NEINGAG

Projekt für serverseitige Anwendungen

Exposee

Neingag ist die Inverse von 9gag.
Während sie ständig von witzigen Medien bombardiert werden, sorgt Neingag dafür,
dass sie sich nicht „totlachen“ können.

Moritz Vetter
Inf2557@hs-worms.de
Matrikelnr. 671303

Björn Ammon
inf2517@hs-worms.de
Matrikelnr. 671303

Inhaltsverzeichnis

1. Einleitung

Für das Modul „Serverseitige Anwendungen“ soll eine Anwendung erstellt werden, die mit einer Datenbank kommuniziert. Hierbei steht die Funktionalität im Vordergrund, Design ist nicht ausschlaggebend.

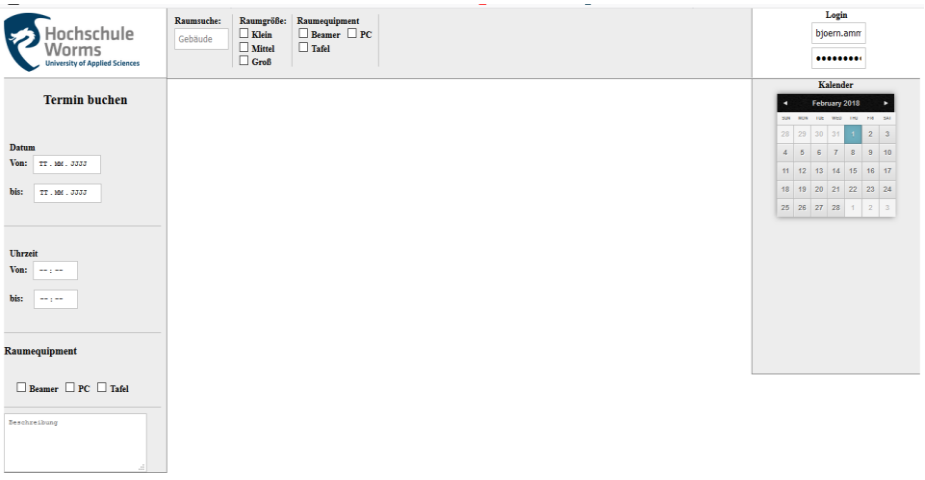
Diverse Funktionalitäten wie Login und verschlüsselte Passworteinträge in der Datenbank sind Voraussetzungen für ein anständiges Projekt.

2. Ideen

2.1 Raumbuchung

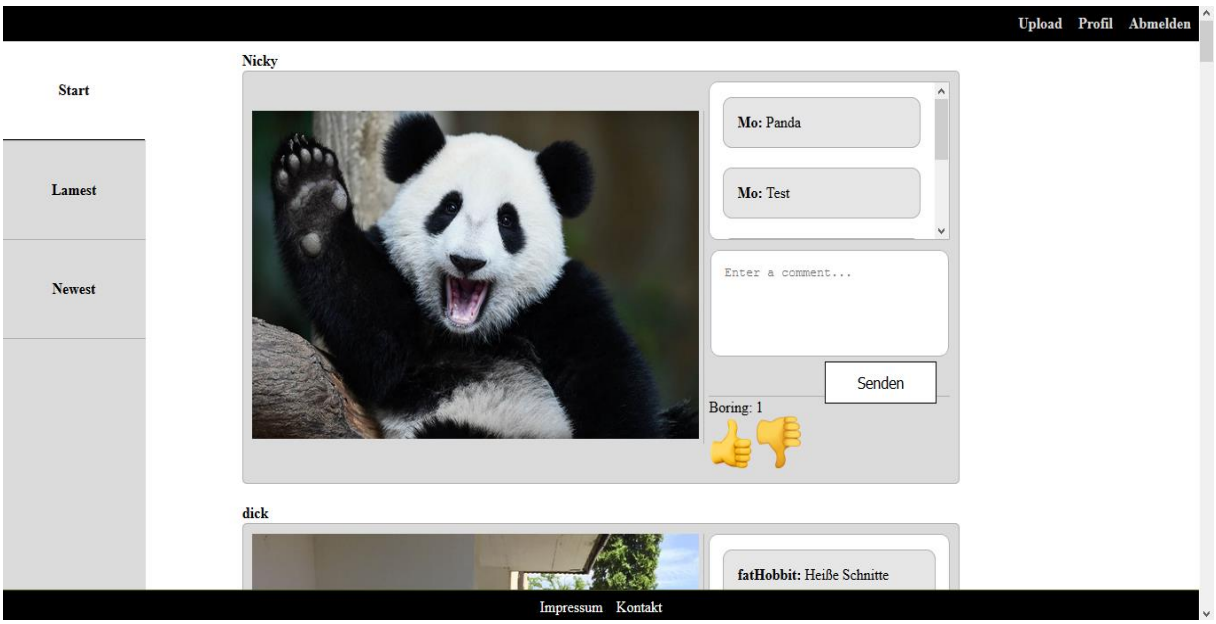
Die erste Idee war eine Seite zur Raumbuchung an der Hochschule Worms.
Hier hätte man die Möglichkeit von einem Account aus, einen Raum zu buchen. Studenten der Hochschule Worms hätten hier ihren Login des Rechenzentrums verwendet, also z. B. inf2517@hs-worms.de. Organisationen sollten einen extra Account erstellen, z. B. veranstaltunge-eichbaum@gmx.de als E-Mail-Adresse. Informationen wären dann an die angegebene Adresse gesendet worden.
Nun hätte der Nutzer einen Raum ausgesucht. Mit der Eingabe der Gebäudenummer und der Raumnummer wäre er auch visuell, mit einer Blaupause im Zentrum der Seite, zum Raum gelangt. Hier hätte er auch Informationen über den Raum ablesen können, z. B. die Raumgröße, mit welchen Gadgets er ausgestattet ist (Beamer), oder ob und in welchem Zeitraum der Raum bereits reserviert wurde (angaben können privatisiert werden).
Die Suche nach einem treffenden Raum hätte auch nur über die angegebenen Suchkriterien erfolgen können.
Hat der Nutzer einen passenden Raum für seine Veranstaltung gefunden, konnte er eine Buchung beantragen. Zeitraum wäre eine benötigte Information, während „Raumequipment“, oder „Beschreibung“ freiwillig ist.
Der Admin hätte durch den Master-Login diese Information erhalten und die Reservierung bestätigen können.

Da diese Idee unserer Meinung nach zu simpel war, haben wir sie verworfen. Diverse Funktionalitäten wie Login, Raumsuche und Raumbuchung waren bereits gegeben.



2.2 Neingag

Die zweite und etwas herausfordernde Idee war, eine Seite zu erstellen, die eine Upload-, Bewertungs- und Kommentarfunktion beherrscht. Nach langem grübeln kam dann einfach die Idee „Neingag“ zu erstellen, eine Seite die nur für langweilige Bilder gedacht ist und den Nutzer davor schützen sollte, sich nicht „tot zu lachen“.
Auch hier gibt es eine Registration und das dazugehörige Login. Sofern ein Nutzer eingeloggt ist, kann er sämtliche Funktionen nutzen. Als uneingeloggter Nutzer steht hier lediglich die Betrachtung der Bilder zur Auswahl.
War der Login erfolgreich, kann der Nutzer Kommentare hinterlassen, Bilder bewerten und sogar eigene Bilder hinzufügen. Auch eine eigene Profilseite steht dem Nutzer zur Verfügung. Hier kann der Nutzer auch sein Profil bearbeiten und seine eigenen Bilder sehen.
Die Anzeige der Bilder kann unterschiedlich sortiert werden. Zur Auswahl stehen „Newest“, „Latest“ und „Start“ zur Verfügung. Letzteres gibt eine zufällige Folge von Bildern zurück. Hier werden allerdings nicht die Bilder vom eingeloggten Nutzer angezeigt.
Bei Unzufriedenheit hat der Nutzer natürlich die Option, seinen Account und andere Informationen zu löschen.



3. Vorbereitung

Bevor wir uns an die Schreibarbeit gemacht haben, haben wir erstmal geklärt, welche Programmiersprache wir benutzen, wie unsere Seite aussehen soll, wie und welche Funktionalitäten wir implementieren, welches Datenbanksystem wir verwenden und wie wir unsere Datenbank aufbauen.
Dank der hilfreichen Seite „php-einfach.de“ und der Einführung von Herrn Maler, wollten wir uns damit durch die php-Sprache führen lassen.
Das erleichterte uns auch die Auswahl des Datenbanksystems, da MySQL des Öfteren empfohlen wurde und wir bereits Vorkenntnisse darin hatten.
Daraufhin erstellten wir ein Wireframe der Seite, bis unsere Zufriedenheit etwas gesättigt wurde.
Bevor wir unsere Datenbank aufgebaut haben, überlegten wir uns diverse Funktionalitäten und damit auch die Informationen, die die Datenbank speichern musste.
Durch diese Organisation gelang uns der Grundaufbau, durch das abwechselnde Programmieren haben wir uns jedoch des Öfteren gegenseitig Fehler eingebaut.
Wir lagerten und aktualisierten den Code zwar regelmäßig über GitHub, doch durch schlecht dokumentierten Änderungen gab es fast regelmäßig Verständnisprobleme, die das Lösen der Fehler erschwerten. Dieses Problem beseitigten wir durch Hinzufügen einer Text-Datei mit ausführlichen Update-Informationen, bis dahin ging jedoch viel Zeit durch Korrekturen verloren.

4. Realisierung

4.1 Datenbank

Users:

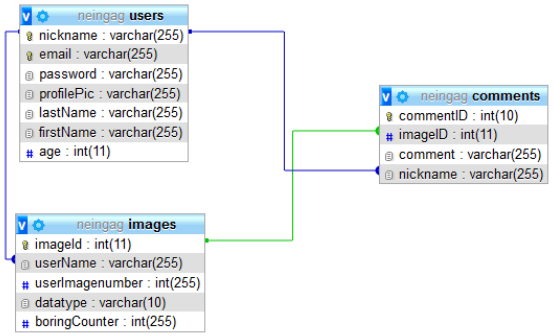
„Users“ speichert sämtliche Informationen, die den Nutzer betreffen. Die wichtigsten hier sind E-Mail, Nickname und Passwort. Ersteres sind „Unique Keys“, dürfen also nur einmal in der Datenbank vorhanden sein. Dadurch konnten wir den Nickname auch als „Primary Key“ verwenden, was uns die Informationsabfrage erleichtern wird, da wir das dann auch als „Foreign Key“ verwenden können. Restliche Informationen sind optional. Passwort wird verschlüsselt angezeigt.

Images:

„Images“ erstellt und inkrementiert automatisch die „ImageId“ („Primary Key“) bei einem Upload. „UserName“ gibt uns die Information, wer die Bilder hochgeladen hat und die „userImagenumber“ indiziert und inkrementiert (durch Code) die Nummer des Bildes vom jeweiligen User. Hat beispielsweise „MaxMuster“ drei Bilder hochgeladen, wird das nächste Bild die „userImagenumber == 4“ haben. Außerdem dienen die Infos zur Abspeicherung und Aufrufung der Bilder, genauso wie das Dateiformat. Die „Votes“ der Bilder werden auch hier abgespeichert. Dazu später mehr.

Comments:

„Comments“ speichert sämtliche Kommentare in der Datenbank. „CommentId“ wird automatisch mit jedem neuen Kommentar inkrementiert. Außerdem wird dazu auch der Nickname des Autors und die ID des betroffenen Bildes gespeichert.



4.2 Registration

Die Registration war unsere erste Programmierhürde. Nachdem wir ein Formular erstellten, welches sämtliche Eingaben per POST-Methode übergibt, speicherten wir alles in ein Array, und verglichen die gegebenen Informationen mit unseren Bedingungen. Entspricht das Passwort unseren Vorstellungen? Wurde der Benutzername bereits vergeben, oder ist die E-Mail-Adresse bereits vorhanden? Existenzfragen wurden mit folgendem Codeschnipsel gelöst:

```php
//Check, if email address/nickname is been taken
if(!$check) {
    $sql = "SELECT * FROM users WHERE email = '$newUser['email']' OR nickname = '$user['nickname']'";
    $user = $pdo->query($sql)->fetch();

    if($user['email'] == $newUser['email']) {
        echo 'Diese E-Mail-Adresse ist bereits vergeben<br>';
        $check = true;
    }
    if($user['nickname'] == $newUser['nickname']){
        echo 'Dieser Nickname ist bereits vergeben<br>';
        $check = true;
    }
}
```

Sobald alle Bedingungen erfüllt sind, wird das Passwort verschlüsselt und ein Datenbankeintrag hinterlegt.

```php
$newUser['password'] = password_hash($newUser['password'], PASSWORD_DEFAULT);

//write in database
$statement = $pdo->prepare("INSERT INTO users (nickname, email, password) VALUES (:nickname, :email, :password)");
$result = $statement->execute($newUser);
```

Zeitgleich wird ein Ordner für die Bilder des frisch registrierten Nutzers erstellt, worin sämtliche Uploads gespeichert werden.

```php
//create individual folder
mkdir('users/'.$newUser['nickname']);
```

4.3 Login

Nachdem die Registration erfolgreich abgeschlossen wurde, kann nun endlich der Login stattfinden.
Hier wird lediglich die E-Mail-Adresse, oder Nickname, so wie das Passwort per POST-Methode übergeben.
Daraufhin wird aus der Datenbank die jeweilige Spalte mit der angegebenen E-Mail-Adresse/Nickname ausgegeben und das jeweilige Passwort über eine gegebene Funktion überprüft. Diese Funktion vergleicht einen Hash-Wert mit dem Passwort und kann damit die Richtigkeit verifizieren.
Danach wird der Nickname für weitere Zwecke in der $_SESSION – Variable gespeichert.

```php
$sql = "SELECT * FROM users WHERE email = '$email' OR nickname = '$email'";
$user = $pdo->query($sql)->fetch();

//check password
if (isset($user) && password_verify($loginPassword, $user['password'])) {
    $_SESSION['userid'] = $user['nickname'];
```

4.4 Upload

Der Upload war eine weitere Hürde für uns. Hier mussten wir Fragen beantworten wie: „Wie begrenzen wir die Dateiformate?“, „Wie bzw. unter welchem Namen speichern wir die Bilder?“, „Wie rufen wir sie ab?“ und vor allem „Wie unterscheiden wir Profilbilder mit den nicht-Profilbildern?“.

Dank php-einfach.de ließen sich die Fragen schnell beantworten. Mit Hilfe der Bibliotheksfunktion „pathinfo()“ konnten wir den Dateityp herausfiltern und begrenzen.

```php
$extension = strtolower(pathinfo($_FILES['datei']['name'], PATHINFO_EXTENSION));

//Check IMG Type
$allowed_extensions = array('png', 'jpg', 'jpeg', 'gif');
if(!in_array($extension, $allowed_extensions)) {
    die("Ungültige Dateiendung. Nur png, jpg, jpeg und gif-Dateien sind erlaubt");
}
```

Auch weitere Restriktionen haben wir eingeführt, z. B. die Bildgröße auf 500kb.

Mit der Hilfe der Variable $_SESSION, nehmen wir zur Dateibenennung den Nickname, fügen am Ende ein „_“, die Bildnummer und den Dateityp hinzu. Die Bildnummer inkrementieren wir manuell ab dem Wert 1.

Den Wert 0 erhalten nur Profilbilder, was uns die Trennung erleichtert.

```php
if($isProfPic)
    $newImage['userImagenumber'] = 0;
else {
    //Individual IMG ID increment
    $sql = "SELECT * FROM images WHERE userName='$newImage[userName]' ORDER BY userImagenumber DESC";
    $lastImagenumber = $pdo->query($sql)->fetch();
    echo '<br />'.$lastImagenumber['userImagenumber'];
    if(empty($lastImagenumber['userImagenumber'])){
        $newImage['userImagenumber'] = 1;
    }else {
        $lastImagenumber['userImagenumber']++;
        $newImage['userImagenumber'] = $lastImagenumber['userImagenumber'];
    }
}
```

Nachdem das Hexenwerk vollbracht wurde, wird das Bild genommen und in den jeweiligen Nutzerordner auf dem Server gespeichert. So können wir es später für die Betrachtung nutzen.

Um jedoch noch während des Uploads von Profilbildern und normalen Bildern zu unterscheiden, mussten wir herausfinden, woher die Upload-Abfrage kommt. Also haben wir über die GET-Methode die Information von der Profil-Seite über die URL herausgefiltert und mit Hilfe einer Fallunterscheidung die Gründe des Uploads bestimmt.
Falls es dann ein Profilbild war, wurde in der Datenbanktabelle „users“ die jeweilige Spalte „profilePic“ verändert.

```php
if($isProfPic){
    $statement = $pdo->prepare("UPDATE users SET profilePic = ? WHERE nickname = ?");
    $statement->execute(array($newImage['userName'].'_0.'.$extension, $_SESSION['userid']));
```

4.5 Anzeige

Nun können wir endlich die Bilder anzeigen lassen. Dazu haben wir uns verschiedene Anzeigemöglichkeiten überlegt: „Newest“, „Lamest“ und „Start“.
Um zu erkennen, welche Sortiermethode der Nutzer erwünscht, wird auf der jeweiligen Seite einer Variable einen Wert zugewiesen und in ein Switch-Case übergeben.

```
// grab image ids from db, sorted by $sort variable from start page.
//1 = sorted by boring, 2 = sorted by time, else sorted randomly
switch ($sort) {
    case 1:
        $array = fetchImageIds("boringCounter DESC", $loggedUser);
        break;
    case 2:
        $array = fetchImageIds("imageId DESC", $loggedUser);
        break;
    default:
        $array = fetchImageIds("imageId", $loggedUser);
        shuffle($array);
        break;
}
```

Je nach gewünschter Sortierung wird dann der Befehl per String als Parameter in einer Funktion übergeben, ausgeführt und das Ergebnis im Array gespeichert. Es wird auch der Nickname als Parameter übergeben, um die Anzeige eigener Bilder zu vermeiden.

```
<?php
    // fetch all image ids excluding those of a logged user (argument 2), sorted by argument 1
    function fetchImageIds($sorting, $loggedUser) {
        $pdo = new PDO('mysql:host=localhost;dbname=neinGag', 'root', '');
        $sql = "SELECT * FROM images ORDER BY ".$sorting."";
        foreach( $pdo->query($sql) as $row)
            if($row['userName'] != $loggedUser)
                $tmp[] = $row['imageId'];
        return $tmp;
    }
?>
```

Sobald das Array gefüllt ist, wird es über eine Schleife in einem schön verpackten Div-Container ausgegeben.

4.6 Kommentieren / Votes

Nun fehlen nur noch die Kommentare und die Up-/ Downvotes.

Die Votes beginnen ab dem Wert 0, so landet zumindest die erste Information darüber in der Datenbank. Danach wird bei jedem Vote abgefragt, welchen Wert der „boringCounter“ des jeweiligen Bildes hat und dementsprechend hinzuaddiert, oder wie im folgendem Beispiel subtrahiert.

```php
$sql = "SELECT * FROM images WHERE imageId = $_GET[imgID]";
$image = $pdo->query($sql)->fetch();

$count = $image['boringCounter'];
$count--;

$statement = $pdo->prepare("UPDATE images SET boringCounter = ? WHERE imageId = ?");
$statement->execute(array($count, $_GET[imgID]));
```

Beim Absenden wird der Kommentar in eine Variable gespeichert und mit ein paar anderen Informationen der Datenbank hinzugefügt.

```php
if(isset($_POST['text']) && !empty($_POST['text'])){
    $comment = $_POST['text'];

    $sql = "SELECT * FROM users WHERE nickname = '$_SESSION[userid]'";
    $userID = $pdo->query($sql)->fetch();

    $statement = $pdo->prepare("INSERT INTO comments (imageID, nickname, comment) VALUES (?, ?, ?)");
    $result = $statement->execute(array($_GET['imgID'], $userID['nickname'], $comment));
}
```

Durch diese Informationen können wir dafür sorgen, dass die jeweiligen Bilder ihre Kommentare vom richtigen Autor angezeigt bekommen.

```php
// load comments from database
$sql = "SELECT * FROM comments WHERE imageID=".$image['imageId']."";
foreach( $pdo->query($sql) as $row){
    $tmp[] = $row['comment'];
    $tmpNicks[] = $row['nickname'];
}

if(isset($tmp)){
$cnt = 0;
    foreach( $tmp as $comment)
        echo '<div class="singleComment"><b>'.$tmpNicks[$cnt++].":</b> ".$comment.'</div><br/>';
```

5. Zusammenfassung

Im Großem und Ganzem sind wir stolz auf unser Werk. Die Arbeit war es Wert um die Erfahrung zu machen, wie das Ganze von der serverseitigen Position aus funktioniert. Dennoch haben wir noch nicht genügend an Erfahrung gesammelt. Viele weitere Funktionen werden wir zukünftig einbauen, um zu sehen wie die Dinge funktionieren.

6. Ausblick

Die Seite ist noch lange nicht perfekt und kann verbessert werden. Funktional können wir noch viel mehr Bearbeitungsmöglichkeiten bieten, wie Privatsphäre, Soziale Erweiterungen, Profilbearbeitungen und vieles mehr.
Außerdem kann das Aussehen auch noch angepasst werden. Wir haben versucht, so gut wie kein JavaScript zu verwenden, was aber optisch viel mehr Positives beigetragen hätte. Auch hier werden wir uns noch verbessern.
Außerdem werden wir diese Anwendung wahrscheinlich auch fürs Smartphone als App umsetzen, um auch hier die nötigen Erfahrungen zu sammeln.

7. Quellenverzeichnis

1. php-einfach.de: Upload, Login, Registration, Datenbankabfrage, Datenbankaktualisierung und weitere Hilfen.

2. stackoverflow.com: Nachschlagen bei Fehlermeldungen