
A07: Python Webframeworks

CherryPy - CRUD

SEW
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	Umsetzung	3
2.3.1	CherryPy	3
2.3.2	HTML	6
2.4	Aufgetretene Probleme	7
2.4.1	Tabelle nicht gefunden	7
2.4.2	Datensatz nicht richtig abgespeichert	7
2.4.3	JS Funktionen nicht aufrufbar	7
2.5	Ergebnis	8

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
- Umsetzung
- 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:

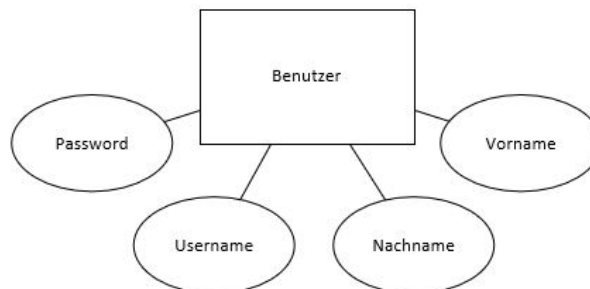


Abbildung 1: Aufbau von Benutzer

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
    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 Umsetzung

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:
```

```

3      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></tr>\" \
      \"<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":
    sleep(0.05)
    with sqlite3.connect(DB_STRING) as c:
        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></tr>\" \
        \"<td>Username</td><td>Aendern</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><td> <button onClick='updateBenutzer(\" + str(row[0]) + \")'>Aendern
            </button></td></tr>\"
13     response += '</table>'
    return response

```

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

Wenn der Benutzer eine Person bearbeiten will, müssen die aktuellen Informationen von dieser Person abgerufen werden, hierfür gibt es den Parameter `getBenutzer`. Dieser hat zusätzlich als input die Nummer, welche der Primary Key der Entität ist. Die zurückgelieferten Informationen werden, dann in Input Felder angezeigt um diese zu bearbeiten.

```

1  if param == "getBenutzer":
    with sqlite3.connect(DB_STRING) as c:
        r = c.execute("SELECT * FROM benutzer WHERE nr=" + input)
4        while True:
            row = r.fetchone()
6            if row is None:
                break
8            response = str(row[0]) + "#" + row[1] + "#" + row[2] + "#" + row[3] + "#" + row[4];
    return response

```

Listing 5: Auslesen eines Benutzer aus der Datenbank

Wenn der Benutzer die Informationen in den Input Feldern geändert hat und auf Speichern klickt, werden die neuen Informationen über den Parameter `updateBenutzer` und die Informationen als input an den Server gesendet. Anschließend wird der Benutzer in der Datenbank aktualisiert.

```

1  if param == "updateBenutzer":
    with sqlite3.connect(DB_STRING) as c:
3      liste = input.split('#')

```

```

5      c.execute("UPDATE benutzer SET vorname='" + liste[1] + "',nachname='"
        + liste[2] + "',username='" + liste[3] + "',password='" + liste[4] + "' WHERE nr=" + liste
        [0])
    return "Benutzer erfolgreich geaendert."

```

Listing 6: Updaten der Benutzerinformationen

Wird ein neuer Benutzer erstellt, wird über den Parameter create und den Benutzerinformationen als input an den Server gesendet. Dieser speichert die Informationen in der Datenbank.

```

1  if param == "create":
2      with sqlite3.connect(DB_STRING) as c:
3          liste = input().split('#')
4          c.execute("INSERT INTO benutzer(vorname, nachname, username, password) VALUES (" + liste[0] + ",
                    "
                    + liste[1] + ", " + liste[2] + ", " + liste[3] + ")")
6      return "Benutzer erfolgreich gespeichert."

```

Listing 7: Updaten der Benutzerinformationen

Der delete Branch ist sehr ähnlich zum update Branch, der einzige Unterschied ist, dass anstelle eines Update Button wird ein Löschen Button erstellt.

```

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

```

Listing 8: Auslesen aller Benutzer mit einen Löschen Button

Abschließend können Benutzer aus der Datenbank gelöscht werden, hierfür wird als Parameter deleteBenutzer angegeben und als input die Nummer des Benutzers. Falls ein ungültiger Parameter gesendet wird, wird ein error zurückgeliefert.

```

1  if param == "deleteBenutzer":
2      with sqlite3.connect(DB_STRING) as c:
3          c.execute("DELETE FROM benutzer WHERE nr=" + input)
4          return self.POST("delete", "")
5
6      return "Ungueltige Anfrage"

```

Listing 9: Löschen eines bestimmten Benutzers

Weiters gibt es die Methode setup_database diese wird immer beim Starten des Servers ausgeführt und soll sicherstellen, dass die Datenbank sowie Tabelle vorhanden ist. Außerdem werden in diesem Fall noch 2 Benutzer als Testdaten standardmäßig importiert.

```

1  def setup_database():
2      with sqlite3.connect(DB_STRING) as con:
3          con.execute("DROP TABLE IF EXISTS benutzer")
4          con.execute("CREATE TABLE IF NOT EXISTS benutzer(nr INTEGER PRIMARY KEY AUTOINCREMENT, vorname
                    VARCHAR, "
                    "nachname VARCHAR, username VARCHAR, password VARCHAR)")

```

```

6 con.execute("INSERT INTO benutzer VALUES(null, 'Marvin', 'Ertl', 'mertl', 'password')")
  con.execute("INSERT INTO benutzer VALUES(null, 'Lukas', 'Zuba', 'lzuba', 'password')")

```

Listing 10: Erstellen der Tabelle Benutzer beim Start des Servers

Ebenso gibt es eine Methode namens `cleanup_database`, diese löscht die Tabelle beim Beenden des Servers.

```

1 def cleanup_database():
  with sqlite3.connect(DB_STRING) as con:
3     con.execute("DROP TABLE IF EXISTS benutzer")

```

Listing 11: Löschen der Tabelle benutzer beim Beenden des Servers

Abschließend wird in einer IF-Anweisung die Konfiguration hineingeschrieben um den Server zu starten. Ebenfalls wird hier auch angegeben welche Methoden beim Starten sowie beim Beenden des Servers ausgeführt werden sollen. Außerdem wird die `auto reload` ausgeschaltet, dies führt ansonsten dazu, dass wenn das Python Programm abstürzt der Server weiterhin im Hintergrund den Port belegt. Schlussendlich wird angegeben welche Klasse beim Aufrufen reagiert, sowie welche Klasse auf die HTML Befehle reagiert.

```

1 if __name__ == '__main__':
  conf = {
3     '/': {
        'tools.sessions.on': True,
5         'tools.staticdir.root': os.path.abspath(os.getcwd())
        },
7     '/generator': {
        'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
9         'tools.response_headers.on': True,
        'tools.response_headers.headers': [( 'Content-Type', 'text/plain')],
11    },
        '/static': {
13        'tools.staticdir.on': True,
        'tools.staticdir.dir': './public'
15    }
    }
17 cherrypy.engine.subscribe('start', setup_database)
  cherrypy.engine.subscribe('stop', cleanup_database)
19 cherrypy.config.update({'server.socket_port': 8080})
  cherrypy.config.update({'engine.autoreload_on': False})
21
23 webapp = CRUD()
  webapp.generator = CRUDWebService()
  cherrypy.quickstart(webapp, '/', conf)

```

Listing 12: Konfiguration des Servers

2.3.2 HTML

Bei der `index.html` Datei handelt es sich um eine einfache Webseite welche mittels Bootstrap gestaltet worden ist. Auf den HTML Teil wird hier nicht genauer eingegangen, da es sich bei diesem um ein paar Tabellen sowie input Felder und Button mit ein bisschen CSS Formatierung handelt.

Viel wichtiger ist der JavaScript Teil, dieser liest nämlich die Eingaben aus und sendet diese an den Server.

Nachfolgend ist der JavaScript Code, welcher aufgerufen wird, wenn der Button `showRead` angeklickt wird. Als erstes werden alle anderen Tabs geschlossen und anschließend über die Post

Methode mittels den Parameter read eine Anfrage gesendet. Als Response kommt die Tabelle mit allen Benutzer zurück, diese wird an bei dem Element listBenutzer eingefügt.

```
1 $("#showRead").click(function(e) {  
2     closeTabs();  
3     $.post("/generator", {"param": "read", "input": ""})  
4     .done(function (string) {  
5         $("#read-string").show();  
6         document.getElementById("listBenutzer").innerHTML = string;  
7     })  
8     e.preventDefault();  
9 });
```

Listing 13: Anfrage senden an den Server

Die restlichen Methoden für die anderen Funktionen sind gleich aufgebaut, einige überprüfen die Eingaben davor noch bzw. fassen die Angaben zu einen String zusammen, welcher anschließend als String über die input Variable an den Server gesendet wird.

Ansonsten wurde noch ein bisschen JavaScript verwendet um, Eingabemöglichkeiten zu verstecken bzw. anzuzeigen wenn diese benötigt werden. Dadurch wird die Oberfläche übersichtlicher und einfacher bedienbar.

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.

2.4.3 JS Funktionen nicht aufrufbar

Ein nicht so einfach findbarer Fehler war, *function showuser not found*

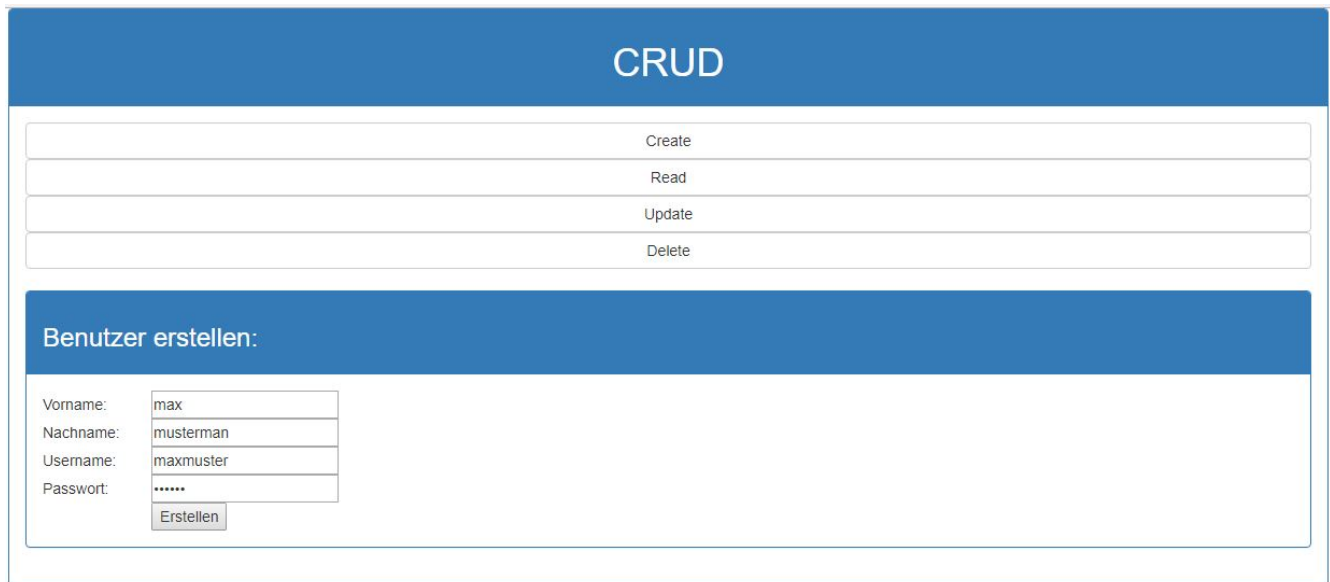
Bei diesem Fehler wurde eine Anfrage an den Server gesendet, durch einen Syntax Fehler bzw. dadurch dass es lange her ist, seit wir mit JavaScript gearbeitet haben, gab es einen Fehler beim Absenden der Anfrage. Da aber die Anfrage gesendet worden ist, haben wir den Fehler erst am

Server gesucht, da dort der Fehler auch aufgetreten ist, dabei war aber die gesendete Anfrage nicht im richtigen Format.

2.5 Ergebnis

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

CREATE:



The screenshot shows a web application interface for a CRUD system. At the top, there is a blue header bar with the text "CRUD" in white. Below the header, there is a white box containing four buttons: "Create", "Read", "Update", and "Delete". Below this box, there is a blue bar with the text "Benutzer erstellen:". Below the blue bar, there is a white form with four input fields: "Vorname:" (containing "max"), "Nachname:" (containing "musterman"), "Username:" (containing "maxmuster"), and "Passwort:" (containing "*****"). Below the input fields, there is a button labeled "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:

Vorname:

Nachname:

Username:

Passwort:

Nr	Vorname	Nachname	Username	Ändern
1	Marvin	Ertl	mertl	<input type="button" value="Ändern"/>
2	Lukas	Zuba	lzuba	<input type="button" value="Ändern"/>
3	max	musterman	maxmuster	<input type="button" value="Ändern"/>

Abbildung 4: UPDATE Funktion

DELETE:

CRUD

Create

Read

Update

Delete

Benutzer löschen:

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

Abbildung 5: DELETE Funktion

CRUD

Create

Read

Update

Delete

Benutzer löschen:

Nr	Vorname	Nachname	Username	Löschen
1	Marvin	Ertl	mertl	<button>Löschen</button>
2	Lukas	Zuba	lzuba	<button>Löschen</button>

Abbildung 6: DELETE nach dem Löschen von Max Mustermann

Listings

1	Klasse zur Darstellung der Index.html	3
2	Klasse zur verwaltung der HTML Befehle	3
3	Auslesen aller Benutzer aus der Datenbank	3
4	Auslesen aller Benutzer aus der Datenbank und bearbeitbar zurückliefern	4
5	Auslesen eines Benutzer aus der Datenbank	4
6	Updaten der Benutzerinformationen	4
7	Updaten der Benutzerinformationen	5
8	Auslesen aller Benutzer mit einen Löschen Button	5
9	Löschen eines bestimmten Benutzers	5
10	Erstellen der Tabelle Benutzer beim Start des Servers	5
11	Löschen der Tabelle benutzer beim Beenden des Servers	6
12	Konfiguration des Servers	6
13	Anfrage senden an den Server	7

Abbildungsverzeichnis

1	Aufbau von Benutzer	2
2	CREATE Funktion	8
3	READ Funktion	9
4	UPDATE Funktion	9
5	DELETE Funktion	10
6	DELETE nach dem Löschen von Max Mustermann	10