# Ziele

Mit Hilfe der 2. Aufgabe sollen Sie die serverseitige Verarbeitung weiter einüben.

# Webanwendung "Studieninformationstag"

## Beschreibung der Anwendung

Mit der Webanwendung "Studieninformationstag" sollen sich Schülerguppen zur Teilnahme an Veranstaltungen des Studieninformationstages anmelden können:

- als Startseite wird die Liste der Veranstaltungen angezeigt:
  - Bezeichnung der Veranstaltung
  - Beginn und Ende
  - Raum
  - maximal möglich Anzahl Anmeldungen
  - aktuelle Anzahl Anmeldungen
- für jede Veranstaltung ist von der Startseite aus die Anmeldung möglich, wenn die maximale Anzahl Anmeldungen noch nicht überschritten wurde
- zur Anmeldung wird ein Formular verwendet:
  - die Veranstaltung, zu der die Anmeldung erfolgt, wird angezeigt
  - die Schülergruppe gibt den Namen der Schule, die Klassenstufe und die Anzahl der Schüler an
  - mit dem Schalter "Anmelden" erfolgt die verbindliche Anmeldung
  - mit dem Schalter "Zurück" wird wieder die Startseite aufgerufen.

Der aktuelle Belegungsstand der einzelnen Veranstaltungen des Studieninformationstages kann durch eine weitere Seite abgefragt werden:

- die Veranstaltungen werden alphabetisch aufgelistet
  - die Angaben zu jeder Veranstaltung werden dargestellt
  - die teilnehmenden Schülergruppen werden alphabetisch sortiert nach Namen der Schule aufgeführt.

Der Aufruf der Startseite erfolgt mit "http://localhost:8080/". Das Anmeldeformular ist sinnvoll nur über die Auswahl in der Startseite erreichbar. Die Abfrage des Belegungsstands erfolgt mit "http://localhost:8080/belegung".

Die Daten zu den Veranstaltungen werden serverseitig in JSON-Dateien gespeichert. Zur Vereinfachung der Aufgabenstellung gibt es keine Pflegefunktion. Die Dateiinhalte können direkt mit einem Code-Editor bearbeitet werden.

## Schritt 1: Verzeichnisse und Dateien erstellen

Erstellen Sie folgende Verzeichnisstruktur:

```
web
   /p2
       /sit
           /app
           /content
           /data
           /doc
           /template
```

Erstellen Sie im Verzeichnis `web/p2/sit` die Datei `server.py` (siehe Anlage 1).

Erstellen Sie im Verzeichnis `web/p2/sit/app` die Dateien

- `__init__.py` (wie in Aufgabe 1)
- `application.py` (Inhalt: siehe Anlage 2)
  - nimmt die Anfragen des Webclient entgegen (*Requests*) und erzeugt die Antworten des Webservers (*Responses*)
  - verwendet dazu die Methoden aus den beiden anderen Modulen
- `database.py` (Inhalt: siehe Anlage 3)
  - implementiert eine einfache Datenhaltung
- `view.py` (Inhalt: siehe Anlage 4)
  - erzeugt das Markup, das ausgeliefert werden soll.

Erstellen Sie im Verzeichnis `web/p2/sit/content`

- die zunächst leere Datei `sit.css`

Erstellen Sie das Verzeichnis `web/p2/sit/template`.

Achten Sie darauf, bei allen Textdateien die Zeichenkodierung UTF-8 (ohne BOM [Byte Order Mark] ]) zu verwenden!

## Schritt 2: Ergänzungen vornehmen

Die vorgegebene Implementierung ist nicht vollständig. Ergänzen Sie folgende Punkte:

- erstellen Sie im Verzeichnis `web/p2/sit/template` die Vorlagen für das Markup der Liste, des Formulars und der Anzeige der Belegung
  - orientieren Sie sich am Aufbau der Templates der Aufgabe 1
  - sehen Sie folgende Dateien vor:
    - Startseite: `list0.tpl`, `list1.tpl`, `list2.tpl`
    - Anmeldeformular: `form0.tpl`, `form1.tpl`, `form2.tpl`
    - Startseite: `belegung0.tpl`, `belegung1.tpl`, `belegung2.tpl`
- implementieren Sie auf der Serverseite die Erzeugung der Seite zur Anzeige der Belegung (siehe Dateien `application.py`, `database.py`, `view.py`)
  - zur Vereinfachung reicht es aus, eine Tabelle zu verwenden, in der die Veranstaltung in der ersten Spalte aufgeführt und ggf. mehrfach - je nach Anzahl der Schülergruppen - wiederholt wird
- gestalten Sie die Seiten mit Hilfe von CSS: tragen Sie die CSS-Stilregeln dazu in die Datei `sit.css` ein
  - recherchieren Sie bei http://www.w3schools.com, welche einfachen Möglichkeiten bestehen.

## Schritt 3: Dokumentation erstellen

Erstellen Sie eine Dokumentation. Legen Sie dazu im Verzeichnis `web/p2/sit/doc` die Datei `sit.md` an. Schreiben Sie die Dokumentation als `Markdown`-Dokument und sehen Sie folgende Gliederung vor:

- Aufbau der Webanwendung
  - Aufbau des Webclient
    - Seite "Liste Veranstaltungen"
    - Formular "Anmeldung"
    - Seite "Belegungsstand"
  - Aufbau des Webservers
- Durchgeführte Ergänzungen
- Beschreibung des HTTP-Datenverkehrs
  - beim Start der Anwendung
  - beim Speichern von Formulardaten
  - verwenden Sie Screenshots der "Netzwerkanalyse" des Webbrowser und geben Sie an

- welche Anfragen an den den Webserver geschickt werden (HTTP-Methode, URI, Inhalt der Anfrage)
- welche Antworten der Webserver liefert (Inhalt beschreiben).

Geben Sie einleitend Ihre Gruppenzugehörigkeit, den Aufbau Ihres Teams und das Gültigkeitsdatum der Dokumentation an.

Die Dokumentation wird als utf-8 kodierter Text mit der einfachen Auszeichnungssprache "markdown" erstellt. Mit Hilfe des Werkzeugs "pandoc" (siehe http://pandoc.org) kann eine Umsetzung in eine HTML-Datei erfolgen:

```
pandoc -f markdown -t html5 -s <IhreDatei> -o <IhreHTML5Datei>
```

Die in "pandoc" verfügbaren Erweiterungen der Auszeichnungssprache "markdown" können genutzt werden.

## Testat

Sie erhalten das Testat, wenn Sie die geforderten Ergänzungen demonstrieren und erläutern können und die Dokumentation vorlegen.

# Anlagen

## 1: Datei server.py

```python
#coding: utf-8
import os
import cherrypy
from app import application

#--------------------------------------
def main():
#--------------------------------------
   # Get current directory
   try:
      current_dir = os.path.dirname(os.path.abspath(__file__))
   except:
      current_dir = os.path.dirname(os.path.abspath(sys.executable))
   # disable autoreload and timeout_monitor
   cherrypy.engine.autoreload.unsubscribe()
   cherrypy.engine.timeout_monitor.unsubscribe()
   # Static content config
   static_config = {
      '/': {
            'tools.staticdir.root': current_dir,
            'tools.staticdir.on': True,
            'tools.staticdir.dir': './content'
      }
   }
   # Mount static content handler
   root_o = cherrypy.tree.mount(application.Application_cl(), '/', static_config)
   # suppress traceback-info
   cherrypy.config.update({'request.show_tracebacks': False})
   # Start server
   cherrypy.engine.start()
   cherrypy.engine.block()

#--------------------------------------
if __name__ == '__main__':
#--------------------------------------
   main()
# EOF
```

## 2: Datei application.py

```python
# coding: utf-8

import cherrypy

from .database import Database_cl
from .view import View_cl

#-------------------------------------------------------
```

```
9  class Application_cl(object):
10 #-----------------------------------------------------------
11
12    #-----------------------------------------------------------
13    def __init__(self):
14    #-----------------------------------------------------------
15       # spezielle Initialisierung können hier eingetragen werden
16       self.db_o = Database_cl()
17       self.view_o = View_cl()
18
19    @cherrypy.expose
20    #-----------------------------------------------------------
21    def index(self):
22    #-----------------------------------------------------------
23       return self.createList_p()
24
25    @cherrypy.expose
26    #-----------------------------------------------------------
27    def reservations(self):
28    #-----------------------------------------------------------
29       return self.createReservationList_p()
30
31    @cherrypy.expose
32    #-----------------------------------------------------------
33    def register(self, id):
34    #-----------------------------------------------------------
35       return self.createForm_p(id)
36
37    @cherrypy.expose
38    #-----------------------------------------------------------
39    def save(self, **data_opl):
40    #-----------------------------------------------------------
41       # Sichern der Daten: aufgrund der Formularbearbeitung muss
42       # eine vollständige HTML-Seite zurückgeliefert werden!
43
44       # data_opl: Dictionary mit den gelieferten key-value-Paaren
45
46       id_s = data_opl["id_s"]
47       self.db_o.addReservation_px(id_s, data_opl)
48
49       return self.createForm_p(id_s)
50
51    @cherrypy.expose
52    #-----------------------------------------------------------
53    def default(self, *arguments, **kwargs):
54    #-----------------------------------------------------------
55       msg_s = "unbekannte Anforderung: " + \
56               str(arguments) + \
57               ' ' + \
58               str(kwargs)
59       raise cherrypy.HTTPError(404, msg_s)
60    default.exposed= True
61
62    #-----------------------------------------------------------
63    def createList_p(self):
64    #-----------------------------------------------------------
65       data_o = self.db_o.read_px()
```

```
66          # mit diesen Daten Markup erzeugen
67          return self.view_o.createList_px(data_o)
68
69      #-------------------------------------------------------
70      def createReservationList_p(self):
71      #-------------------------------------------------------
72          data_o = self.db_o.readReservations_px()
73          # mit diesen Daten Markup erzeugen
74          return self.view_o.createReservationList_px(data_o)
75
76      #-------------------------------------------------------
77      def createForm_p(self, id_spl):
78      #-------------------------------------------------------
79          data_o = self.db_o.read_px(id_spl)
80          # mit diesen Daten Markup erzeugen
81          return self.view_o.createForm_px(id_spl, data_o)
82
83  # EOF
```

## 3: Datei database.py

```
1   # coding: utf-8
2
3   import os
4   import os.path
5   import codecs
6
7   import json
8
9   #-------------------------------------------------------
10  class Database_cl(object):
11  #-------------------------------------------------------
12
13      # - die Daten werden als eine JSON-Datei abgelegt
14
15      #-------------------------------------------------------
16      def __init__(self):
17      #-------------------------------------------------------
18          self.data_o = None
19          self.readData_p()
20
21      #-------------------------------------------------------
22      def read_px(self, id_spl = None):
23      #-------------------------------------------------------
24          # (Veranstaltungen)
25          # hier zur Vereinfachung:
26          # Aufruf ohne id: alle Einträge liefern
27          data_o = None
28          if id_spl == None:
29              data_o = self.data_o
30          else:
31              if id_spl in self.data_o:
32                  data_o = self.data_o[id_spl]
33
34          return data_o
```

```
35
36    #-----------------------------------------------------
37    def readReservations_px(self, id_spl):
38    #-----------------------------------------------------
39       # (Veranstaltungen + Anmeldungen)
40
41       # --Ihre Ergänzung--
42
43    #-----------------------------------------------------
44    def addReservation_px(self, id_spl, data_opl):
45    #-----------------------------------------------------
46
47       # --Ihre Ergänzung--
48
49    #-----------------------------------------------------
50    def readData_p(self):
51    #-----------------------------------------------------
52       # Datei muss existieren, daher Ausnahme erzeugen, wenn sie nicht geöffnet werden kann
53       fp_o = codecs.open(os.path.join('data', 'sit.json'), 'r', 'utf-8')
54       with fp_o:
55          self.data_o = json.load(fp_o)
56
57       return
58
59    #-----------------------------------------------------
60    def saveData_p(self):
61    #-----------------------------------------------------
62       with codecs.open(os.path.join('data', 'sit.json'), 'w', 'utf-8') as fp_o:
63          json.dump(self.data_o, fp_o)
64
65 # EOF
```

## 4: Datei view.py

```
1  # coding: utf-8
2
3  # sehr einfache Erzeugung des Markups für vollständige Seiten
4  # jeweils 3 Abschnitte:
5  # - begin
6  # - content
7  # - end
8
9  # bei der Liste wird der content-Abschnitt wiederholt
10 # beim Formular nicht
11
12 import codecs
13 import os.path
14 import string
15
16 #-----------------------------------------------------
17 class View_cl(object):
18 #-----------------------------------------------------
19
20    #-----------------------------------------------------
21    def __init__(self):
```

```
22    #-----------------------------------------------------
23       pass
24
25    #-----------------------------------------------------
26    def createList_px(self, data_opl):
27    #-----------------------------------------------------
28       markup_s = ''
29       markup_s += self.readFile_p('list0.tpl')
30
31       markupV_s = self.readFile_p('list1.tpl')
32       lineT_o = string.Template(markupV_s)
33       # mehrfach nutzen, um die einzelnen Zeilen der Tabelle zu erzeugen
34       for id_s in data_opl:
35          data_o = data_opl[id_s]
36          markup_s += lineT_o.safe_substitute (bezeichnung_s=data_o['bezeichnung_s']
37          ,   beginn_s=data_o['beginn_s']
38          ,   ende_s=data_o['ende_s']
39          ,   raum_s=data_o['raum_s']
40          ,   max_i=data_o['max_i']
41          ,   act_i=data_o['act_i']
42          ,   id_s=id_s
43          )
44
45       markup_s += self.readFile_p('list2.tpl')
46
47       return markup_s
48
49
50    #-----------------------------------------------------
51    def createReservationList_px(self, data_opl):
52    #-----------------------------------------------------
53       markup_s = ''
54       markup_s += self.readFile_p('belegung0.tpl')
55
56       markupV_s = self.readFile_p('belegung1.tpl')
57       lineT_o = string.Template(markupV_s)
58       # mehrfach nutzen, um die einzelnen Zeilen der Tabelle zu erzeugen
59
60       # --Ihre Ergänzung--
61
62       markup_s += self.readFile_p('belegung2.tpl')
63
64       return markup_s
65
66    #-----------------------------------------------------
67    def createForm_px(self, id_spl, data_opl):
68    #-----------------------------------------------------
69
70       # da hier nur ein leeres Formular benötigt wird, werden die als Parameter
71       # gelieferten Daten ignoriert
72
73       markup_s = ''
74       markup_s += self.readFile_p('form0.tpl')
75
76       markupV_s = self.readFile_p('form1.tpl')
77       lineT_o = string.Template(markupV_s)
78       markup_s += lineT_o.safe_substitute (
```

```
79
80          # -- Ihre Ergänzung
81
82       )
83
84       markup_s += self.readFile_p('form2.tpl')
85
86       return markup_s
87
88    #-------------------------------------------------------
89    def readFile_p(self, fileName_spl):
90    #-------------------------------------------------------
91       content_s = ''
92       with codecs.open(os.path.join('template', fileName_spl), 'r', 'utf-8') as fp_o:
93          content_s = fp_o.read()
94
95       return content_s
96 # EOF
```

## 5: Datei sit.json (Beispiel)

```
1  {
2     "1": {
3        "max_i": 5,
4        "ende_s": "12.00",
5        "bezeichnung_s": "Raspi basteln",
6        "beginn_s": "11.00",
7        "act_i": 2,
8        "raum_s": "B320",
9        "anmeldungen": [
10          {
11             "anzahl_s": 20,
12             "schule_s": "schule1",
13             "stufe_s": 13
14          },
15          {
16             "anzahl_s": 15,
17             "schule_s": "schule2",
18             "stufe_s": 12
19          }
20       ]
21    }
22    "2": {
23       "max_i": 10,
24       "ende_s": "11.00",
25       "bezeichnung_s": "Routenplaner",
26       "beginn_s": "10.00",
27       "act_i": 0,
28       "raum_s": "Audimax",
29       "anmeldungen": [
30       ]
31    }
32 }
```