
Protokoll

CherryPy

Software Entwicklung
5BHIT 2017/18

Marvin Ertl & Lukas Zuba

Note:
Betreuer:

Version 1.0
Begonnen am 22. Dezember 2017
Beendet am 11. Januar 2018

Inhaltsverzeichnis

1	Aufgabenstellung	1
2	Ergebnis	2
2.1	Allgemein	2
2.2	CherryPy und andere angewendeten Technologien	3
2.3	Vorgehensweise	3
2.3.1	CherryPy	3
2.4	Aufgetretene Probleme	4
2.4.1	Tabelle nicht gefunden	4
2.4.2	Datensatz nicht richtig abgespeichert	5
2.5	Ergebnis	5

1 Aufgabenstellung

Suche dir mit einem/einer Partner/in ein Python Webframework aus und präsentiere deine Lösung!
Grundanforderungen (70%):

- Installiere und konfiguriere eines der präsentierten Frameworks
- Überlege dir einen sinnvollen Anwendungsfall für das Framework und erstelle eine passende Datenbank dazu
- Erstelle eine simple Seite, welche Datensätze aus einer Datenbank anzeigt
- Protokolliere deine Vorgehensweise, aufgetretene Probleme etc.

Erweiterungen (30%):

- Verwende ein ansprechendes Design
- Die Seite erlaubt zusätzlich:
 - Das Bearbeiten von Datensätzen
 - Das Löschen von Datensätzen
 - Das Erstellen von neuen Datensätzen

Abgabe: Protokoll inkl. Arbeitsaufwand, Screenshots und Beschreibungen

2 Ergebnis

Folgende Punkte wurden bearbeitet:

- Allgemein
- CherryPy und andere angewendeten Technologien
- Vorgehensweise
- Aufgetretene Probleme
- Ergebnis

2.1 Allgemein

Wir haben uns für CherryPy entschieden und für einen Anwendungsfall, der schnell zu implementieren und eine niedrige Komplexität aufweist. Das ist laut CherryPy einer der häufigsten Anwendungsfälle für CherryPy. Ein weiterer wichtiger Vorteil ist, dass CherryPy auf OOP ausgelegt ist und lässt sich dadurch zusätzlich leichter schreiben und verstehen.

Wir haben eine Webseite mit CRUD-Funktionen erstellt mit denen man Benutzer verwalten kann. Dazu haben wir in der Datenbank eine Tabelle angelegt die wie folgt aussieht

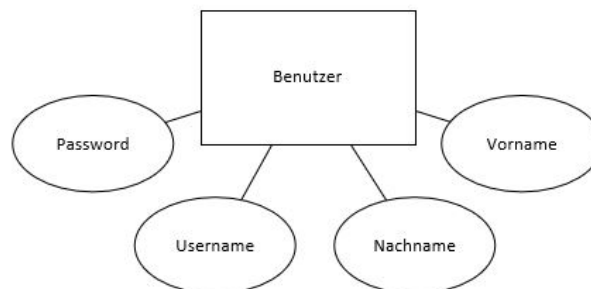


Abbildung 1: Aufbau von Benutzer

Das Einsteigen in CherryPy ist mit Abstand einer der größten Vorteil, den mit wenigen Zeilen lässt sich eine Webseite erstellen und bereitstellen.

```
1 import cherrypy
3 class HelloWorld(object):
    @cherrypy.expose
5     def index(self):
        return "Hello World!"
7
cherrypy.quickstart(HelloWorld())
```

Mit diesen Zeilen ist eine Webseite erstellt und funktionsfähig.

Insgesamt beläuft sich der Arbeitsaufwand auf ungefähr **10 Stunden**.

2.2 CherryPy und andere angewendeten Technologien

CherryPy ist ein Non-Full-Stack-Framework und bietet mit einer sehr einfachen Verfahren einen Webserver an, der mit wenigen Befehlen gestartet und mit Inhalt gefüllt werden kann.

Weiters haben wir für unsere Persistierung die eingebaute Schnittstelle von Python zu SQLite verwendet. Dieses Schnittstelle ermöglicht es mit wenigen Befehlen eine simple Datenbank zu betreiben. Allerdings hatten wir auch einigen Probleme mit der Datenbank auf die wir später noch genauer eingehen.

Um die Webseite mit der CherryPy-Rückgaben zu verbinden, haben wir jQuery verwendet.

Zusätzlich haben wir, um der Webseite ein sprechenden Design zu geben, CSS und Bootstrap verwendet.

2.3 Vorgehensweise

Die Übung ist in 2 Bereiche aufzuteilen, den Server Teil, dieser kümmert sich um das Speichern und Lesen aus der Datenbank, diese Aufgabe wurde mittels den Python WebFramework CherryPy umgesetzt. Der Client Teil kümmert sich hingegen um die Darstellung sowie um die Eingaben, dies wurde mittels HTML, CSS und JavaScript umgesetzt.

2.3.1 CherryPy

Als erstes wurde eine Klasse CRUD erstellt, diese wird aufgerufen sobald man auf localhost mit der entsprechenden Port Nummer zugreift. Die Klasse öffnet die index.html Datei und sendet den Inhalt an den Browser zurück.

```
1 class CRUD(object):  
2     @cherrypy.expose  
3     def index(self):  
4         return open('index.html')
```

Listing 1: Klasse zur Darstellung der Index.html

Nun wurde eine Klasse erstellt, welche auf die HTML Befehle reagiert, dies sind GET, POST, PUT, In unserem Fall haben wir nur POST verwendet mit 2 Parameter. Der erste Parameter param gibt an welche Aktion wir durchführen wollen, der zweite Parameter input gibt liefert zusätzliche Informationen, wie zum Beispiel eingeben Werte.

```
1 @cherrypy.expose  
2 class CRUDWebService(object):  
3  
4     @cherrypy.tools.accept(media='text/plain')  
5     def POST(self, param, input):
```

Listing 2: Klasse zur verwaltung der HTML Befehle

Wird ein `read` als Parameter gesendet, erwartet sich der Client alle Benutzer. Aus diesem Grund werden alle Benutzer aus der Datenbank ausgelesen und eine HTML Tabelle erstellt. Diese wird dann als HTML Response zum Client zurückgesendet und dargestellt.

```

1  if param == "read":
    with sqlite3.connect(DB_STRING) as c:
2     r = c.execute("SELECT * FROM benutzer")
    response = "<table width='100%' class='table table-striped table-bordered' \
5     \"cellspacing='0'><tr><td>Nr</td><td>Vorname</td><td>Nachname</td>\" \
    \"<td>Username</td></tr>\"
7     while True:
        row = r.fetchone()
9     if row is None:
        break
11    response += "<tr><td>\" + str(row[0]) + \"</td><td>\" + row[1] + \"</td><td>\" + row[2] + \"</td><td>\" \
    + row[3] + \"</td></tr>\"
13    response += "</table>\"
    return response

```

Listing 3: Auslesen aller Benutzer aus der Datenbank

Auf den Parameter `update` reagiert das Programm sehr ähnlich, einziger Unterschied ist, dass die Benutzer in einer Tabelle zurückgeliefert werden, die bei jedem Benutzer einen Button zum Bearbeiten hat. Außerdem müsste hier ein kurzes `sleep` eingebaut werden, da `update` gleich nach einer Datenbank ändern die neuen Benutzer ausliest, dadurch konnte es passieren das beim Auslesen noch nicht die neuen Werte dabei waren.

```

1  if param == "update":
2     sleep(0.05)
    with sqlite3.connect(DB_STRING) as c:
4     r = c.execute("SELECT * FROM benutzer")
    response = "<table width='100%' class='table table-striped table-bordered' \
6     \"cellspacing='0'><tr><td>Nr</td><td>Vorname</td><td>Nachname</td>\" \
    \"<td>Username</td><td>Aendern</td></tr>\"
8     while True:
        row = r.fetchone()
10    if row is None:
        break
12    response += "<tr><td>\" + str(row[0]) + \"</td><td>\" + row[1] + \"</td><td>\" + row[2] + \"</td><td>\" \
    + row[3] + \"</td><td> <button onClick='updateBenutzer(\" + str(row[0]) + \")'>Aendern\" \
14    \"</button></td></tr>\"
    response += '</table>'
16    return response

```

Listing 4: Auslesen aller Benutzer aus der Datenbank und bearbeitbar zurückliefern

2.4 Aufgetretene Probleme

2.4.1 Tabelle nicht gefunden

Die häufigste Fehlermeldung dieser Art war: *no such table : benutzer*

Diesen Fehler konnten wir auf keine genaue Stelle eingrenzen, da dieser Fehler zufällig beim Starten des Server hin und wieder aufgetreten ist.

Gelöst haben wir diesen Fehler dann durch das Sicherstellen, dass die Tabelle noch existiert und wenn nicht die Tabelle neu erstellt und mit Standarddatensätzen wieder befüllt wird.

2.4.2 Datensatz nicht richtig abgespeichert

Ein Fehler, der leicht zu lösen war, war: *no such attribute : zuba*

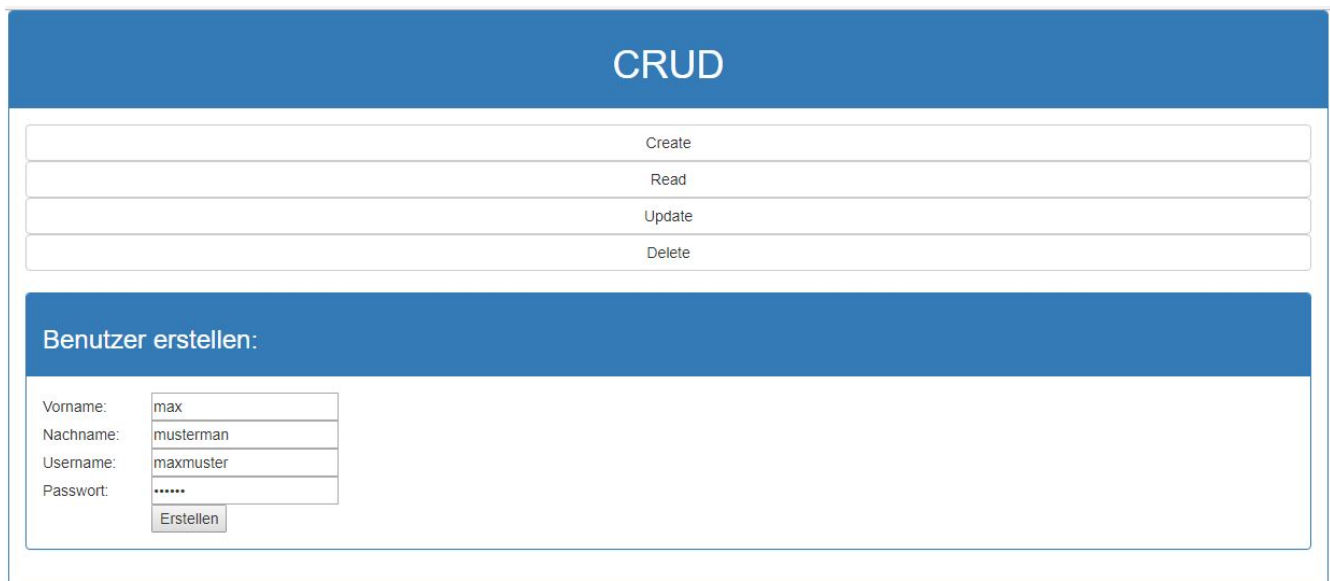
Hierbei hat es sich um eine INSERT Anweisung gehandelt, die einen syntaktisch falschen Aufbau hatte, den wir allerdings erst sehr spät bemerkt haben.

Gelöst wurde dies, indem wir die Zusammensetzung der Anweisung neu geschrieben haben und explizit die Reihenfolge der Attribute angegeben haben.

2.5 Ergebnis

Das Ergebnis der Übung ist eine Webseite, die die CRUD Funktionen bereitstellt, um Benutzer zu verwalten.

CREATE:



CRUD	
	Create
	Read
	Update
	Delete

Benutzer erstellen:

Vorname:	<input type="text" value="max"/>
Nachname:	<input type="text" value="musterman"/>
Username:	<input type="text" value="maxmuster"/>
Passwort:	<input type="password" value="*****"/>
<input type="button" value="Erstellen"/>	

Abbildung 2: CREATE Funktion

READ:

CRUD

Create

Read

Update

Delete

Benutzer auslesen:

Nr	Vorname	Nachname	Username
1	Marvin	Ertl	mertl
2	Lukas	Zuba	lzuba
3	max	musterman	maxmuster

Abbildung 3: READ Funktion

UPDATE:

CRUD

Create

Read

Update

Delete

Benutzer updaten:

Nummer:

3

Vorname:

max

Nachname:

mustermann

Username:

maxmuster

Passwort:

Speichern

Nr	Vorname	Nachname	Username	Ändern
1	Marvin	Ertl	mertl	Ändern
2	Lukas	Zuba	lzuba	Ändern
3	max	musterman	maxmuster	Ändern

Abbildung 4: UPDATE Funktion

DELETE:

CRUD

Create

Read

Update

Delete

Benutzer löschen:

Nr	Vorname	Nachname	Username	Löschen
1	Marvin	Ertl	mertl	<div>Löschen</div>
2	Lukas	Zuba	lzuba	<div>Löschen</div>
3	max	mustermann	maxmuster	<div>Löschen</div>

Abbildung 5: DELETE Funktion

CRUD

Create

Read

Update

Delete

Benutzer löschen:

1	Marvin	Ertl	mertl	<div>Löschen</div>
2	Lukas	Zuba	lzuba	<div>Löschen</div>

Abbildung 6: DELETE nach dem Löschen von Max Mustermann

Literatur

Tabellenverzeichnis

Listings

1	Klasse zur Darstellung der Index.html	3
2	Klasse zur verwaltung der HTML Befehle	3
3	Auslesen aller Benutzer aus der Datenbank	4
4	Auslesen aller Benutzer aus der Datenbank und bearbeitbar zurückliefern	4

Abbildungsverzeichnis

1	Aufbau von Benutzer	2
2	CREATE Funktion	5
3	READ Funktion	6
4	UPDATE Funktion	6
5	DELETE Funktion	7
6	DELETE nach dem Löschen von Max Mustermann	7