# Dokumentation der Eigenleistung

Christoph Niederer Matr. Nr. 6990752

Die nachfolgende Dokumentation wird die Funktion der „Benutzer Verwaltung“ erklären. Es werden folgende Aspekte beleuchtet:

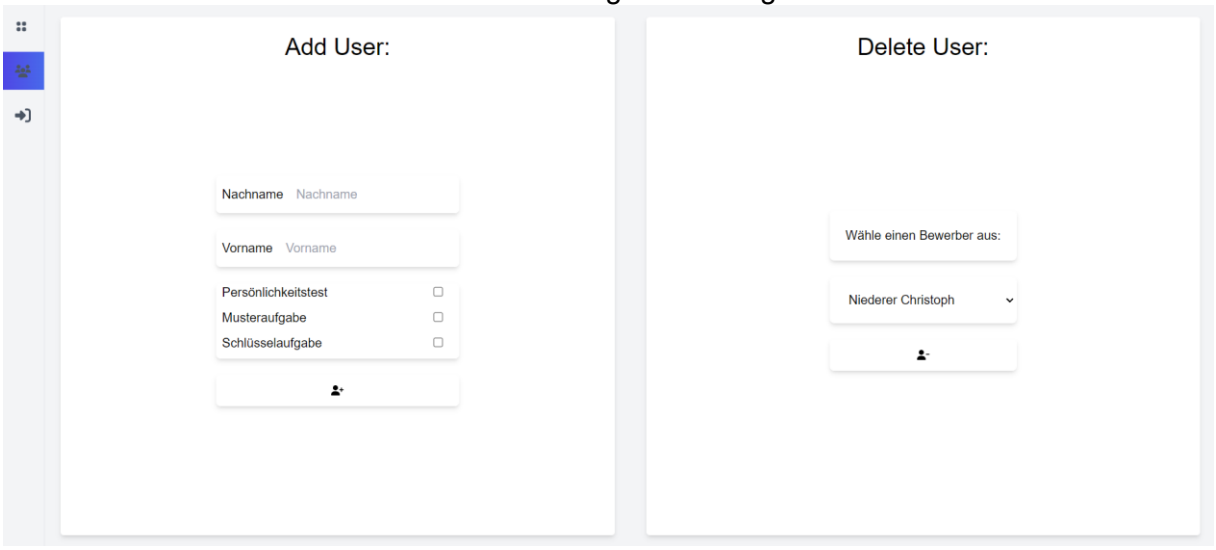


# 1.1 Beschreibung der Funktion

Die Funktion „Benutzer Verwaltung“, im Backend auch bekannt als „ManageUsers“ ist eine Funktion die lediglich für die Ausbilder relevant ist. Sobald sich ein Ausbilder angemeldet hat, hat er in der Navigationsleiste den Punkt „Manage Users“. Er wird dann auf eine HTML Seite weitergeleitet in der er dann neue Nutzer anlegen kann. Hierfür gibt er Vor- sowie Nachnamen des Nutzers ein und welche Tests er absolvieren muss. Der Ausbilder hat allerdings nicht nur die Möglichkeit einen Nutzer anzulegen, er kann auch Nutzer löschen, wenn dieser z.B nicht mehr benötigt wird. Der neue Nutzer wird nach betätigen des Knopfes sofort in der Datenbank gespeichert, oder beim entfernen des Nutzers sofort aus der Datenbank gelöscht.

# 1.2 Einbindung in das Gesamtprojekt

Die Funktion ist wie bereits erwähnt lediglich für die Ausbilder. Die Nutzer welche den Asessment Test durchlaufen bekommen von dieser Funktion nichts mit. Der Ausbilder meldet sich mit seinen Login Daten an und bekommt dann ein erweitertes Dashboard angezeigt. Dort kann er dann weiter zur Nutzer Verwaltung die wie folgt aussieht:



Man hätte natürlich die User auch manuell in die Datenbank eintragen können, doch durch diese Funktion spart sich der Ausbilder einen aufwändigen Prozess. Er muss lediglich Namen sowie die Tests die zu absolvieren sind eintrage. Automatisch werden dann im Hintergrund die richtigen Tabellen ausgefüllt, eine „UserID“ wird erstellt sowie ein individueller Anmeldetoken für den Nutzer. Beim Löschen eines Nutzers, kann der Ausbilder den passenden Nutzer auswählen und auf einen Knopf drücken und alle Einträge des Nutzers werden sofort gelöscht. Diese Funktion erleichtert also dem Ausbilder die Erstellung der Nutzer. Diese Funktion konnte bereits gegen Anfang des Gesamtprojekts implementiert werden, da es kaum Abhängigkeiten gab. Der Login musste zuvor implementiert werden, danach dann diese Funktion.

# 1.3 Idee, Funktionsweise und Angabe des Codes

Frontend:

```
<main class="flex-auto grid grid-cols-2 gap-10 h-screen">
    <div class="bg-white rounded shadow-md my-5 ml-5 flex flex-col">
        <h1 class="text-center pt-5 text-3xl">Add User:</h1>
        <div class="h-full flex justify-center items-center max-h-full">
            <form method="post" action="/ManageUsers" enctype="application/x-www-form-urlencoded">
                <fieldset class="bg-white rounded-md shadow-md mb-5">
                    <label class="pl-2">
                        Nachname
                    </label>
                    <input class="rounded-md p-3" type="text" name="nachname" placeholder="Nachname" />
                </fieldset>
                <fieldset class="bg-white rounded-md shadow-md mb-5">
                    <label class="pl-2">
                        Vorname
                    </label>
                    <input class="rounded-md p-3" type="text" name="vorname" placeholder="Vorname" />
                </fieldset>
                <fieldset class="bg-white rounded-md shadow-md mb-5">
                    <div class="flex justify-between items-center mb-2">
                        <label class="pl-2">Persönlichkeitstest</label>
                        <input type="checkbox" id="personality" name="personality" value="1" class="mr-5">
                    </div>
                    <div class="flex justify-between items-center mb-2 ">
                        <label class="pl-2">Musteraufgabe</label>
                        <input type="checkbox" id="pattern" name="pattern" value="1" class="mr-5">
                    </div>
                    <div class="flex justify-between items-center mb-2">
                        <label class="pl-2">Schlüsselaufgabe</label>
                        <input type="checkbox" id="key" name="key" value="1" class="mr-5">
                    </div>
                </fieldset>
                <fieldset class="bg-white rounded-md shadow-md mb-5">
                    <div class="flex items-center justify-center">
                        <button class="rounded-md p-3 w-full flex justify-center" type="submit" name="Button" value="addButton">
                            <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 640 512" class="w-4 h-4">
                                <path d="M96 128a128 128 0 1 1 256 0A128 128 0 1 1 96 128zM0 482.3C0 383.8 79.8 304 178.3 304h91.4C368.2 304 448 383.8 448 482.3c0 16.4-13.3 29.7-29.7 29.7H29.7C13.3 512 0 498.7 0
                            </svg>
                        </button>
                    </div>
                </fieldset>
            </form>
        </div>
    </div>
```

Code für das Hinzufügen eines Nutzers

Zuvor wird der Header sowie die Navigationsleiste für alle Seiten implementiert. Daher ist der gesamte Rest der Seite innerhalb des <main> Tags. Das erste <Div> ist für das gesamte weiße Viereck. Zum Eintragen der Daten wird eine Form benutzt welche nach absenden die Daten direkt an die passende „Route“ im Backend sendet. Das erste Fieldset ist für den Nachnamen, das zweite für Vorname. Fieldset drei beinhaltet die Informationen welche Tests der Nutzer machen muss. Als Abschluss dann der Knopf um abschicken der Daten. Die Klassen sind für das Design verantwortlich da wir Tailwind nutzen.


Code für das Löschen eines Nutzers

<Div> ist wieder für die große Fläche verantwortlich. Es wird wieder eine Form genutzt die alle Daten an das Backend sendet. Dem <select> Tag werden die passenden Daten vom Backend gesendet damit dort alle Nutzer welche sich in der Datenbank befinden angezeigt werden. Als „value“ wird die ID der Nutzer gesetzt, angezeigt werden Nach- und Vorname. Beim auswählen eines Nutzers werden dann nach Knopfdruck alle Daten der passenden UserID gelöscht.

Diese Funktion benötigt kein JS, da alle Daten durch die Form an das Backend gesendet werden.

Backend:

```python
@app.route('/ManageUsers', methods= ['POST', 'GET'])
def ManageUsers():
    session.pop('profile_loaded', None)
    if session.get('loggedIn'):
        db = getDB()
        cur = db.cursor()
        #Add User
        pressed_button = request.form.get('Button')
        if request.method == "POST" and pressed_button == "addButton":
            add_User(db,cur)
```

Backend Code für das hinzufügen eines Nutzers

Dies ist die passende Route für die Benutzerverwaltung. Zuerst wird überprüft ob man eingeloggt ist um sicherzustellen, dass man ohne Login nicht auf diese Seite kommt. Danach wird die Verbindung zur Datenbank hergestellt. Da sich zwei Knöpfe auf der Seite befinden wird danach überprüft welcher gedrückt wurde um die Richtige Funktion auszuführen. In diesem Fall der „addButton“. Danach wird dann die eigentliche Funktion zum hinzufügen eines Benutzers aufgerufen.

```python
def add_User(db, cur):
    name = request.form.get('nachname')
    firstname = request.form.get('vorname')
    personality = request.form.get('personality')
    pattern = request.form.get('pattern')
    key = request.form.get('key')
    if not personality:
        personality = "0"
    if not pattern:
        pattern = "0"
    if not key:
        key = "0"

    tokens = cur.execute('SELECT (Token) FROM User')
    tokens = tokens.fetchall()
    tokens_values = [row[0] for row in tokens]
    token = generate_token_ID()
    while token in tokens_values:
        token = generate_token_ID()

    ids = cur.execute('SELECT (ID) FROM User')
    ids = ids.fetchall()
    id_values = [row[0] for row in ids]
    id = generate_token_ID()
    while id in id_values:
        id = generate_token_ID()

    cur.execute('INSERT INTO USER (Nachname, Vorname, Token, ID, Persönlichkeitstest, Musteraufgabe, Schlüsselaufgabe) VALUES(?,?,?,?,?,?,?)',
                (name, firstname, token, id,personality,pattern,key))
    db.commit()
```

Funktion zum hinzufügen des Nutzers

Die Übergabeparameter (db, cur) sind lediglich da um die zuvor begonnen Verbindung zur Datenbank aufrecht zu erhalten. Dann werden zu beginn die Daten des Nach- und Vornamens sowie die Informationen zu den Tests gespeichert. Falls die Checkboxen im Frontend nicht betätigt wurden soll der Wert auf 0 gesetzt werden um zu signalisieren, dass dieser Test nicht von den Nutzern gemacht werden muss. Anschließend werden individuelle Tokens und IDs erstellt, falls es einen erstellten Token oder eine ID bereits gibt wird solange ein neuer Token und eine neue ID erstellt bis es keine Übereinstimmung gibt. Sobald alle dieser Informationen vorhanden sind, werden diese in die Datenbank eingetragen. Auf die Funktion der Erstellung der Tokens und IDs gehe ich am Ende noch ein.

```python
        #Delete User
        if pressed_button == "deleteButton" and request.method == "POST":
            delete_User(db,cur)

        values = cur.execute('SELECT Nachname, Vorname, ID FROM User')
        data = values.fetchall()
        nachnamen = [row[0] for row in data]
        vornamen = [row[1] for row in data]
        ID = [row[2] for row in data]
        datas = {
        'nachnamen': nachnamen,
        'vornamen': vornamen,
        'ID': ID
        }
        close_connection()
        return render_template("ManageUsers.html", data = datas)
    return redirect(url_for("LogIn"))
```

Backend Code für das Löschen der Nutzer

Hier wird ebenfalls wieder überprüft welcher Knopf gedrückt wurde. In diesem Fall der „deleteButton“ Wenn dies der Fall ist, wird die delete_User(db,cur) Funktion aufgerufen. In „values“ werden alle Daten der „USER“ Tabelle gespeichert. nachnamen, vornamen, ID werden dann alle Daten einzeln als Arrays gespeichert. Mithilfe der „for in“ Funktion werden alle Ergebnisse aus der SQL-Abfrage durchitteriert. Durch datas werden die Daten dann zurück an das Frontend geschickt. Diese werden in dem <select> Tag verwendet. „close_connection“ schließt die Verbindung zur Datenbank. Das erste return Statement lädt die ManageUsers Seite wenn die Login Daten übereinstimmen. Falls dies nicht der Fall ist, ist die erste if-Abfrage „loggedIn“ false und man landet auf der Login Seite.

```
def delete_User(db, cur):
    selected_user = request.form.get('user_id')
    cur.execute('DELETE FROM User WHERE ID = ?', (selected_user,))
    cur.execute('DELETE FROM Persoenlichkeit WHERE ID = ?', (selected_user,))
    cur.execute('DELETE FROM Musteraufgabe WHERE ID = ?', (selected_user,))
    cur.execute('DELETE FROM Schlüsselaufgabe WHERE ID = ?', (selected_user,))
    db.commit()
```

Funktion zum Löschen eines Nutzers

Die Übergabeparameter dienen hier ebenfalls zur Verbindung der Datenbank. Die ID des Nutzers der bei dem <select> Tag ausgewählt wurde, wird nun vom Backend empfangen. Daraufhin werden alle Einträge die es in der Datenbank gibt gelöscht.

```
def getDB():
    db = getattr(g, '_database', None)
    if db is None:
        db = g._database = sqlite3.connect(DATABASE, timeout=10)
    return db

def close_connection():
    db = getattr(g, '_database', None)
    if db is not None:
        db.close()
```

Funktionen zum verbinden und beenden der Verbindung mit der Datenbank

```
def generate_token_ID():
    char = string.ascii_letters + string.digits
    random_char = ''.join(random.choice(char) for i in range (6))
    return random_char
```

Funktion zum generieren von Tokens und IDs

Hier wird eine vordefinierte Funktion genutzt damit der „char“ aus Buchstaben und Zahlen bestehen kann. Daraufhin wird ein zufälliger sechsstelliger Code erstellt.

# 1.4 Entscheidungen

Um eine zukünftige und gewisse Modularität zu gewährleisten wurde entschieden auswählen zu können welche Tests welcher Nutzer bearbeiten muss. Falls diverse Arbeitsgruppen diesen Asessment Test durchlaufen müssen, kann man individuell die Tests auswählen. Wir haben uns gegen eine manuelle Eintragung und für diese Funktion entschieden um dem Ausbilder deutlich Zeit zu ersparen. Des weiteren ist so sichergestellt, dass die Tokens und IDs für jeden Nutzer individuell sind und es keine Dopplungen gibt, dies wäre schwerwiegende wenn man nicht jeden Nutzer individuell und eindeutig zuordnen kann.