

Ziele

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

Webanwendung "Studieninformationstag"

Beschreibung der Anwendung

Mit der Webanwendung "Studieninformationstag" sollen sich Schülergruppen 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; von dieser Seite aus soll es auch möglich sein, Anmeldungen zu bearbeiten (falls bei der Anmeldung falsche Angaben gemacht wurden) oder sogar zu entfernen:

- die Veranstaltungen werden alphabetisch aufgelistet
 - die Angaben zu jeder Veranstaltung werden dargestellt
 - die teilnehmenden Schülergruppen werden alphabetisch sortiert nach Namen der Schule aufgeführt
- bei jeder teilnehmenden Schülergruppe besteht die Möglichkeit zum Bearbeiten und zum Löschen der Anmeldung
 - das Löschen muss bestätigt werden
 - das Bearbeiten erfolgt wiederum mit Hilfe eines Formulars:
 - die Veranstaltung, zu der die Anmeldung bearbeitet wird, wird angezeigt
 - es können bearbeitet werden: Name der Schule, Klassenstufe und Anzahl der Schüler
 - mit dem Schalter "Speichern" werden die Änderungen gespeichert
 - mit dem Schalter "Zurück" wird wieder die Belegungsübersicht (mit den geänderten Daten) aufgerufen.

Die beiden Formulare sollen durch eine passende Überschrift eindeutig unterschieden werden können.

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 für die Veranstaltungsdaten. 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`
- die zunächst leere Datei `sit.js`

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`
 - Belegungsübersicht: `belegung0.tpl`, `belegung1.tpl`, `belegung2.tpl`
 - Formular zur Bearbeitung der Anmeldungen: `formEdit0.tpl`, `formEdit1.tpl`, `formEdit2.tpl`
- implementieren Sie serverseitig 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
- implementieren Sie serverseitig die Erzeugung des Formulars zur Bearbeitung der Anmeldungen, die Speicherung der Änderungen und die Löschung von Anmeldungen (siehe Dateien `application.py`, `database.py`, `view.py`)
 - implementieren Sie die clientseitige Bestätigung der Löschung

- 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
 - gestalten Sie insbesondere die Tabellen in einer Weise, die die Übersichtlichkeit und Bedienbarkeit verbessert.

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"
 - Formular "Änderung Anmeldung"
 - Aufbau des Webservers
- Durchgeführte Ergänzungen
- Gestaltung von Tabellen mit CSS.

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

```
1 #coding: utf-8
2 import os
3 import cherrypy
4 from app import application
5
6 #-----
7 def main():
8 #-----
9     # Get current directory
10    try:
11        current_dir = os.path.dirname(os.path.abspath(__file__))
12    except:
13        current_dir = os.path.dirname(os.path.abspath(sys.executable))
14    # disable autoreload and timeout_monitor
15    cherrypy.engine.autoreload.unsubscribe()
16    cherrypy.engine.timeout_monitor.unsubscribe()
17    # Static content config
18    static_config = {
19        '/': {
20            'tools.staticdir.root': current_dir,
21            'tools.staticdir.on': True,
22            'tools.staticdir.dir': './content'
23        }
24    }
25    # Mount static content handler
26    root_o = cherrypy.tree.mount(application.Application_cl(), '/', static_config)
27    # suppress traceback-info
28    cherrypy.config.update({'request.show_tracebacks': False})
29    # Start server
30    cherrypy.engine.start()
31    cherrypy.engine.block()
32
33 #-----
34 if __name__ == '__main__':
35 #-----
36     main()
37 # EOF
```

2: Datei application.py

```
1 # coding: utf-8
2
3 import cherrypy
4
5 from .database import Database_cl
6 from .view import View_cl
7
8 #-----
```

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

```
66 #-----
67     data_o = self.db_o.read_px()
68     # mit diesen Daten Markup erzeugen
69     return self.view_o.createList_px(data_o)
70
71 #-----
72 def createReservationList_p(self):
73 #-----
74     data_o = self.db_o.readReservations_px()
75     # mit diesen Daten Markup erzeugen
76     return self.view_o.createReservationList_px(data_o)
77
78 #-----
79 def createForm_p(self, id_spl):
80 #-----
81     data_o = self.db_o.read_px(id_spl)
82     # mit diesen Daten Markup erzeugen
83     return self.view_o.createForm_px(id_spl, data_o)
84
85 # 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     # Ergänzen Sie weitere Methoden, wenn Sie dies für erforderlich halten!
16
17     #-----
18     def __init__(self):
19     #-----
20         self.data_o = None
21         self.readData_p()
22
23     #-----
24     def read_px(self, id_spl = None):
25     #-----
26         # (Veranstaltungen)
27         # hier zur Vereinfachung:
28         # Aufruf ohne id: alle Einträge liefern
29         data_o = None
30         if id_spl == None:
31             data_o = self.data_o
32         else:
```

```
33         if id_spl in self.data_o:
34             data_o = self.data_o[id_spl]
35
36         return data_o
37
38         #-----
39     def readReservations_px(self, id_spl):
40         #-----
41         # (Veranstaltungen + Anmeldungen)
42
43         # --Ihre Ergänzung--
44
45         #-----
46     def addReservation_px(self, id_spl, data_opl):
47         #-----
48
49         # --Ihre Ergänzung--
50
51         #-----
52     def readData_p(self):
53         #-----
54         # Datei muss existieren, daher Ausnahme erzeugen, wenn sie nicht geöffnet werden kann
55         fp_o = codecs.open(os.path.join('data', 'sit.json'), 'r', 'utf-8')
56         with fp_o:
57             self.data_o = json.load(fp_o)
58
59         return
60
61         #-----
62     def saveData_p(self):
63         #-----
64         with codecs.open(os.path.join('data', 'sit.json'), 'w', 'utf-8') as fp_o:
65             json.dump(self.data_o, fp_o)
66
67     # 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 # Ergänzen Sie weitere Methoden, wenn Sie dies für erforderlich halten!
21
22 #-----
23 def __init__(self):
24 #-----
25     pass
26
27 #-----
28 def createList_px(self, data_opl):
29 #-----
30     markup_s = ''
31     markup_s += self.readFile_p('list0.tpl')
32
33     markupV_s = self.readFile_p('list1.tpl')
34     lineT_o = string.Template(markupV_s)
35     # mehrfach nutzen, um die einzelnen Zeilen der Tabelle zu erzeugen
36     for id_s in data_opl:
37         data_o = data_opl[id_s]
38         markup_s += lineT_o.safe_substitute (bezeichnung_s=data_o['bezeichnung_s']
39         ,   beginn_s=data_o['beginn_s']
40         ,   ende_s=data_o['ende_s']
41         ,   raum_s=data_o['raum_s']
42         ,   max_i=data_o['max_i']
43         ,   act_i=data_o['act_i']
44         ,   id_s=id_s
45         )
46
47     markup_s += self.readFile_p('list2.tpl')
48
49     return markup_s
50
51
52 #-----
53 def createReservationList_px(self, data_opl):
54 #-----
55     markup_s = ''
56     markup_s += self.readFile_p('belegung0.tpl')
57
58     markupV_s = self.readFile_p('belegung1.tpl')
59     lineT_o = string.Template(markupV_s)
60     # mehrfach nutzen, um die einzelnen Zeilen der Tabelle zu erzeugen
61
62     # --Ihre Ergänzung--
63
64     markup_s += self.readFile_p('belegung2.tpl')
65
66     return markup_s
67
68 #-----
69 def createForm_px(self, id_spl, data_opl):
70 #-----
71
72     # da hier nur ein leeres Formular benötigt wird, werden die als Parameter
73     # gelieferten Daten ignoriert
74
```



```
75     markup_s = ''
76     markup_s += self.readFile_p('form0.tpl')
77
78     markupV_s = self.readFile_p('form1.tpl')
79     lineT_o = string.Template(markupV_s)
80     markup_s += lineT_o.safe_substitute (
81
82         # -- Ihre Ergänzung
83
84     )
85
86     markup_s += self.readFile_p('form2.tpl')
87
88     return markup_s
89
90 #-----
91 def readFile_p(self, fileName_spl):
92 #-----
93     content_s = ''
94     with codecs.open(os.path.join('template', fileName_spl), 'r', 'utf-8') as fp_o:
95         content_s = fp_o.read()
96
97     return content_s
98 # 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 }
```