5
- pip 9.0.1
- setuptools 33.1.1
- wheel 0.29.0
- itsdangerous 0.24
- click 6.7
- Werkzeug 0.11.15
- Jinja2 2.9.4
- Markupsafe 0.23
- Flask-MysqlDB 0.2

JavaScript wird als Teil von Bootstrap verwendet sowie auch in Teilen von JQuery zum übertragen der Daten aus dem Registrierungs- und Login-Formularen in die Datenbank.

Nach diversen Probleme mit der Dantebank SQLITE und Flask als Webframework wurde Ende Januar 2017 der komplette Entwicklungs-Stack gegen PHP 7.0.13 mit XAMPP ausgetauscht.

Folgende Softwarepakete beinhaltet XAMPP:

- PHP 7.0.13
- Apache 2
- MariaDB-Server und -Client 10.0.29
- Perl

Trotz Installation durch XAMPP wurde Perl nicht genutzt.

## Verwendetes Frameworks

Im Projekt wurden verschiedene Frameworks verwendet. In einer frühen Phase wurde zuerst Django 1.10.5 verwendet. Dies wurde nach längerer Überlegung verworfen, da der Umfang den Django mitbringt zu groß für das Projekt gewesen wäre.
Deswegen wurde ab Dezember 2016(siehe Protokoll) auf das Webframework Flask 0.12 umgestiegen. Flask ist ein leichtgewichtiges Framework das nur die nötigsten Funktionen mitbringt und mit zusätzlichen Python Paketen erweiterbar ist.

Nach großen Problemen mit Flask und seinen Erweiterungen wurde auf PHP 7.0.13 gewechselt und jegliches Framework verworfen.

## Verwendete Datenbanken

Zuerst wurde aus Einfachheit eine SQLITE3-Datenbank angelegt. Dies wurde im Laufe des Projekts verworfen, da eine Anbindung sich für uns schwierig gestaltete. Es gab Probleme die SQLITE-Datenbank in ein richtiges Format zu exportieren und ein abspeichern von Daten war nicht möglich. Es wurde ab Dezember 2016(siehe Protokoll) auf MariaDB gewechselt, diese wurde mittels XAMPP installiert. Ende Januar 2017 nachdem die gleichen Probleme mit MariaDB und Flask auftraten, Python mit Flask gegen PHP auszutauschen.

Zum Verwenden der Datenbanken wurden verschiedene Programme genutzt. Für PostgreSQL wurde PostgreSQL Studio in Version 2.0 benutzt. Dies ist eine Webanwendung diemit Apache Tomcat zum Verwalten von PostgreSQL ausgeliefert wird.

Für SQLITE3 haben wir SQLite Studio verwendet in Version 3.1.1. Dies ist eine Desktopanwendung zum Verwalten einer SQLITE Datenbank.

Für MariaDB haben wir von XAMPP das Tool PHPMyAdmin in Version 4.6.5.2. Dies ist wie PostgreSQL-Studio eine Webanwendung, die mit Apache2 ausgeliefert wird.

### Weitere Hilfsmittel

Es wurden diverse weitere Hilfsmittel verwendet. Zum einen Bootstrap um die HTML-Templates zu erstellen und eine einheitliche, schlicht gestaltete Oberfläche zu haben. Zum anderen wurde auf JQuery zurückgegriffen um die Formulardaten in die SQLITE-Datenbank zu übertragen.
Nach schwerwiegenden Problemen mit Flask und SQLITE als Software Stack, haben wir uns dafür entschieden die XAMPP-Entwicklungsumgebung zu installieren. Grundsätzlich ausschlaggebend für XAMPP war die einfache Handhabung von PHPMyAdmin und MariaDB, sowie die Verwendung von PHP.

Zum Teilen des Codes haben wir auf GitHub gesetzt. Damit hatten wir Probleme beim Umzug von Django auf Flask. Daraufhin haben wir beim Projektneustart auf die Hochschule Owncloud gesetzt.

# Kapitel 2: Aufbau der Software und Datenbank

### Aufbau der Software

Das Projekt ist in folgender Ordnerstruktur gegliedert:

```
htdocs
├── css
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   ├── bootstrap.min.css.map
│   ├── bootstrap-theme.css
│   ├── bootstrap-theme.css.map
│   ├── bootstrap-theme.min.css
│   ├── bootstrap-theme.min.css.map
│   └── style.css
├── img
│   ├── Asgard_Hotel.gif
│   ├── blur-1867663_960_720.jpg
│   ├── Dom_Hotel.gif
```

```
│   ├── hotel-1330841_960_720.jpg
│   ├── hotel-389256_960_720.jpg
│   ├── keyboard-1621105_960_720.jpg
│   ├── LogoBM.png
│   ├── LogoLang.png
│   ├── maldives-666122_960_720.jpg
│   ├── module_table_bottom.png
│   ├── module_table_top.png
│   └── Park_Hotel.gif
├── inc
│   ├── config.inc.php
│   ├── functions.inc.php
│   └── password.inc.php
├── index.php
├── internal.php
├── js
│   ├── bootstrap.js
│   └── bootstrap.min.js
├── login.php
├── logout.php
├── passwortvergessen.php
├── passwortzuruecksetzen.php
├── register.php
├── settings.php
├── templates
│   ├── error.inc.php
│   ├── footer.inc.php
│   └── header.inc.php
├── tipp.php
└── webalizer
```

In der Ordnerstruktur werden nur die wichtigsten Ordner aufgeführt, da das Dashboard von PHPMyAdmin sehr viele Dateien beinhaltet. Es wurde zusätzlich die Fonts von Bootstrapp entfernt.

Die wichtigsten Dateien sind die index.php, register.php, tipp.php, settings.php und internal.php. Die hier benannten Dateien rufen die anderen PHP-Dateien auf, diese sind ausgelagerte Funktionen, die z.B. den Login oder den Logout übernehmen. Dadurch wurde die Webanwendung entsprechend auf gesplittet und bietet eine gewisse Übersichtlichkeit.

Die index.php-Datei zeigt die Hauptseite an und bietet die Möglichkeit sich einzuloggen oder zu registrieren. Die register.php-Datei ist die Registrierungsseite um auf den internen Bereich der Webanwendung zuzugreifen. Die internal.php-Datei ist die php-Seite die für den internen Bereich da ist. Hier kann sich der Nutzer die einzelnen Hotels ansehen und sich die Beschreibungen durchlesen. Die auf der Tipp.php-Seite können die eingeloggten Benutzer Kommentare zu den Hotels schreiben. Auf der settings.php-Seite können eingeloggte Benutzer Ihre Passwörter, Benutzernamen und E-Mail zurücksetzen.

## Aufbau der Datenbank

Die Datenbank hat folgende Tabellen:

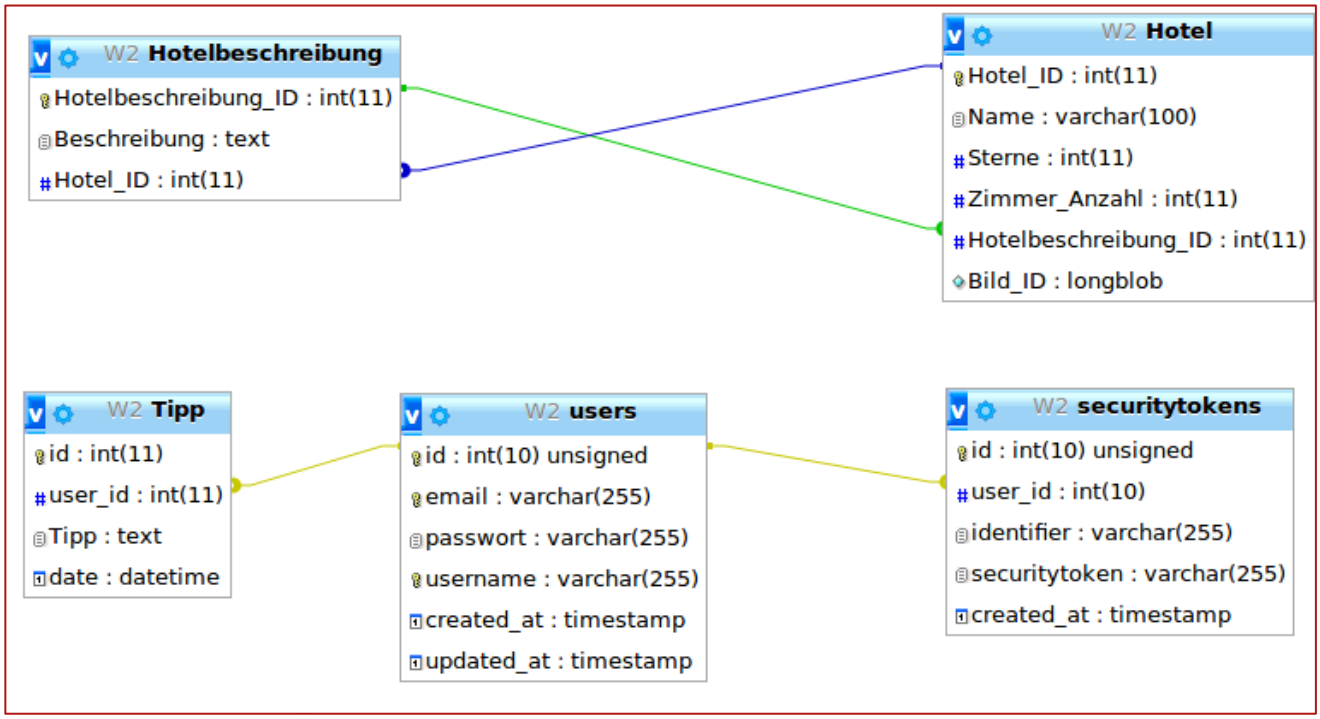


*Abbildung 1: Datenbank Übersicht*

Die Datenbank besteht aus den Tabellen Users, Securitytokens, Tipp, Hotel und Hotelbeschreibung. Die Tabellen Users und Securitytokens haben eine 1:1-Beziehung sowie Users und Tipp. Dies ist Wichtig, da die Securitytokens zum Verifizieren der Session benötigt werden. Die Tabelle Hotel und Hotelbeschreibung haben eine 1:1-Beziehung.

# Kapitel 3: Aufgetretene Probleme

## Festlegen des Software-Stacks bzw. Recherche

Von Anfang an hatten wir gemeinsam Probleme uns auf einen Software-Stack zu einigen. Da wir beide uns noch gar nicht mit dem Thema Webentwicklung befasst hatten, mussten wir erstmal Recherche betreiben welche Sprachen, Webframeworks, Datenbanken und Entwicklungsumgebungen wir benötigen. Schnell war uns klar, dass wir die Webanwendung trotz erhöhter Komplexität, im ggs. zu PHP oder Java, in Python entwickeln wollten. Eine Wahl eines geeigneten Webframeworks gestaltete sich schwierig, da mehrere zur Auswahlstanden. Folgende Frameworks wurden von uns hauptsächlich in Erwägung gezogen:

### Django

Django ist mit, dass beliebteste Webframework für Python. Anfangs waren wir sehr begeistert da sich damit in relativ kurzer Zeit eine sehr umfangreiche Webanwendung schreiben lässt. Wir haben uns gegen Django entschieden, da es einem Entwickler sehr viel abnimmt und man mit wenigen Zeilen Code schon eine vollständige Webanwendung von Django zur Verfügung gestellt bekommt. Deshalb haben wir den Eindruck gewonnen das dieses Framework ungeeignet ist für das Fach W2: Serverseitige Anwendungen, da

eine Eigenleistung nicht erforderlich war. Wir hatten aber bereits mehrere Wochen mit Django und entsprechenden Datenbanken, dass Projekt vorangetrieben. Entsprechend war der Zeitverlust groß.

### Flask

Flask ist ein weiteres Webframework für Python. Es besticht durch seine Erweiterbarkeit und ist wesentlich anspruchsvoller für Entwickler, da Flask einem nur die Grundlegenden Funktionen wie ein eingebauter Entwicklungswebserver bietet und sich zurecht als Micro-Framework bezeichnet. Wir haben uns für Flask entschieden, da es gerade für kleinere Webanwendungen geeignet ist und wir uns dadurch erhofften die Webanwendung von Grund auf selbst zu gestalten.

Nach der Auswahl des Frameworks hatten wir eine längere Phase der Entscheidungsfindung gerade was die Datenbank betrifft. Es standen verschiedene Datenbanken zur Diskussion. Gerade im Python Umfeld ist PostgresSQL als Datenbank beliebt. Diese hatten wir auch mit Django benutzt und mussten uns entsprechend einarbeiten. PostgreSQL wurde nach dem Wechsel von Django auf Flask verworfen, weil wir mit Flask keine Server-Client-Datenbank benutzen wollten. Unsere Wahl fiel auf die In-Memory Datenbank SQLITE. Dies hatte zum Vorteil das wir nur eine Datenbankdatei im Projekt hatten, die relativ, zu anderen Datenbanken, klein war.

Durch die Recherchen hatten wir viel Zeit verloren. Auch dadurch das wir zuerst auf PostgresSQL mit Django gesetzt und dort auch schon einiges an Dokumentation gelesen hatten und schon mit dem Projekt relativ weit fortgeschritten waren.
Nach dem Wechsel auf Flask konnten wir nur unsere bereits mit Bootstrapp erstellten HTML5-Dateien migrieren, die wir auch noch an Jinja2 die Templating-Sprache von Flask anpassen mussten. Dadurch entstand hier auch noch eine Baustelle für uns.

Bis Mitte Januar sind wir gut vorangekommen in Flask und hatten schon unsere HTML-Templates erstellt sowie Routing-und GET- und POST-Funktionen geschrieben. Zusätzlich hatten wir mit Flask-SQLAlchemy eine Datenbank Anbindung relativ schnell realisiert. Die Probleme kamen erst als wir versuchten eine Registrierung der Nutzer in Flask zu realisieren. Normalerweise kann mit der Flask Erweiterung Flask-WTForms einfach Formulare erstellen die die Daten der HTML Formulare mit Jinja2 an diese weitergeben. Durch den mittlerweile größeren Zeitdruck haben wir gleichzeitig an zwei Lösungen gearbeitet. Zum einen versucht die Flask-WTForms weiterzuentwickeln und alternativ dazu mit JQuery eine Abfrage der eingegebenen Daten zu realisieren.

Anfang der KW4 2017 haben wir uns dann dazu entschieden das Projekt komplett nochmal in PHP zu entwickeln. Da uns von Kommilitonen geraten wurde, dass man mit PHP viele von unseren Problemen relativ einfach umsetzten kann.

## Probleme mit GIT

Während des Projektes haben wir zum aktualisieren des Python-Codes GitHub verwendet. Wir hatten während des Projektes mehrfach Probleme mit GitHub die ca. 4 Stunden des Projekts belegt haben. Es gab Probleme mit dem Remote-Repository und der Struktur des Projektes. Gerade beim Wechsel von Django auf Flask hatten wir Probleme die neue Projekt-Struktur im Repository zu ändern. Was uns gerade in dieser Kritischen Phase

aufgehalten hat. Beim zweiten Neustart des Projektes mit PHP haben wir uns auf die Hochschul Owncloud zum verteilen des Codes geeinigt.

## Probleme mit Speichern von Daten in der Datenbank

Wie schon beim Festlegen des Software-Stacks beschrieben, hatten wir erhebliche Probleme mit dem durchreichen der Daten vom HTML-Formular in die Datenbank, mit Flask und SQLITE. Ein Wechsel auf MariaDB hatte auch nicht funktioniert, weil wir dachten, dass es am Format von SQLITE liegt.

Wir hatten uns zwei mögliche Lösungen erarbeitet. Einmal mit der Flask-Erweiterung Flask-WTForms und mit JQUERY. Beides funktionierte nicht.

```python
from flask import Flask, render_template, json, request
from flask.ext.mysql import MySQL
from werkzeug import generate_password_hash, check_password_hash

app = Flask(__name__)
#Datenbank verbindung herstellen
mysql = MySQL()
app.config ['MYSQL_DATABASE_USER'] = 'Tim'
app.config ['MYSQL_DATABASE_PASSWORD'] = '123'
app.config ['MYSQL_DATABSE_DB'] = 'w2'
mysql.init_app(app)

#Connection zur Datenbank herstellen
conn = mysql.connect()
cursor = conn.cursor()

#Die Funktion sammelt alle Reihen einer Query und gibt eine Liste von tupeln zurück. Mysql-Connector.
data = cursor.fetchall()

 #Abfrage ob die Daten überhaupt gesammelt wurden.
if len(data) is 0:
    return json.dumps({'error':str(data[0])})
else:
    #Datenbank Connector der die Daten speichern soll
    conn.commit()
    return json.dumps({'message':'User created successfully !'})
#Einzelne Routen zu den Seiten, index.html, signup.html.
@app.route("/")
def main():
    return render_template('index.html')

@app.route('/showSignUp')
def showSignUp():
    return render_template('signup.html')

@app.route('/signup', methods=['POST'])
def signUp():
    _name = request.form['inputName']
    _email = request.form['inputEmail']
    _passwort = request.form['inputPassword']

#validierung der Werte

    if _name and _email and _passwort:
        return json.dumps({'html':'<span>Alle Felder korrekt !!</span>'})
    else:
        return json.dumps({'html':'<span>Füllen Sie die alle Felder aus</span>'})

if __name__ == "__main__":
    app.run()
```

*Abbildung 2: Code Beispiel mit JQuery und Json*

Im oberen Teil des Codes ist die Verbindung mit der Datenbank zu sehen. @app.route('/') zum Beispiel ist eine der Routen die, wenn man sie aufruft die Templates Rendern und so eine Seite aufbauen. Es wurde von uns versucht mit JSON und JQuery eine entsprechendes Speichern der Daten in der Datenbank zu realisieren.

```
8
9   application=Flask(__name__)
10  application.config.from_pyfile('config.py')
11
12  #Routen zur Base.html mit Funktion zum Rendern des Templates
13  @application.route('/')
14  @application.route('/index')
15  def test():
16      return render_template('base.html')
17
18  #Route zur register.html
19  @application.route('/register', methods=('POST', 'GET'))
20  def register():
21      return render_template('register.html', RegisterForm=RegistrationForm())
22  #Formvalidierung mit WTForms
23  def register_validate():
24      register_form = RegistrationForm()
25      if register_form.validate_on_submit():
26          nutzer = Kunde(
27              kundenname=register_form.kundenname.data,
28              passwort=register_form.passwort.data,
29              email=register_form.email.data,
30          )
31          #Einfügen der Daten in die Datenbank SQLAlchemy-Funktionen
32          db.session.add(nutzer)
33          db.session.commit()
34          flash('Registrierung erfolgreich!')
35          return redirect(url_for('/index'))
36      else:
37          #Wenn registrierung nicht erfolgreich bleibt man auf der register.html
38          return redirect(url_for('/register'))
39
40  #Formular mit WTForms für die register.html
41  class RegistrationForm(DefaultForm):
42      #Einzelene Registrierungsfelder
43      kundenname = StringField('Benutzername', [validators.Length(min=4, max=50)])
44      email = StringField('Email-Addresse', [validators.Length(min=6, max=50)])
45      passwort = PasswordField('Passwort', [validators.DataRequired(),
46          validators.EqualTo('bestaetigen', message='Passwoerter müssen uebereinstimmen')])
47      password_confirm = PasswordField('Passwort wiederholen', validators=[InputRequired()])
48      #Validierung ob
49      def validate(self):
50          if not FlaskForm.validate(self):
51              return False
52
53          if Kunde.query.filter_by(username=self.kundenname.data).first() is not None:
54              self.username.errors.append("Benutzername gibts es bereits.")
55              return False
56
57          if Kunde.query.filter_by(email=self.email.data).first() is not None:
58              self.email.errors.append("E-Mail-Adresse wird bereits verwendet")
59              return False
60          else:
61              return True
```

*Abbildung 3: Codebeispiel mit WTForms*

Hier wird eine mögliche Variante des Abspeicherns der Daten mit Hilfe des Flask-Moduls Flask-WTForms gezeigt. Die einzelnen Forms wurden mithilfe der Dokumentation von Flask-

WTForms erstellt und sind so auch als Codesnippets verfügbar (Quelle 5: Dokumentation Flask-WTForms).

Da wir mit beiden möglichen Lösungen nicht weiter vorangekommen sind, haben wir beschlossen die Webanwendung in PHP zu gestalten.

Unsere Lösung sieht man in Kapitel 4: Wichtige Codestellen und natürlich als die Abgabe, bzw. das fertige Projekt.

# Kapitel 4: Wichtige Codestellen

In diesem Kapitel werden die wichtigsten Codestellen dargestellt und beschrieben.

## Datenbankanbindung

Im Folgenden wird die Datenbankanbindung gezeigt:

```php
<?php

$db_host = 'localhost';
$db_name = 'W2';
$db_user = 'root';
$db_password = '';
$pdo = new PDO("mysql:host=$db_host;dbname=$db_name", $db_user, $db_password);

```

*Abbildung 4: Datenbankanbindung*

In der config.inc.php werden die Einstellungen für die MySQL-Datenbank festgelegt. In der Variable $pdo wird die PHP Data Object-Datenbankschnittstelle initialisiert und die Treiber für MySQL hinterlegt. Da PDO Prepared-Statements verwendet, schützt dies gegen SQL-Injections.

## Registrieren der Benutzer und Hashen des Passworts

Im Folgenden wird die register.php gezeigt:

```php
<?php
session_start();
require_once("inc/config.inc.php");
require_once("inc/functions.inc.php");

include("templates/header.inc.php")
?>
<div class="container main-container registration-form">
<h1>Registrierung</h1>
<?php
$showFormular = true; //Variable ob das Registrierungsformular anezeigt werden soll

if(isset($_GET['register'])) {
    $error = false;
    $username = trim($_POST['Username']);
    $email = trim($_POST['email']);
    $passwort = $_POST['passwort'];
    $passwort2 = $_POST['passwort2'];

    if(empty($username) || empty($email)) {
        echo 'Bitte alle Felder ausfüllen<br>';
        $error = true;
    }

    if(!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        echo 'Bitte eine gültige E-Mail-Adresse eingeben<br>';
        $error = true;
    }
    if(strlen($passwort) == 0) {
        echo 'Bitte ein Passwort angeben<br>';
        $error = true;
    }
    if($passwort != $passwort2) {
        echo 'Die Passwörter müssen übereinstimmen<br>';
        $error = true;
    }

    //Überprüfung ob die Email-Adresse registriert wurde.
    if(!$error) {
        $statement = $pdo->prepare("SELECT * FROM users WHERE email = :email");
        $result = $statement->execute(array('email' => $email));
        $user = $statement->fetch();

        if($user !== false) {
            echo 'Diese E-Mail-Adresse ist bereits vergeben<br>';
            $error = true;
        }
    }

    //Wenn die EmaiAdresse Korrekt ist wird der User registiert.
    if(!$error) {
        $passwort_hash = password_hash($passwort, PASSWORD_DEFAULT);

        $statement = $pdo->prepare("INSERT INTO users (email, passwort, username) VALUES (:email, :passwort, :username)");
        $result = $statement->execute(array('email' => $email, 'passwort' => $passwort_hash, 'username' => $username));

        if($result) {
            echo 'Du wurdest erfolgreich registriert. <a href="login.php">Zum Login</a>';
            $showFormular = false;
        } else {
            echo 'Beim Abspeichern ist leider ein Fehler aufgetreten<br>';
        }
    }
}
```

*Abbildung 5: Registrierungsfunktion*

In der register.php werden die config.inc.php und die functions.inc.php mit require_once eingebunden. Diese Dateien beinhalten die Datenbank Konfiguration mit der PDO-Datenbankschnittstelle und die functions.inc.php beinhalted die Funktionen zum Hashen der Passwörter und die check_user-Funktion die für das Sessionmanagement benötigt werden.

Die header.inc.php ist ein Template und wird eingebunden, damit die Navigationsbar verfügbar ist.

Im oberen Teil der php-Datei werden die Variablen für Email, Passwort und Benutzer festgelegt. Mit der trim-Funktion werden bei den Strings mögliche Leerzeichen abgeschnitten. In der folgenden IF-Abfragen wird überprüft ob der Username oder die E-Mail leergelassen wurden und ob das Passwort eingeben wurde, bzw. ob die Passwörter übereinstimmen. Entsprechend wird eine Fehlermeldung zurückgegeben.
Danach wird die E-Mail-Adresse geprüft ob diese bereits registriert wurde. Hier wird über die PDO-Datenbankschnittstelle abgefragt ob bereits eine gleiche Email existiert.

Der wichtigste Teil ist im unteren Teil des Bildes zusehen hier wird, falls alle Abfragen korrekt sind, das Passwort mit der Funktion passowrd_hash aus der functions.inc.php gehashed und der Hashwert sowie Benutzername und Email-Adresse in die Datenbank geschrieben. Es wird entsprechend eine Rückmeldung zur erfolgreichen Registrierung zurückgegeben.

## Einloggen der Benutzer

Im Folgenden wird der Login gezeigt, in der login.php:

```php
<?php
session_start();
require_once("inc/config.inc.php");
require_once("inc/functions.inc.php");

$error_msg = "";
if(isset($_POST['email']) && isset($_POST['passwort'])) {
    $email = $_POST['email'];
    $passwort = $_POST['passwort'];

    $statement = $pdo->prepare("SELECT * FROM users WHERE email = :email");
    $result = $statement->execute(array('email' => $email));
    $user = $statement->fetch();

    //Überprüfung des Passworts
    if ($user !== false && password_verify($passwort, $user['passwort'])) {
        $_SESSION['userid'] = $user['id'];

        //Möchte der Nutzer angemeldet beleiben?
        if(isset($_POST['angemeldet_bleiben'])) {
            $identifier = random_string();
            $securitytoken = random_string();

            $insert = $pdo->prepare("INSERT INTO securitytokens (user_id, identifier, securitytoken) VALUES (:user_id, :identifier, :securitytoken)");
            $insert->execute(array('user_id' => $user['id'], 'identifier' => $identifier, 'securitytoken' => sha1($securitytoken)));
            setcookie("identifier",$identifier,time()+(3600*24*365)); //Valid for 1 year
            setcookie("securitytoken",$securitytoken,time()+(3600*24*365)); //Valid for 1 year
        }

        header("location: internal.php");
        exit;
    } else {
        $error_msg =  "E-Mail oder Passwort war ungültig<br><br>";
    }

}

$email_value = "";
if(isset($_POST['email']))
    $email_value = htmlentities($_POST['email']);

include("templates/header.inc.php");
?>

<?php
if(isset($error_msg) && !empty($error_msg)) {
    echo $error_msg;
}
?>
    <label for="inputEmail" class="sr-only">E-Mail</label>
    <input type="email" name="email" id="inputEmail" class="form-control" placeholder="E-Mail" value="<?php echo $email_value; ?>" required autofocus>
    <label for="inputPassword" class="sr-only">Passwort</label>
    <input type="password" name="passwort" id="inputPassword" class="form-control" placeholder="Passwort" required>
    <div class="checkbox">
      <label>
        <input type="checkbox" value="remember-me" name="angemeldet_bleiben" value="1" checked> Angemeldet bleiben
      </label>
    </div>
    <button class="btn btn-lg btn-primary btn-block" type="submit">Login</button>
    <br>
    <a href="passwortvergessen.php">Passwort vergessen</a>
  </form>

</div> <!-- /container -->
```

*Abbildung 6: Userlogin*

In der login.php-Datei wird zuerst geprüft ob eine E-Mail oder Passwort eingegeben wurde. Falls etwas eingegeben wurde wird mit $_POST die Daten dem Server mitgeteilt, indem sie an das Formular übergeben werden. Die E-Mail-Adresse wird entsprechend mit der Datenbank abgeglichen und das Passwort nochmals überprüft. Es gibt noch die Option ob der User angemeldet bleiben möchte. Hier werden Random Strings, mittels der Funktion random_sting in der functions.inc.php, in die Variablen identifier und securitytoken gespeichert.

Für die Cookies wird eine Gültigkeit von einem Jahr festgelegt. Im unteren Teil des Codes ist das Formular dargestellt. Hier wird die Checkbox **Angemeldet bleiben** und die Label zum eingeben des Passwortes und der Email-Adresse festgelegt.

## Usertipp-Kommentare

Um die Kommentar zu realisieren wurde eine tipp.php erstellt. Diese ist im Folgenden dargestellt.

```php
<?php
session_start();
require_once("inc/config.inc.php");
require_once("inc/functions.inc.php");

$user = check_user();

include("templates/header.inc.php");
?>
<img id="bg" src="./img/keyboard-1621105_960_720.jpg" hight="600" width="500";>
<div allign="center">
<ul class="blocktext3">
<?php
        if (isset($_POST['Tipp'])) {
                $statement = $pdo->prepare("insert into Tipp(user_id, Tipp) values (?, ?)");
                $statement->execute(array($_SESSION['userid'], $_POST['Tipp']));
        }
    ?>
<div class="container">
        <h1>USER-TIPP</h1>
<?php
      if (isset($_POST['Tipp'])) {
        $statement = $pdo->prepare('SELECT u.username, t.tipp from Tipp as t join users as u on u.id = t.user_id');
        $statement->execute();
        if ($statement->rowCount() == 0) {
          echo "<p>Keine Ergebnisse gefunden</p>";
        } else {
          while ($row = $statement->fetch()) {
            echo "<b><u>".$row['username']."</b></u>";
            echo "<br>";
            echo "<i>".$row['tipp']."</i>";
      echo "<br><br>";
          }
        }
        }
        ?>
    </div>

<form class="form-register" action=""  method="post">
<label for="Tipp" class="sr-only">Tipp</label>
            <textarea class="span6" rows="3" placeholder="Tipp" name="Tipp" required autofocus></textarea>
<br><br>
<button class="btn btn-lg btn-primary" type="submit">Posten</button>
</form>
</ul>
</div>
```

*Abbildung 7: tipp.php*

Zuerst wird die aktuelle Usersession aufgerufen und es wird geprüft ob die entsprechenden *.php-Dateien schon eingebunden sind.

Im ersten PHP-Script wird mit dem INSERT-SQL-Befehl Daten in der Datenbank abgelegt. Dazu wird ein Prepared-Statement verwendet. Mit der execute()-Funktion werden die Felder userid und tipp in der Datenbank mit den Daten des Benutzers beschrieben.

Im zweiten PHP-Script wird in der Datenbank mit einem JOIN-SQL-Befehl der Username und der Tipp aus der Datenbank gelesen. Das Prepared-Statement wird mit der execute()-Funktion ausgeführt. Danach wird abgefragt ob die Anzahl der Zeilen des Prepared-Statements Null ist. Das bedeutet, wenn in der Datenbank kein Benutzername und Tipp, die übereinstimmen, vorhanden sind, wird die Fehlermeldung „ Keine ergebisse gefunden“ ausgegeben. Sind Kommentare vorhanden werden in der while-Schleife werden alle verfügbaren Kommentare mit der fetch()-Funktion ausgelesen und angezeigt.

## Anzeigen der Beschreibungen in der internal.php

```
<h4> Folgende Hotels werden für Sie empfohlen: </h4>
</div>
   <div class="row">
        <div class="col-md-4">
          <ul class="blocktext"><b>
                <h2>Dom Hotel Worms: </h2>
        <td><img src="img/Dom_Hotel.gif" alt="Dom Hotel" width="200" height="109"/><br /></td>
                  <?php
            $statement=$pdo->prepare('SELECT Beschreibung FROM Hotelbeschreibung WHERE Hotelbeschreibung_ID = ?');
            $statement->execute(array(2));
            $bes=$statement->fetch();
            echo $bes['Beschreibung'];?>
                </div>
        </ul></b>
    <div class="col-md-4">
     <ul class="blocktext"><b>
        <h2>Parkhotel Prinz Carl: </h2>
        <td><img src="img/Park_Hotel.gif" alt="Parkhotel Prinz Carl" width"200" height="109"/><br /></td>
                <?php
            $statement=$pdo->prepare('SELECT Beschreibung FROM Hotelbeschreibung WHERE Hotelbeschreibung_ID = ?');
            $statement->execute(array(1));
            $bes=$statement->fetch();
            echo $bes['Beschreibung'];?>
        </div>
        </ul></b>
    <div class="col-md-4">
     <ul class="blocktext"><b>
        <h2> Asgard Hotel: </h2>
        <td><img src="img/Asgard_Hotel.gif" alt="Asgrad Hotel" width"200" height="109"/><br /></td>
                <?php
            $statement=$pdo->prepare('SELECT Beschreibung FROM Hotelbeschreibung WHERE Hotelbeschreibung_ID = ?');
            $statement->execute(array(3));
            $bes=$statement->fetch();
            echo $bes['Beschreibung'];?>
        </div>
        </ul></b>
      </div>
        </div>

```

*Abbildung 8: Ausschnitt internal.php - Prepared-Statements*

Hier ist ein Ausschnitt aus der internal.php-Datei zu sehen. Die einzelnen GIF-Dateien der Hotels sind im HTML-Code hinterlegt und die Beschreibungen werde mit Prepared-Statements aus der Datenbank ausgelesen.

# Kapitel 5: Benutzerhandbuch

## Hauptseite

Wenn man die Webanwendung aufruft wird man auf die Hauptseite geleitet. Die Hauptseite bietet eine Möglichkeit dem BMS Hotel-Club beizutreten. Die Hauptseite bietet zusätzlich eine Möglichkeit sich ein und aus zu loggen sowie Informationen über den Hotel-Club. Mit einem Klick auf **Jetzt Mitglied werden** kann man sich registrieren


*Abbildung 9: Hauptseite der Anwendung*

## Registrieren

Wenn man dem Hotel-Club beitreten möchte, kann man sich einen Benutzer erstellen und ein Passwort vergeben. Man wird direkt eingeloggt und auf den internen Bereich weitergeleitet.

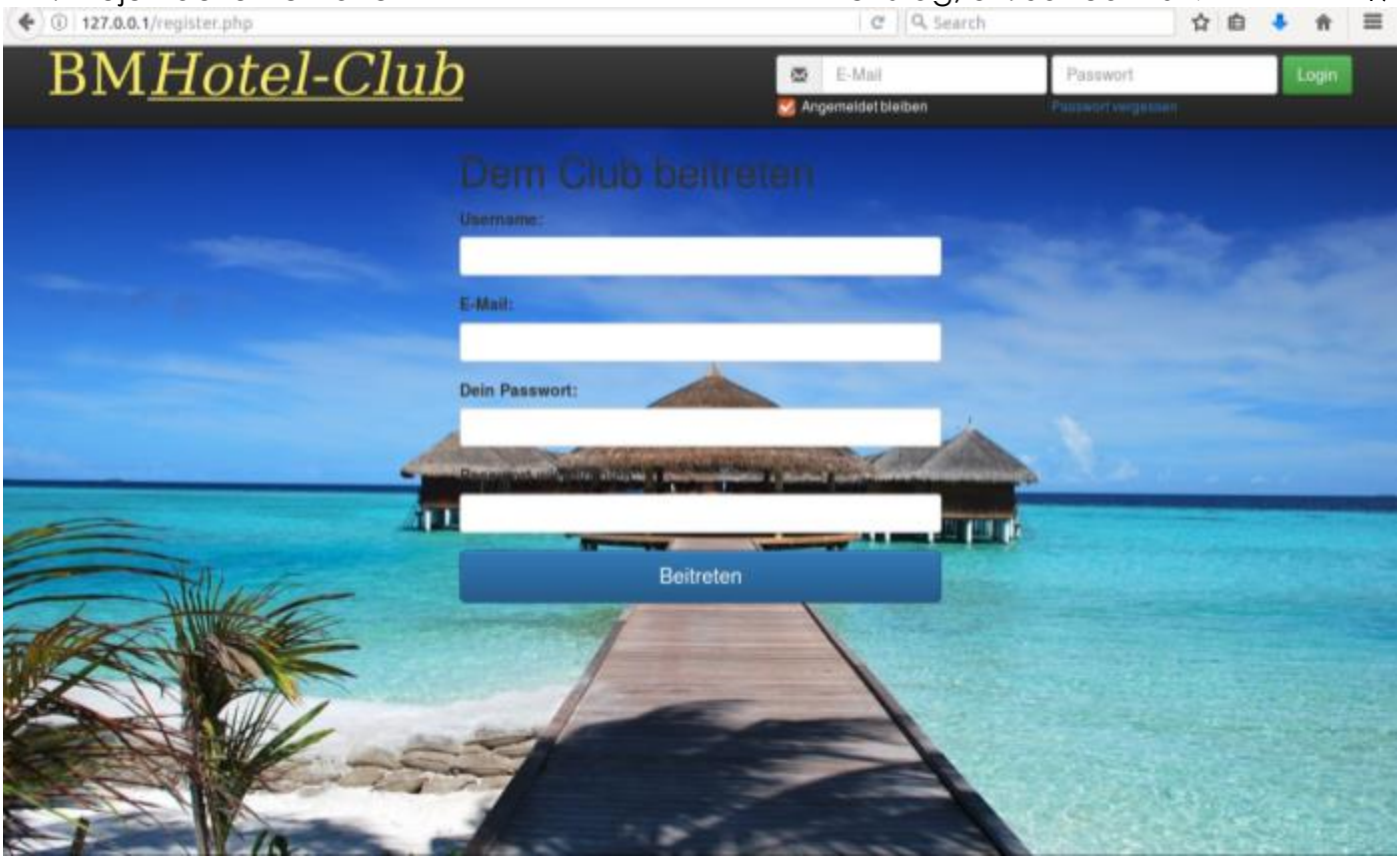


*Abbildung 10: Dem Klub beitreten*

## Login und Logout

Es gibt mehrere Möglichkeiten sich ein und aus zu loggen. Zum einen auf der Hauptseite in der Navigationsbar und auf der Login-Seite. Zum anderen kann man sich auch über die Navigationsbar ausloggen.

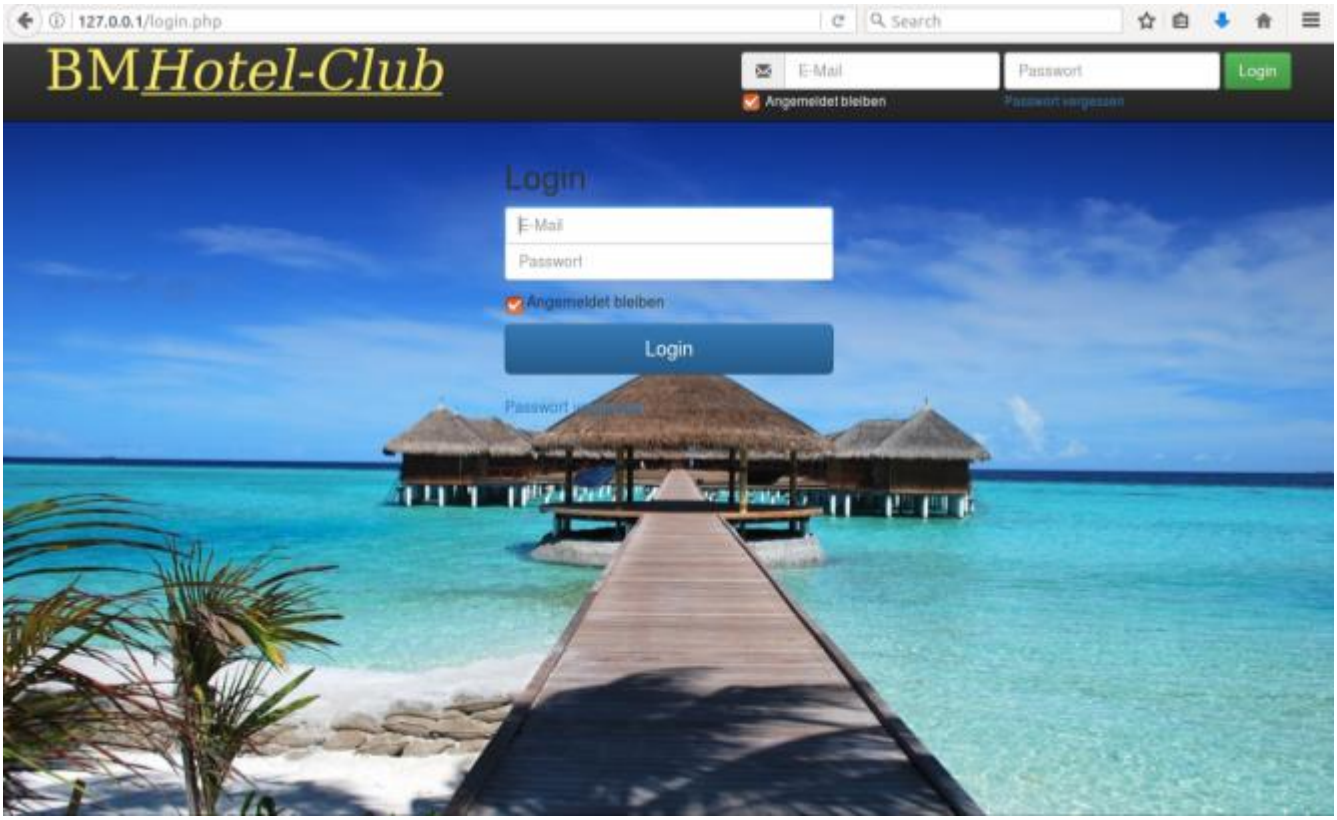


*Abbildung 11: Member Login*

## Members Area

Im internen Bereich gibt es die Hotelvergleiche. Dort kann man die exklusiven Hotels sich anschauen und Bewerten. Zusätzlich gibt es eine Beschreibung der einzelnen Wormser Hotels.

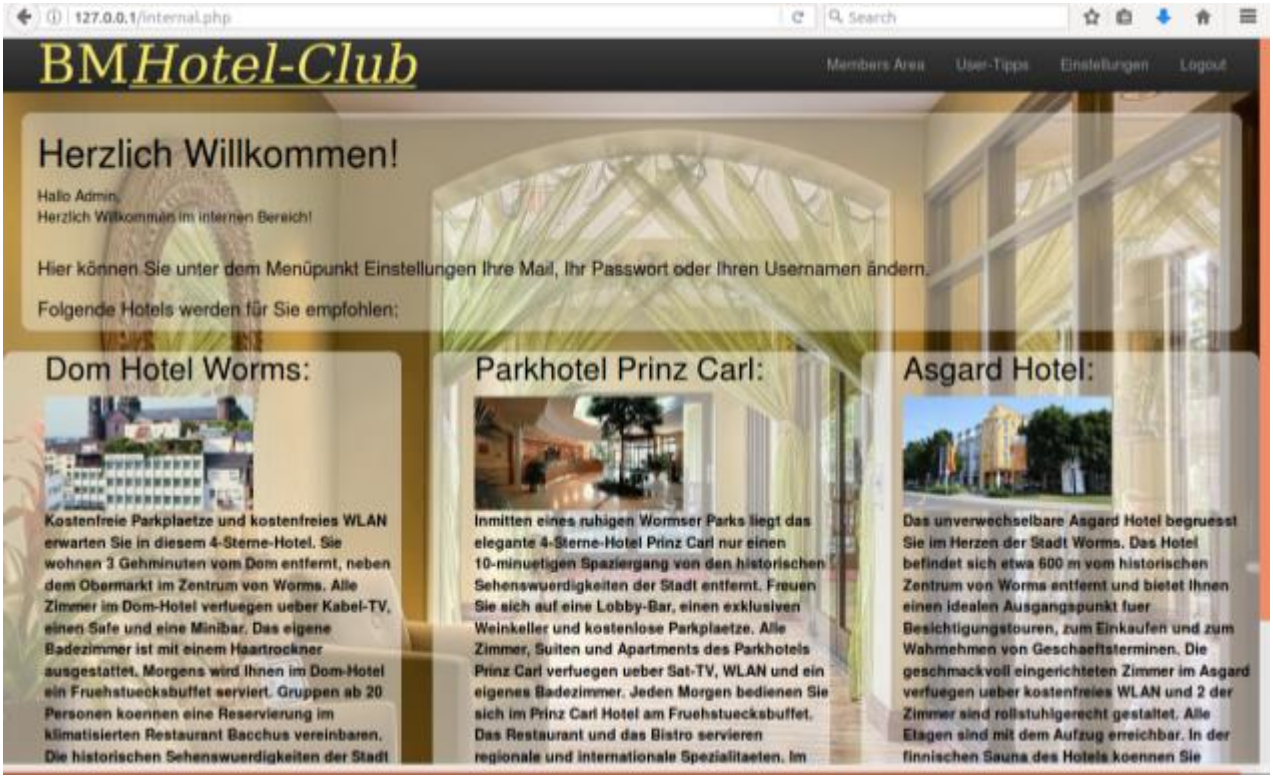


*Abbildung 12: Member Area*

## User-Tipps

Damit die User im Hotel-Club auch objektiv die einzelnen Hotels bewerten können und auch Erfahrungen austauschen können gibt es die Möglichkeit einer Usertippfunktion. Dies bedeutet das Nachrichten in einem Chat gepostet werden können und für alle Member des Hotel-Clubs sichtbar sind.

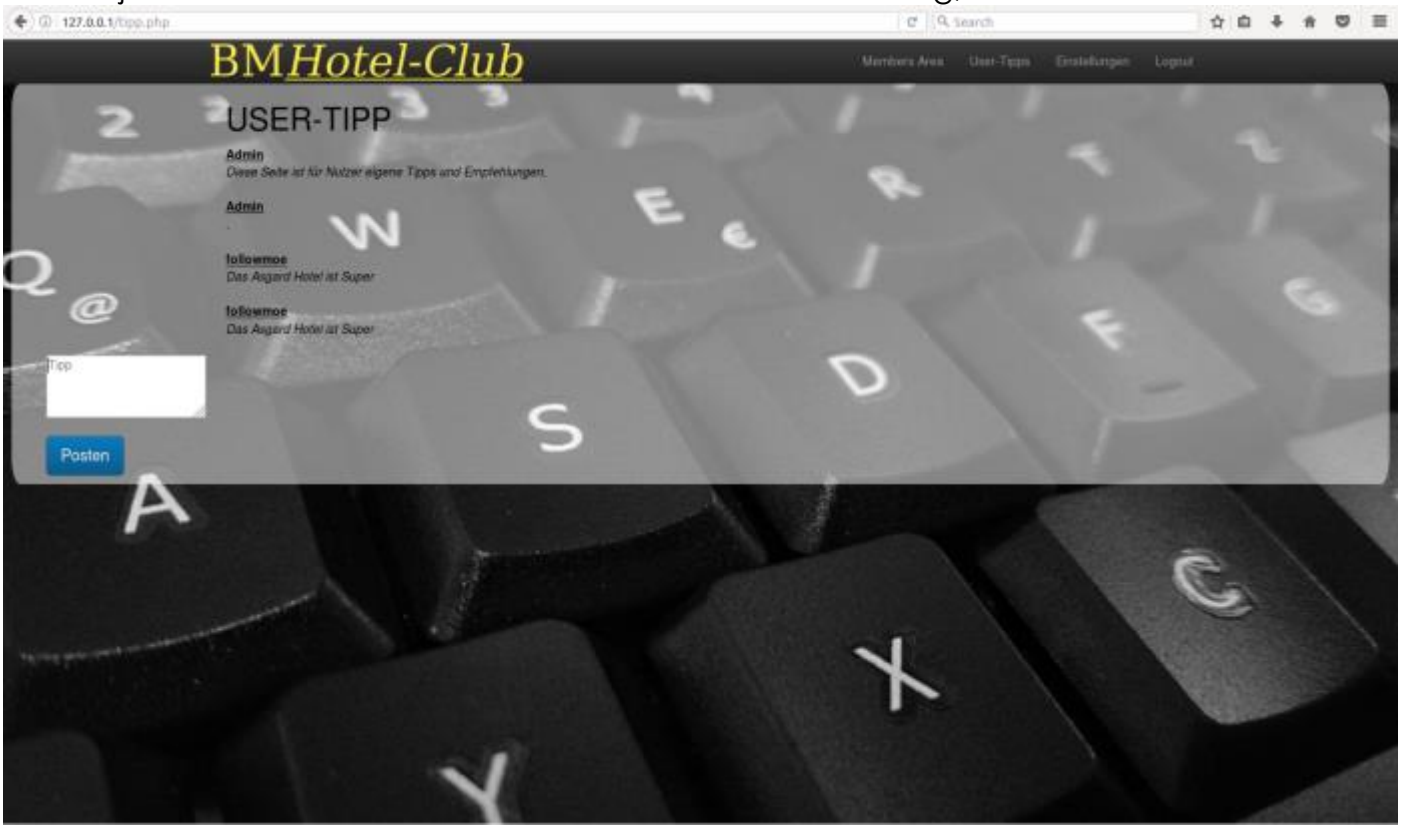


*Abbildung 13: Usertipp-Bereich*

## Einstellungen

In den Einstellungen können die Benutzer ihren Benutzernamen ändern sowie ihre Emailadresse zurücksetzten und das Passwort ändern.

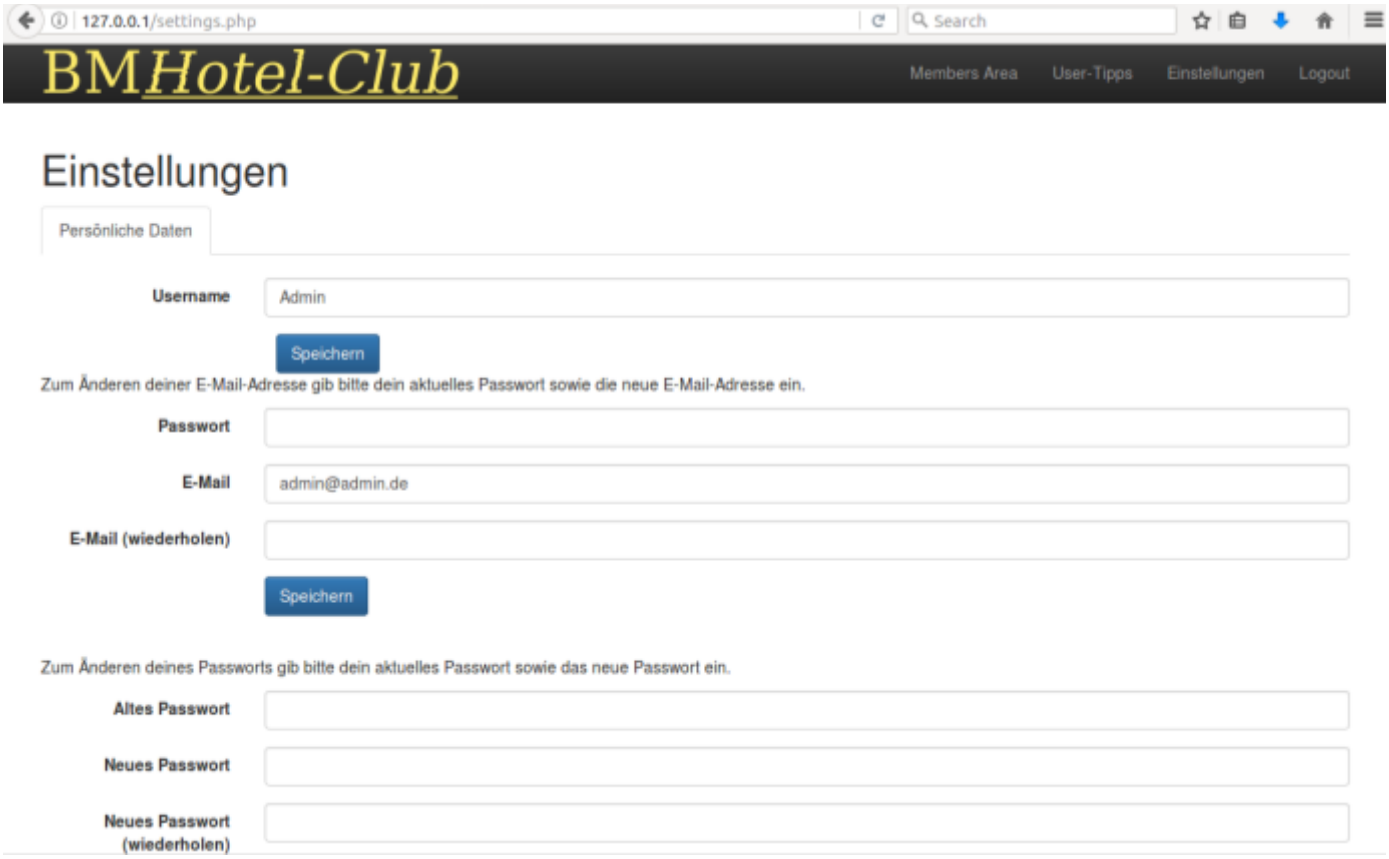


*Abbildung 14: User Einstellungen*

# Kapitel 6: Installationsanleitung

Zur Installation der Webanwendung müssen folgende Vorarbeiten geleistet werden. Diese Anleitung ist für Linux Ubuntu > Version 16.04

## Installieren von XAMPP als Entwicklungsumgebung

a. Download von XAMPP auf dieser Seite. Es wurde PHP 7.0.13 verwendet.
b. Ändern der Berechtigungen des Installers mit **chmod 755 xampp-linux-*-installer.run**
c. Installieren von XAMPP mit **sudo ./xampp-linux-*-installer.run**

## Starten von XAMPP

Zum Starten von XAMPP gibt man im Terminal folgenden Befehl ein: **sudo /opt/lampp/lampp start** ein. Ein Stopp oder Neustart ist mit **stopp/restart** möglich.

Alternative kann man dies auch über das XAMPP Dashboard machen.

## Importieren der Datenbank

Öffnen Sie einen beliebigen Browser und kopieren sie folgendes in die Adresszeile: **http://localhost/phpmyadmin**

Zuerst muss eine neue Datenbank Namens **W2** erstellt werden. Dazu klick man im Datenbankbaum auf **New**.

Zum Importieren der Datenbank wählt man in der PHPMyAdmin-Oberfläche den Reiter Importieren aus.

1. Nun wählt man die beigefügte **W2.sql** aus und beginnt den Import der Database.
2. Also Login-Informationen wurde als Username **root** vergeben und es wurde kein Passwort hinterlegt. Da es sich um einen Prototypen der Webanwendung handelt. In einer Produktionsumgebung würde das Passwort anders ausfallen.

Nach importieren der Datenbank muss man XAMPP stoppen, damit der htdocs Ordner kopiert werden kann.

## Kopieren des htdocs-Ordners

Nun muss der htdocs-Ordner aus dem Portfolio in den htdocs-Ordner von der XAMPP Installation kopiert werden. Diese liegt unter **/opt/lamp/htdoc**. Es wird davon ausgegangen, dass der Ordner W2Projekt_Final im Ordner Downloads liegt. Es kann natürlich auch ein anderer absoluter Pfad als Quelle im cp-Befehl angegeben werden.

Am besten kopiert man dies mit dem Befehl:

**Sudo cd /opt/lampp/htdocs && cp -r ~/Downloads/W2Projekt_FINAL/htdocs .**

Die Webanwendung sollte nun verwendet werden können. Dazu muss man in einem beliebigen Browser folgendes in die Adresszeile eingeben:

**127.0.0.1/**

# Anhang

## Quellenverzeichnis

Im Folgenden werden die Quellen die für das Projekt verwendet werden aufgelistet.

### Python-Quellen

Quelle 01: The Flask Mega Tutorial
Quelle 02: Dokumentation Django
Quelle 03: Dokumentation Flask
Quelle 04: Dokumentation Flask-SqlAlchemy
Quelle 05: Dokumentation Flask-WTForms
Quelle 06: Tutorial Pythonprogramming.net
Quelle 07: Tutorial tutroialspoint.com
Quelle 08: Fragen rund um Code – stackoverflow.com
Quelle 09: Dokumentation SQLITE
Quelle 10: Dokumentation PostgresSQL
Quelle 11: YouTube Django for Beginners
Quelle 12: YouTube Webdevelopment with Django and Python
Quelle 13: YouTube Practical Flask Web Development Tutorials
Quelle 14: YouTube Flask Tutorial

### PHP-Quellen

Quelle 01: Dokumentation PHP
Quelle 02: Tutorial php-einfach.de
Quelle 03: Tutorial selfhtml.org
Quelle 04: FAQ - XAMPP
Quelle 05: YouTube PHP-Tutorial The New Boston
Quelle 06: Bilder der Website – pixarbay.com
Quelle 07: Hoteldaten – Booking.com

## Abbildungsverzeichnis