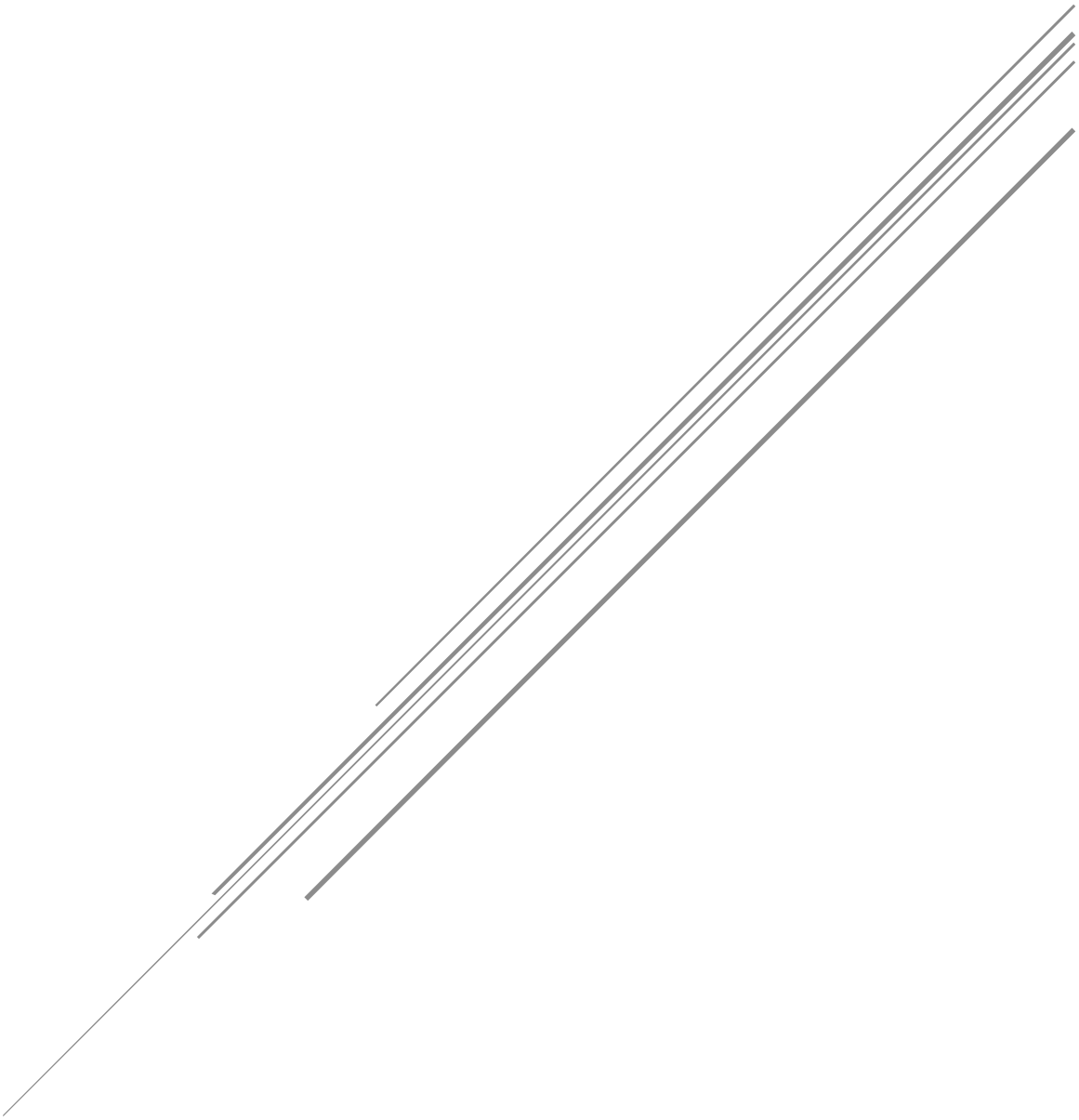
SAPV2

Projekt Dokumentation



Lukas Breiter, Marco Rampinelli

Modul 151 Datenbanken in Web-Applikation einbinden

BBZ BL
GRUNDBILDUNG
WEITERBILDUNG

Autor: Lukas Breiter, Marco Rampinelli
Version: 2.0
Datum: 29.06.2022

Datum	Beschreibung	Status	Autor	Version
15.5.2022	Erstellung	Erledigt	M	0.1
22.5.2022	Fertigstellung	Erledigt	M&L	1.0

SAPv2

Wir werden als Projekt die Benutzerverwaltung von SAP ganzheitlich erneuern.

Ausgangssituation / Problembeschreibung:

Wir möchten ein Benutzerverwaltungssystem programmieren, bei welchem nur der Admin Benutzer hinzufügen kann. Der einzelne Benutzer kann seine eigenen Benutzerdaten anpassen und sogar ein Profilbild einfügen. Benutzer können nicht andere Benutzer verändern. Wir brauchen dieses System, da unser aktuelles System SAP bereits, seit mehr als 10 Jahren veraltet ist.

Ist Situation:

- *Benutzerverwaltungssystem*
- *Benutzer Berechtigungen*
- *Admin Benutzer*
- *Benutzer in Datenbank abgespeichert*
- *Benutzerübersicht*
- *Benutzer können sich selbst bearbeiten*
- *Admin kann Benutzer hinzufügen*
- *Admin kann Benutzer löschen*
- *Admin kann Benutzer bearbeiten*
- *Admin Oberfläche*

Dokumentation

C3: In meinem Projekt werden alle Benutzereingaben clientseitig validiert.
In diesem Projekt wurde all Benutzereingaben clientseitig validiert, indem bei den Benutzer Inputs eine max. länge sowie das Feld required gesetzt wurden:

```
<input type="text" name="firstname" class="form-control" id="firstname" value="<?php echo $firstname ?>" placeholder="Geben Sie Ihren Vornamen an." maxlength="30" required>
```

Um Passwörter zu validieren wird auf der Clientseite Regex verwendt. Wir wählten den Expression: "(?=^.{8,}$)((?=.*\d+)(?=.*\W+))(?![.\n])(?=.*[A-Z])(?=.*[a-z]).*$". Dieser ist zwar nicht perfekt, eignet sich aber ganz gut für unsere Zwecke. (min. 8 Zeichen, min.1 Zahl, kein leerschlag, min.1 Grossbuchstabe, min.1 Kleinbuchstabe). Passwörter haben eine max. length von 255 und Emails haben eine maximallänge von 100 Zeichen.

C4: In meinem Projekt werden alle Benutzereingaben serverseitig validiert.

```php
// Vorname ausgefüllt?
if (isset($_POST['firstname'])) {
  //trim
  $firstname = trim($_POST['firstname']);

  //mindestens 1 Zeichen und maximal 30 Zeichen lang
  if (empty($firstname) || strlen($firstname) > 30) {
    $error .= "Geben Sie bitte einen korrekten Vornamen ein.<br />";
  }
} else {
  $error .= "Geben Sie bitte einen Vornamen ein.<br />";
}
```

Die Serverseitige Validation wir bei uns durch Überprüfungen in PHP umgesetzt. Zuerst wird für jede Variable geprüft ob sie gesetzt wurde und anschließend wird geprüft ob sie nicht nur Leerzeichen enthält oder die maximale länge überschreiten. Dies wird für jeden Input gemacht. Zusätzlich überprüfen wir immer wenn ein Benutzername erstellt oder geändert wird, ob dieser bereits in der Datenbank existiert. Dazu wird der Benutzer in der DB gesucht, und wenn ein leeres Ergebnis zurückkommt geht das

```php
//Check if username already exists.
$ckeckquery = "SELECT * FROM `users` WHERE username=?";
$stmt = $mysqli->prepare($ckeckquery);
$stmt->bind_param("s", $username);
$stmt->execute();
$result = $stmt->get_result();
$row = $result->fetch_assoc();

if ($row != NULL) {
  $error .= "Benutzername bereits vorhanden, bitte wählen sie einen anderen.";
}
```

ganze Weiter. Beim Daten ändern, wenn der Benutzername nicht geändert wurde war das ein wenig schwieriger, denn es gibt ja immer einen Benutzer mit dem eigenen Benutzern Namen in der DB – nämlich der User selbst dies lösten wir indem überprüft wird ob die ID des zu bearbeitenden Users mit der ID welche zurückgeliefert wird wenn ein passender Username gefunden wird übereinstimmt.

C5: Script-Injection wird in meinem Projekt konsequent verhindert.

Wenn immer etwas aus der Datenbank geholt wird, was von einem Benutzer erstellt

```php
while ($row = $result->fetch_assoc()) {
    echo "<tr>";
    echo "<td>" . htmlspecialchars($row['firstname']) . "</td>"
    echo "</tr>";
}
```

, und anschliessend verwendet wird, werden die Daten zuerst gereinigt oder bei IDs wird überprüft, ob diese Numerische werte haben, ansonsten wird direkt abgebrochen. Die Daten werden erst beim Holen

```php
!isset($_GET["id"]) or !is_numeric($_GET["id"]))
```

gereinigt, um zu verhindern das ein langer Name, welcher zusätzlich Sonderzeichen enthält nicht die Zeichenlimite von 30 auf der DB überschreitet.

C6: In meinem Projekt wird das Session-Handling korrekt eingesetzt. Ein angemeldeter Benutzer hat Zugriff auf weitere Funktionen, welche nicht angemeldeten Benutzern verwehrt bleiben.

In jedem File wird als erstes eine Session gestartet. Nach dem Login wird dann

```php
<?php

session_start();
```

personalisiert mit allen Daten welche später im Code wieder gebraucht werden. Die Session neu gestartet, um bestimmte Angriffe zu verhindern. Benutzer, welche eingeloggt sind bekommen zugriff auf ganz neue Features & die Navigationsbar passt sich je nach Login Status & Rechte, die der Benutzer hat an. Diese Infos werden aus den Daten, welche in der Session gespeichert sind bezogen.

```php
// Session personifizieren
$_SESSION['loggedin'] = true;
$_SESSION['username'] = $row['username'];
$_SESSION['userid'] = $row['id'];
$_SESSION['admin'] = $row['admin'];

// Session ID regenerieren
session_regenerate_id(true);
```

```php
<?php
if (!isset($_SESSION['loggedin']) or !$_SESSION['loggedin']) {
    echo '<li class="nav-item"><a class="nav-link" href="register.php">Registrierung</a></li>';
    echo '<li class="nav-item"><a class="nav-link" href="login.php">Login</a></li>';
} else {
    if ($_SESSION['admin'] == 1) {
        echo '<li class="nav-item"><a class="nav-link" href="register.php">Neuer Benutzer</a></li>';
    }
    echo '<li class="nav-item"><a class="nav-link" href="admin.php">Benutzerliste</a></li>';
    echo '</ul></div>';
    echo '<div class="dropdown ms-auto">';
    echo '<button class="btn btn-default dropdown-toggle" type="button" id="menu1" data-toggle="dropdown" st
    echo '<img src="https://www.innovaxn.eu/wp-content/uploads/blank-profile-picture-973460_1280.png" alt="P
    echo '<span class="caret"></span>';
    echo '</button>';
    echo '<ul class="dropdown-menu dropdown-menu-right" role="menu" aria-labelledby="menu1">';
    if (!$_SESSION['admin'] == 1) {
        echo '<li class="nav-item"><a class="dropdown-item" href="password.php">Passwort ändern</a></li>';
    } else {
        echo '<li class="nav-item"><a class="dropdown-item" href="overview.php">Meine Benutzer</a></li>';
        echo '<li class="nav-item"><a class="dropdown-item" href="password.php">Passwort ändern</a></li>';
    }
    echo '<li class="nav-item"><a class="dropdown-item" href="logout.php">Logout</a></li>';
}
```

C7: Ein angemeldeter Benutzer kann sich wieder abmelden. Die Session wird dabei korrekt beendet.

Eingeloggte Benutzer haben in der Navigationsbar die Möglichkeit sich auszulagern. Dabei werden sie auf ein PHP File weitergeleitet, in welchem der Session Array gelehrt wird & anschließend die Session zerstört wird.

```php
echo '<li class="nav-item"><a class="dropdown-item" href="logout.php">Logout</a></li>';
```

```php
<?php
// Session starten
session_start();
// Session leeren
$_SESSION = array();
session_destroy();
// Weiterleiten auf login.php
header('Location: login.php');
```

C8: In meinem Projekt wird Session-Fixation und Session-Hijacking erschwert.

Durch das Regenerieren der Session ID direkt nach dem Personalisieren, machen wir diese Spezifischen Angriffs Methoden schwieriger.

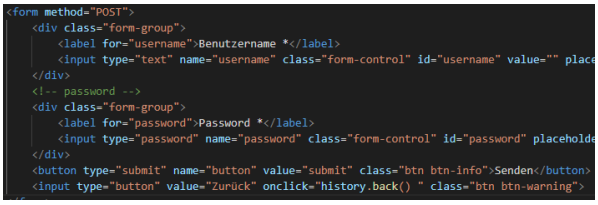
```php
// Session ID regenerieren
session_regenerate_id(true);
```

C10: Ein Benutzer kann sich an meinem Projekt registrieren. Dazu erfasse ich sinnvolle Daten des Benutzers.

Die Benutzer können sich beim Besuchen der Registrierungsseite neu registrieren. Dabei werden Vorname, Nachname, E-Mailadresse, Benutzername und Password aufgenommen und in der DB gespeichert.

```php
<!-- password -->
<div class="form-group">
  <label for="password">Password *</label>
  <input type="password" name="password" class="form-control" id="password" placeholder="Gross- und Kleinbuchstaben, Zahlen, Sonderzeichen, min.
</div>
<!-- Send / Reset -->
<button type="submit" name="button" value="submit" class="btn btn-primary">Erstellen</button>
<input type="button" value="Zurück" onclick="history.back() " class="btn btn-warning">
```

C11: Ein Benutzer kann sich an meinem Projekt anmelden. Nach der Anmeldung stehen dem Benutzer weitere Funktionen zur Verfügung.



Wenn ein Benutzer sich einloggt, nachdem er registriert wurde, sieht er zusätzliche Optionen, abhängig von seinen rechten. Auch wird ihm ein Profilbild (aktuell statisch) angezeigt, über welches er weitere Optionen

```php
if (!isset($_SESSION['loggedin']) or !$_SESSION['loggedin']) {
    header('Location:  login.php');
```

erhält. Eingeloggte Benutzer sehend folgende Seiten, welche ein nicht eingeloggter User nicht sieht: admin/Benutzerübersicht, overview/Benutzerspezifische Übersicht, Edit, neuen Benutzer erstellen, Passwort ändern.
Ein nicht eingeloggter User kann auf login.php zugreifen, ein eingeloggter kann dies nicht.

C12: In meinem Projekt kann ein angemeldeter Benutzer sein Passwort ändern.

Damit ein Eingeloggter Benutzer sein Passwort ändern kann, muss er zuerst sein aktuelles Passwort bestätigen, und anschließen zwei Mal das gleiche neue Passwort eintippen. Wenn das alles erfüllt wird, wird sein Passwort geändert. Das neue Passwort wird wieder gehasht und überschreibt das alte passwort.

```php
if ($row = $result->fetch_assoc()) {

    // Passwort ok?
    if (password_verify($oldpassword, $row['password'])) {
        if (strcmp($newpassword, $repeatnewpasswort) !== 0) {
            $error .= "Passwort stimmt nicht überein";
        }
    } else {
        $error .= "Falsches Passwort";
    }
} else {
    $error .= "Benutzername oder Passwort sind falsch";
}
```

C13: In meinem Projekt können angemeldete Benutzer weitere Informationen in der Datenbank erfassen.

Benutzer, welche sich selbst registriert haben, können unter dem Tab "Neuer Benutzer" einen neuen Benutzer erfassen, welcher dann weniger rechte hat. Dies wird über denn Tinyint "admin" geregelt. Unter "Creator" wird die ID des Benutzers gespeichert, welchen diesen erstellt hat, bei selbst Registrierung ist dieser Wert NULL.

```php
// überprüfen wie der Benutzer erstellt wurde - verteilen der Rechte
if (!isset($_SESSION['loggedin']) or !$_SESSION['loggedin']) {
  $admin = 1;
  $creator = NULL;
} else {
  $admin = 0;
  $creator = $_SESSION['userid'];
}
```

C14: Diese Datensätze können ausschliesslich vom Ersteller dieser Datensätze einzeln geändert werden.

Falls ein Benutzer einen Datensatz bearbeiten will, wird die abgespeicherte ID des eingeloggten Users mit der ID verglichen, welche beim Ziel unter Creator gespeichert ist. Wenn dies übereinstimmt, kann das Ziel bearbeitet werden, ansonsten wird der Admin zurück zur Übersicht geleitet. Dies wird überprüft, indem geschaut wird, ob die SQL Query einen leeren Datensatz zurückgibt. So wird auch gleich verhindert, dass nichtexistierende Datensätze bearbeitet werden.

```php
$id = htmlspecialchars($_GET["id"]);
$query = "SELECT * FROM `users` WHERE id=" . $id . " AND creator=" . $_SESSION['userid'];
```

```php
if (!isset($User['username'])) {
  header('Location: overview.php');
}
```

C15: Diese Datensätze können ausschliesslich vom Ersteller dieser Datensätze einzeln gelöscht werden.

Beim Löschen passiert mehr oder weniger das gleiche wie beim Bearbeiten des

```php
$query = "DELETE FROM users WHERE id=? and creator=?";
$stmt = $mysqli->prepare($query);
```

Benutzers, nur die Query sieht ein wenig anders aus.

Um das Löschen von nichtexistierenden Datensätzen oder das Löschen von Datensätzen, auf welche der aktuell eingeloggte User keinen Zugriff hat zu

```php
} else {
    // Anzahl betroffener Zeilen, grösser als 0?
    if ($mysqli->affected_rows) {
        $message .= 'Datensatz erfolgreich gelöscht.<br>';
        header('Location: overview.php');
    } else {
        $error .= "<h1>Unzureichende Berechtigung</h1>";
    }
```

verhindern wird wieder auf Creator überprüft und falls eine leere Antwort von der DB kommt eine Fehlermeldung ausgegeben.

C16: In meinem Projekt wird SQL-Injektion konsequent verhindert.

```php
// Query erstellen
$query = "SELECT id, username, password, admin from users where username = ?";

// Query vorbereiten
$stmt = $mysqli->prepare($query);
if ($stmt === false) {
    $error .= 'prepare() failed ' . $mysqli->error . '<br />';
}
// Parameter an Query binden
if (!$stmt->bind_param("s", $username)) {
    $error .= 'bind_param() failed ' . $mysqli->error . '<br />';
}
// Query ausführen
if (!$stmt->execute()) {
    $error .= 'execute() failed ' . $mysqli->error . '<br />';
}
```

Wir verwenden immer, wenn eine Benutzereingabe möglich ist, prepared Statements, so kann es nicht zu SQL-Injektion kommen.

Abnahme:

Kunde:
Daniel Brodbeck

Projektleitung:
Lukas Breiter & Marco Rampinelli