Moderne Webtechnologien I (WT 1)
Übung: PHP Teil 2)

WiSe 23/24

Prof. Dr. C. Köhn
Daniela Böing
Nils Lentzen

20. November 2023

Website:
https://www.hochschule-bochum.de/fbe/fachgebiete/institut-fuer-informatik/labor-fuer-medienkommunikation-internet-und-robotik/

Hochschule Bochum
Bochum University
of Applied Sciences

MIRO LAB

Inhaltsverzeichnis

1 PHP und Datenbanken

1.1 Einstieg und Wiederholung

In dieser Übung werden Sie neben einer Wiederholung der PHP Grundlagen wie der Objektorientierung Hinweise zur Verwendung von Datenbanken finden.

Übung 1:

Laden Sie sich die Vorgabe (äquivalent mit der Lösung aus der JavaScript-Übung) aus Moodle herunter. Benennen Sie dann alle *.html*-Dateien in *.php*-Dateien um.

Übung 2:

Starten Sie die Vagrantbox wie aus der letzten Übung bekannt. Bedenken Sie, dass Sie einen public Ordner für die Website benötigen.

In der nun folgenden Übung soll das Speichern und Laden von Highscores umgesetzt werden. Erste Vorbereitungen können dazu jetzt schon getroffen werden.

Übung 3:

Erzeugen Sie eine (kleine) Klasse `Score` in einer getrennten PHP-Datei. Diese soll zwei öffentliche Eigenschaften besitzen: `username` und `points`.

Übung 4:

Beim Klick des „Speichern"-Buttons des Formulars soll eine neue Datei *result.php* aufgerufen werden. Als Methode soll jene gewählt werden, die die Daten nicht in der URL anzeigt. In der *result.php*-Datei soll das Formular ausgewertet werden, indem mit den Werten des Formulars (Username, Punkte) ein neues Objekt von `Score` erstellt wird und dessen Werte auf der Seite ausgegeben werden.

Mit PHP ist es möglich, verschiedene Datenbankmanagementsysteme einzubinden, wie

PostgreSQL (wie aus der Datenbank-Vorlesung bekannt) oder MySQL. Da MySQL eine relationale Datenbank ist, die typischerweise im Web verwendet wird, wird diese auch in dieser Übung verwendet.

In PHP gibt es sowohl die `mysql` als auch die `mysqli` und `PDO_MYSQL` API, wobei `mysql` veraltet ist und in PHP 7 entfernt wurde. Wir werden im Folgenden die aktuellste und am meisten eingesetzte API, `PDO_MYSQL`, verwenden.

PDO (**P**HP **D**ata **O**bjects) stellt eine Abstraktionsschicht (im Gegensatz zu `mysqli`) dar, um auf die Daten zuzugreifen. Der wichtigste, daraus resultierende Vorteil ist, dass die Verwendung von PDO die Verbindung sicherer, übersichtlicher und wiederverwendbarer gestaltet, da z.B. „Prepared Statements“ genutzt werden können.

An dieser Stelle sei auch erwähnt, dass für real eingesetzte Anwendungen meist noch eine Abstraktionsschicht höher gegangen wird und sog. *ORM*s[1] verwendet werden, worauf hier jedoch nicht genauer eingegangen wird.

Weitere Informationen zu PDO können Sie zum Beispiel hier finden:

- `https://phpdelusions.net/pdo`
- `http://php.net/manual/de/book.pdo.php`
- `https://www.w3schools.com/php/php_mysql_connect.asp`

1.2 Datenbankverbindung herstellen

Als Erstes muss eine Datenbankverbindung hergestellt werden, wie im Folgenden erklärt wird.

1.2.1 Zugangsdaten zur MySQL Datenbank

Folgende Zugangsdaten sind dabei von Bedeutung:

- Host: `localhost`
- Nutzer: `wt1_ueb`
- Passwort: `abcd`
- Datenbankname: `wt1uebung`

1.2.2 PDO-Verbindung aufbauen

Wenn PDO verwendet werden soll, gibt es verschiedene Möglichkeiten. Wichtig ist im Allgemeinen, dass es jeweils nur eine PDO-Instanz und somit nur eine Datenbankverbindung

[1] **O**bject-**R**elational **M**apping, ermöglicht kurz umrissen das Arbeiten mit Datenbank-Daten mittels auf die Tabelle zugeschnittenen Methoden

gibt. Dies kann zum Beispiel dadurch sichergestellt werden, dass der Verbindungsaufbau „klassenlos“ ausgelagert wird oder eine Klasse, die für den Verbindungsaufbau zuständig ist, zu einem Singleton gemacht wird. Zur Vereinfachung soll hier zunächst auf beide Varianten verzichtet und diese Problematik außen vor gelassen werden.

In PDO findet die Datenbankverbindung mittels vorgenannter Zugangsdaten und dem sog. DSN (**D**ata **S**ource **N**ame) statt. Diese stellt ein Format zur Verbindung z.B. auf Datenbanken dar.

Ein DSN zur Verbindung auf eine MySQL-Datenbank mittels PDO sieht wie folgt aus:

```
1 $dsn = "mysql:host=$host;dbname=$db;charset=utf8mb4";
```

Hier sind natürlich auch noch weitere Parameter möglich, auf die hier nicht genauer eingegangen werden soll.

Anschließend können noch Optionen für die PDO-Instanz festgelegt werden. Hier gibt es eine Vielzahl an Möglichkeiten, am Wichtigsten ist jedoch, dass ein Parameter wie folgt gesetzt ist (für diese Übung, also im Falle einer Entwicklungs- und keiner Produktivumgebung):

```
1 $options = [
2   PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
3 ];
```

Hierdurch wird sichergestellt, dass PDO mittels des PHP-Handlers im Fehlerfalle eine Exception auf dem Bildschirm wirft. Somit müssen in allen anderen Methoden (außer der gleich genannten Ausnahme) keine try-catch-Blöcke mehr verwendet werden. Diese sollten darüber hinaus hier auch nicht verwendet werden, da ein try-catch-Block zur Unterdrückung des Fehlers führt und somit ein Debugging erschwert wird.

Eine Ausnahme davon bildet jedoch zum Beispiel die Initialisierung des PDO-Objekts. Hier sollte stets ein try-catch-Block mit einer eigenen Fehlermeldung verwendet werden, da sonst im Fehlerfall die Credentials auf der Website ausgegeben werden würden.

Wenn das PDO-Objekt dann mittels Optionen und Credentials erzeugt werden soll, sieht dies wie folgt aus:

```
1 try{
2   $this->connection = new PDO($dsn, $username, $password, $options
        );
3 }
4 catch(PDOException $e){
5   echo "Verbindungsaufbau gescheitert: " . $e->getMessage();
6 }
```

Anschließend kann die Variable `connection` für die Datenbankverbindung verwendet werden.

Übung 5:

Zunächst soll die PHP-Klasse `DatabaseConnection` in der Datei `databaseConnection.php` erstellt werden. Diese wird den Zugang zur Datenbank darstellen.

Die Klasse soll eine private Variable besitzen, z.B. `$connection`. Diese Variable speichert die Datenbankverbindung. Im Konstruktor soll diese Variable initialisiert werden.

Der Destruktor soll die Datenbankverbindung wieder schließen, wenn die Klasse nicht mehr verwendet wird. Dazu soll die Variable einfach auf `NULL` gesetzt werden.

1.2.3 Datenbankverbindung testen

Nun soll die gerade hergestellte Datenbankverbindung getestet werden.

Übung 6:

Testen Sie die Datenbankverbindung. Binden Sie dazu die Datenbankklassen-Datei in der Index-Datei ein. Dann erzeugen Sie dort eine Instanz der Klasse. Wird kein Fehler angezeigt (oder eine von Ihnen temporär hinzugefügte Erfolgsmeldung), funktioniert die Datenbankverbindung.

1.3 Datensätze abrufen und ausgeben

Nun soll mit den Daten der Datenbank gearbeitet werden. Dazu müssen diese zunächst von der Datenbank abgerufen werden.

1.3.1 Aufbau der Datenbanktabelle

Die Datenbank besitzt bereits eine `Score`-Tabelle, die wie folgt aussieht:

- `id INT PRIMARY KEY AUTO_INCREMENT,`
- `username VARCHAR(60) NOT NULL,`
- `points INT NOT NULL`

1.3.2 Abfrage der Daten

Um die Daten abzufragen, bietet es sich bei PDO an, eine passende Modellklasse zu erstellen. Dies wurde mit der Klasse `Score` bereits umgesetzt. Mittels dieser Klasse können die Daten einfach aus der Datenbank ausgelesen werden.

Eine mögliche Methode zum Lesen von Daten (die jedoch nur für SQL-Statements ohne Parameter verwendet werden sollte, wie später noch erläutert wird), ist `PDO::query()`. Dieser kann einfach ein SQL-Statement übergeben werden, wie das folgende Beispiel zeigt:

```
1 $sql = "SELECT matrnr, vorname, nachname, semester FROM Student;";
2 $stmt = $this->connection->query($sql);
```

Das Ergebnis dieser Abfrage ist eine Instanz des Typs `PDOStatement`, das in dieser Form nicht direkt ausgegeben werden kann.

Um dies zu tun, müssen die Ergebnisse ausgelesen werden. Dies ist entweder mit der Methode `PDO::fetch()` (Abfrage einer Datenbankzeile) oder `PDO::fetchAll()` (Abfrage aller Datenbankzeilen) möglich. Die Art der Rückgabe hängt vom sog. „Fetch-Mode“ ab, der zum Beispiel der Methode direkt übergeben werden kann. Beispielhaft können folgende Werte verwendet werden:

- `PDO::FETCH_NUM`: gibt ein nummeriertes Array mit den Werten zurück (Zugriff also über `$result[0]`, usw.)
- `PDO::FETCH_ASSOC`: gibt ein assoziatives Array zurück (Zugriff also über `$result[''matrnr'']`, usw.)
- `PDO::FETCH_CLASS`: gibt eine Klasseninstanz, befüllt mit den enthaltenen Werten, zurück (der Konstruktor wird erst im Anschluss aufgerufen)

Im Folgenden soll der letzte Modus getestet werden. Diesem muss dann als zusätzlicher Parameter der Klassenname übergeben werden, also in diesem Falle `Score`. Für die erste Zeile des Ergebnisses sieht dies wie folgt aus:

```
1 $highscores = $stmt->fetchAll(PDO::FETCH_CLASS, 'Score');
```

Wie bereits erwähnt gibt diese Methode nun ein Array an Ergebnissen zurück, über das im Weiteren zum Beispiel mit einer foreach-Schleife iteriert werden kann.

Übung 7:

Erweitern Sie die `DatabaseConnection`-Klasse, indem Sie eine Methode zum Abrufen der Daten der Score-Tabelle hinzufügen. Verwenden Sie dazu die oben genannten Methoden und geben das Array als Rückgabewert der Methode zurück.

1.3.3 Ausgabe der Daten

Die Highscores aus der Datenbank stehen jetzt zur Verfügung und müssen nun in der Highscore-Tabelle formatiert ausgegeben werden.

Übung 8:

Schreiben Sie die Highscores mittels einer Schleife in die dafür vorgesehene Tabelle.

1.4 Datensätze manipulieren

Neben der Abfrage können auch Datensätze der Datenbank hinzugefügt oder daraus gelöscht werden. Im Rahmen dieser Website ist vor allem das Hinzufügen von Scores interessant.

1.4.1 Hinzufügen von Datensätzen

Ein Hinzufügen von Datensätzen ist für das vorliegende Beispiel mit folgendem SQL-Statement möglich:

```
1 INSERT INTO Score (id, username, points) VALUES(NULL, '<username>'
    , '<points>');
```

Hier wird ersichtlich, dass Usereingaben in die Datenbank geschrieben werden. Dies ist generell ein kritischer Vorgang (Stichwort „SQL-Injection“), weswegen hier nicht die `PDO::query()`-Methode verwendet werden sollte.

Um solche Fälle zu händeln, gibt es die beiden Methoden `PDO::prepare()` und `PDO::execute()`. Hierzu werden sog. „Prepared Statements“ benötigt, also alle Felder im SQL-Statement, die nachher durch Usereingaben gefüllt werden sollen, werden durch einen Platzhaltertext ersetzt. Die jeweiligen Werte werden dann als (assoziatives) Array der `execute`-Methode übergeben. Dadurch wird verhindert, dass zum Beispiel gewisse Sonderzeichen verwendet werden können, um die Datenbank unauthorisiert auszulesen oder zu verändern. Dies kann für das vorliegende Beispiel wie folgt aussehen:

```
1 $sql = "INSERT INTO Score (id, username, points) VALUES (NULL, ?,
    ?);";
2 $stmt = $this->connection>prepare($sql);
3 $result = $stmt->execute([$username, $points]);
```

Übung 9:

Fügen Sie die Methode zum Einfügen eines Scores hinzu und verwenden Sie dabei vorgenannte Methoden und das SQL-Statement. Überprüfen Sie bitte vor dem Einfügen, ob alle Variablen Werte enthalten und es sich bei den Punkten tatsächlich um eine (gültige) Zahl handelt.

1.4.2 Anpassungen der *results.php*

Nach der Implementierung der Insert-Methode kann diese nun verwendet werden.

Übung 10:

Erzeugen Sie nun auf der *results.php* ebenfalls eine Datenbankverbindung. Rufen Sie (statt der Erzeugung und Ausgabe der `Score`-Klasse) nun die Insert-Methode mit den Post-Parametern auf. Nutzen Sie den Rückgabewert der Methode, um eine passende Meldung auf der Seite auszugeben (Erfolg oder Misserfolg der Speicherung).