# Übungsaufgabe 26 - Einbinden der active-record-Klasse `User` in das Projekt Benutzerverwaltung

## 26.a) Projektinitialisierung Projekt Benutzerverwaltung

i) → folgendes nur machen falls noch nicht geschehen

Lade Dir die Datei `benutzerverwaltung.zip` aus der Cloud und entpacke diese in Deinem Projektordner für das Projekt Benutzerverwaltung (z. B. `C:\xampp\htdocs\projekte\benutzerverwaltung`). Falls Du schon Dateien in dem Verzeichnis hast, kannst Du diese entweder löschen oder irgendwo anders archivieren.

Du solltest eine ähnliche Verzeichnisauflistung mittels dem Command `dir` erhalten

```
Verzeichnis von C:\xampp\htdocs\projekte\benutzerverwaltung

11.06.2024  15:10    <DIR>          .
11.06.2024  15:10    <DIR>          ..
09.06.2024  16:41    <DIR>          archiv
09.06.2024  15:53    <DIR>          css
16.06.2023  16:27    <DIR>          fonts
09.06.2024  12:30    <DIR>          inc
09.06.2024  16:38             1.347 index.php
09.06.2024  16:28             2.983 loggedin.php
09.06.2024  13:34               478 logout.php
09.06.2024  15:54             3.603 user_edit.php
09.06.2024  15:50             2.055 user_list.php
09.06.2024  16:45             4.079 user_login.php
09.06.2024  14:36             7.722 user_register.php
               7 Datei(en),         22.267 Bytes
               6 Verzeichnis(se), 743.515.742.208 Bytes frei
```

ii) → ab hier dann machen

Kopiere Dir nun unsere active-record-Klasse `User` aus dem Workshop zu Lektion 10 (oder aus der Cloud `/phpOOP/lek10/inc/User.php`) in das Verzeichnis `benutzerverwaltung\inc`.
(Die bestehende Datei kannst Du überschreiben/ersetzen - diese ist im Archiv `/projekte/benutzerverwaltung/archiv/versions/benutzerverwaltung02.zip` bereits versioniert).

iii) Refactoring der Klasse `User`

- Entferne zunächst überflüssige Kommentare, die Du nicht mehr benötigst
- bei einigen Methoden war der Return-Type nicht richtig, hier muss ggf. ergänzt werden, d. h. immer wenn die Instanz zurückgeben wird muss um `self` ergänzt werden
  (z. B. `public function update(): self|bool`)

Da wir jetzt nach Aufgabe 25 Defaultwerte für die Attribute `role`, `created_at` und `updated_at` im DBMS setzen, müssen wir diese nicht mehr in unseren active-record-Methoden handhaben.

- in der Methode `insert()` kann somit folgender Code gelöscht werden oder auskommentiert werden

```
// in Methode insert()

        # Logik für eventuelle Defaultwerte oder fehlende Wert hinzufügen (sofern
nicht im DBMS gehandhabt), z. B. die Timestamps und die Rolle
        $data['role'] = 'inactive';
        $data['created_at'] = (new DateTime())->format('Y-m-d H:i:s');
        $data['updated_at'] = (new DateTime())->format('Y-m-d H:i:s');
```

- stattdessen könnten wir (optional) sicherstellen, dass diese Wert nicht gesetzt werden, also nur im DBMS, indem wir die key-value-Paare im Datenarray löschen

```
// in Methode insert() - nach obigen Code einfügen

        # optional - sicherstellen, dass alle key-value-Paare gelöscht werden, für
die im DBMS automatisch ein Wert gesetzt wird
        unset($data['id']);
        unset($data['role']);
        unset($data['createdAt']);
        unset($data['updatedAt']);
```

- in der Methode `update()` hatten wir den Timestamp für die Column `updated_at` manuell erzeugt. Das brauchen wir jetzt nicht mehr, da wir in Aufgabe 25 ein entsprechendes Constraint im DBMS gesetzt haben. Folgende Zeile kann somit gelöscht oder auskommentiert werden

```
// in Methode update()
        $data['updated_at'] = (new DateTime())->format('Y-m-d H:i:s');
```

- (optional) wir wollen jedoch sicherstellen, dass die Timestamps nicht geändert werden können, also nur das DBMS und nicht über die Methode. Deswegen löschen wir auch hier die Timestamps indem wir die key-value-Paare im Datenarray löschen

```
// in Methode update() - nach obigen Code einfügen

        # optional - die Timestamps sollen nicht änderbar sein
        unset($data['createdAt']);
        unset($data['updatedAt']);
```

Da wir die Timestamps nun ja ausschließlich im DBMS handhaben, müssen wir in den beiden Methode `insert()` und `update()` unsere Instanz jetzt zwingend mit den Daten in der Datenbank abgleich, d. h. wir müssen jetzt die geänderten persistierten Daten aus der Datenbank auslesen und unsere Instanz dann akualisieren. Am einfachsten geht das, in dem wir den Datensatz neu abfragen und alle Attribute neu setzen.

- in der Methode `insert()` kann somit folgender Code gelöscht werden oder auskommentiert werden

```
// in Methode insert()

        # geänderte Instanz zurückgeben bzw. die Instanz aktualisieren mit den
persititen Daten
        $data['id'] = $id ?? NULL;
        $this->setAttributes($data);
        // wenn in der Datenbank irgendwelche Defaultwerte gesetzt werden, dann
müsste der Datensatz neu eingelesen werden
```

- hierfür folgenden Code einfügen

```
// in Methode insert() - nach obigen Code einfügen

        # den erstellten Datensatz aus der Datenbanktabelle auslesen und die
Instanz selbst aktualisieren
        $sql = 'SELECT * FROM `users` WHERE id = ?';
        try {
            $PDOStatement = $db->prepare($sql);

            $PDOStatement->execute([$id]);
            $data = $PDOStatement->fetch();
        } catch (PDOException $e) {
            echo $e->getMessage(), PHP_EOL;
            // hier ggf. Header-Weiterleitung auf Fehlerseite
            exit;
        }

        # Instanz aktualisieren
        if($data) $this->setAttributes($data);

        return $this;
```

- analog, in der Methode `update()` kann somit folgender Code gelöscht werden oder auskommentiert werden

```
// in Methode insert()

        # geänderte Instanz zurückgeben bzw. die Instanz aktualisieren mit den
persititen Daten

        /*  */$this->setUdpatedAt($data['updated_at']);
        // wenn in der Datenbank irgendwelche Defaultwerte gesetzt werden, dann
müsste der Datensatz neu eingelesen werden
```

- hierfür folgenden Code einfügen

```
// in Methode update() - nach obigen Code einfügen

        # den geänderten Datensatz aus der Datenbanktabelle auslesen und die
Instanz selbst aktualisieren
        $sql = 'SELECT * FROM `users` WHERE id = ?';
        try {
            $PDOStatement = $db->prepare($sql);

            $PDOStatement->execute([$this->id]);
            $data = $PDOStatement->fetch();
        } catch (PDOException $e) {
            echo $e->getMessage(), PHP_EOL;
            // hier ggf. Header-Weiterleitung auf Fehlerseite
            exit;
        }

        # Instanz aktualisieren
        if ($data) $this->setAttributes($data);

        return $this;
```

- bei der `delete()`-Methode hatten wir die Id bei der Aktualisierung der Instanz auf `NULL` gesetzt, so dass diese wie ein neuerstelltes und noch nicht persistiertes Objekt der Klasse User im Code weiter genutzt werden kann. Allerdings sollten dann auch die Timestamps und die Rolle auf `NULL` gesetzt werden, damit es wirklich wieder wie ein neues Objekt erscheint. Wir ergänzen also die wie folgt:

```
// in Methode delete() - nach $this->id = NULL;

        $this->role = NULL;
        $this->updatedAt = NULL;
        $this->updatedAt = NULL;
```

⇒ das sollte es vorläufig gewesen sein.

iv) → Testscript erstellen

Teste nun die Änderungen, in dem Du ein einfaches Testscript namens `test_user.php` erstellst, in dem Du

- einen neuen Datensatz erstellst
- einen Datensatz änderst
- einen Datensatz löschst

## 26.b) Implementieren einer active-record-Methode `findByEmail()`

Wir wollen die active-record-Klasse `User` im Folgenden (Aufgabe 27) in unser Projekt Benuzterverwaltung einbinden und dort verwenden. Da wir einen User i. R. über seinen Loginnamen, sprich die E-Mail-Adresse suchen, ist es hilfreich gleich eine entsprechende Methode zu implementieren.

Erstelle ein Methode `findByEmail()`, die uns einen Datensatz anhand der E-Mail-Adresse abfragt und als Instanz der Klasse User zurückgibt.

> *Anmerkung:* da wir eine neue Instanz mit dem Datensatz erhalten wollen, ist es vorteilhaft die Methode statisch zu implementieren, damit wir nicht erst ein Objekt erzeugen müssen, um die Methode aufrufen zu können.

Die Signatur der Methode lautet dann:

```php
public static function findByEmail(string $email): self|NULL
```

→ teste die Methode in dem Script `test_user.php`

> *Anmerkung:* alternativ könnte man auch die Methode find() parametrisieren, d. h. optional noch die gewünschte column angeben
>
> ```php
> public static function find(mixed $value, string $column = 'id'): self|NULL`
> ```
>
> Des Weiteren könnten bei Bedarf beliebige findBy-Methoden mittels entsprechender Logik in der magischen Methode `__call()` dynamisch erzeugt werden.
>
> → wer Lust hat, kann das ja mal implementieren