Aufgabenstellung 1 (für alle Gruppen): Projektinformationssystem

Fachliche Anforderungen

Sie sollen eine Web-Anwendung entwickeln, mit der eine Firma relevante Informationen zu kundenspezifischen Projekten verwalten kann, insbesondere:

- Verwaltung der Projektdaten
  - Anlegen, Ändern und Löschen von Projektdaten
  - Projektdaten sind z.B.
    - eindeutige Id
    - Nummer
    - Bezeichnung
    - Beschreibung
    - Bearbeitungszeitraum
    - Budget
    - Verweis auf den Kunden (Id des Kunden)
    - Verweise auf die beteiligten Mitarbeiter (Ids der Mitarbeiter)
    - erfasste Aufwendungen je Woche (Stunden) mit Zuordnung zu Mitarbeitern (Ids der Mitarbeiter)
- Verwaltung der Kundendaten
  - Anlegen, Ändern und Löschen von Kundendaten
  - Kundendaten sind z.B.
    - eindeutige Id
    - Nummer
    - Bezeichnung
    - Ansprechpartner
    - Ort
- Verwaltung der Mitarbeiterdaten
  - Anlegen, Ändern und Löschen von Mitarbeiterdaten
  - Mitarbeiterdaten sind z.B.
    - eindeutige Id
    - Name
    - Vorname
    - Funktion
- Auswertung:
  - Projektübersicht mit allen Projekten
    - nach Projektbezeichnung sortiert
    - Auflistung der Projektmitarbeiter, sortiert nach Name, Vorname
    - wöchentlicher Aufwand.

Die Datenbestände sollen als Listen zum Überblick und zur Auswahl einzelner Daten angeboten werden. Für jeden Datentyp muss ein Bearbeitungsformular vorgesehen werden, das die Erfassung neuer Daten bzw. die Bearbeitung bestehender Daten ermöglicht.

Beim Löschen von Daten muss auf die Integrität des Datenbestands geachtet werden.

Anforderungen an die Umsetzung - Schritt 1 (Termin P1)

Die Webanwendung “Projektinformationssystem” wird als Client-Server-Anwendung realisiert. In dieser Variante der Aufgabenstellung werden einzelne Webseiten, ggf. als Formulare, durch den Webserver erzeugt und ausgeliefert. Vermeiden dabei Sie grundsätzlich, Markup (HTML5) oder CSS direkt im Python-Code anzugeben: die serverseitige Erzeugung des Markup (HTML5) erfolgt deshalb mit Hilfe der **Template-Engine mako** (siehe Anhang).

Clientseitig werden nur die Standardmechanismen, die im User-Interface bei HTML5 direkt vorgesehen werden (Hyperlinks, einfache Formulare), verwendet. In Listen können Sie jeden Eintrag als Link implementieren und damit eine einfache Möglichkeit zum Übergang zu einem Detailformular schaffen.

Eine weitergehende clientseitige Bearbeitung erfolgt in diesem Schritt nicht.

Die Auswertung “Projektübersicht” wird jetzt noch nicht realisiert.

Anforderungen an die Umsetzung - Schritt 2 (Termin P2)

In Schritt 2 wird die Nutzung von Links in den Listensichten zum Wechsel in die Detailsichten ersetzt durch eine clientseitige Bearbeitung mit JavaScript:

- der zu bearbeitende Listeneintrag wird markiert
- die vorgesehene Funktion (z.B. “Ändern”) wird mit einem Schalter realisiert
  - die Bedienung des Schalters wertet die Markierung in der Liste aus und führt daraufhin die entsprechende Anfrage zur Darstellung der nächsten Webseite durch.

Die Auswertung “Projektübersicht” wird im Schritt 2 realisiert.

Weitere Anforderungen an die Umsetzung

- Webclient:
  - Verwendung HTML5 (XML-konforme Notation)
  - überprüfung des Markup mit Hilfe der w3c-Validator-Dienste
  - Präsentation mit CSS, ausgelagert in eine externe CSS-Datei
  - Verwendung JavaScript
- Webserver:
  - Verwendung Python (Version 3)
  - Verwendung Framework “cherrypy”
  - Verwendung der Template-Engine “mako” (siehe Anhang).

Die Verwendung von weiteren JavaScript-Bibliotheken / -Frameworks sowie CSS-Frameworks ist NICHT zulässig!

Verwenden Sie folgende Verzeichnisstruktur und erstellen Sie die angegebenen Dateien:

```
web
   /p1
       /pro                 <--- server.py
           /app             <--- __init__.py, application.py, database.py, view.py
           /content         <--- html-Datei(en), css-Datei(en), js-Datei(en)
           /data            <--- Daten in JSON-formatierten Dateien
           /doc             <--- Dateien der Dokumentation
           /template        <--- Vorlagen für mako
```

Sie können sich bei der Erstellung der Python-Module im Verzeichnis `app` *grundsätzlich* an den Vorlagen orientieren, die Ihnen gesondert zur Verfügung gestellt werden.

Anforderungen an die Dokumentation

Erstellen Sie abschließend eine Dokumentation, die Ihre Lösung (Schritt 1 und Schritt 2) beschreibt. Legen Sie dazu in einem Unterverzeichnis `doc` die Datei `pro.md` an. Sehen Sie folgende Gliederung vor:

- einleitend: Ihre Gruppenzugehörigkeit, Aufbau Ihres Team, Gültigkeitsdatum der Dokumentation
- allgemeine Beschreibung Ihrer Lösung
  - Aufgabe der Anwendung
  - Übersicht der fachlichen Funktionen
- Beschreibung der Komponenten des Servers
  - für jede Komponente:
    - Zweck
    - Aufbau (Bestandteile der Komponente)
    - Zusammenwirken mit anderen Komponenten
    - API (Programmierschnittstellen), die die Leistungen der Komponente anbieten
- Datenablage
- Durchführung und Ergebnis der geforderten Prüfungen.

Die Dokumentation wird als utf-8 kodierter Text mit der einfachen Auszeichnungssprache `Markdown` erstellt. Mit Hilfe des Werkzeugs `pandoc` (siehe https://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` sollen genutzt werden.

Bewertung / Testat

Zur Bewertung Ihrer Lösung im Hinblick auf die mögliche Erteilung des Testats müssen Sie vorlegen und erläutern:

- den von Ihnen erstellten Quellcode Ihrer Web-Anwendung in den Varianten für Schritt 1 und Schritt 2
- die von Ihnen erstellte Dokumentation.

Sie müssen die Lauffähigkeit Ihrer Lösungen und die Durchführung der Validierungen nachweisen.

Anhang

Nutzung der W3C-Validatordienste

Mit den W3C-Validatordiensten können Sie überprüfen, ob das von Ihnen verwendete Markup korrekt ist und Sie gültige CSS-Anweisungen verwendet haben.

Sie erreichen die Validatordienste so:

- **w3c-Validator-Dienst (Markup):** http://validator.w3.org/
  - Überprüfung der Korrektheit des Markup
  - Zeigen Sie den Quelltext der zu prüfenden Webseite an (z.B. Kontextmenü Webseite, dort etwa "Seitenquelltext" auswählen)
  - Den angezeigten Quelltext markieren und in die Zwischenablage kopieren
  - Inhalt der Zwischenablage in der Registerkarte "Direct Input" einfügen und Überprüfung starten
- **w3c-Validator-Dienste (CSS):** http://jigsaw.w3.org/css-validator/
  - Überprüfung von CSS-Stilregeln
  - weitere Vorgehensweise wie vor

Datenbasis

Zur Vereinfachung werden die Daten serverseitig im Verzeichnis `data` als `JSON`-formatierte Dateien abgelegt.

Sie können sich bei Ihrer Implementierung an folgenden Überlegungen orientieren:

- jeder Instanz jeder im Datenmodell genannten Klasse wird durch einen Objekt-ID eindeutig identifiziert
- als Objekt-ID kann eine Ganzzahl verwendet werden, die zentral verwaltet wird
  - der aktuelle Wert wird z.B. in einer Datei abgelegt, damit er bei jedem Start des Webservers weiter verwendet werden kann
  - Werte werden nicht neu verwendet, es wird bei Anlegen neuer Instanzen einfach durch Inkrementieren eine neue Objekt-ID erzeugt
- für jede Klasse wird eine eigene JSON-Datei verwendet, die alle Instanzen der Klasse enthält
- für Beziehungen müssen ggf. ebenfalls eigene JSON-Datein verwendet werden
  - Beziehungen werden anhand der Objekt-IDs identiziert
- das von Ihnen zu implementierende Modul `database.py` kapselt die skizzierte Verwaltung der persistenten Daten und stellt den anderen Modulen eine geeignete Schnittstelle zur Bearbeitung zur Verfügung
  - direkte Zugriffe auf die (interne) Form der persistenten Daten durch andere Module sind unbedingt zu vermeiden.

Mako-Template-Engine

Mit der Mako-Template-Engine können Sie insbesondere HTML-Seiten und -abschnitte einfach erzeugen. Mako ist in Python geschrieben, verfügt über eine Programmierschnittstelle in Python und erzeugt (intern) Python-Code zur Ausführung des übersetzten Templates.

Quelle und Installation

Sie finden die `mako template library` unter http://www.makotemplates.org/, dort insbesondere die zwar ausführliche, für den Anfänger aber manchmal etwas unübersichtliche Dokumentation.

Der Download-Bereich verweist auf den *Python Package Index (pypi)* . Verwenden Sie das angebotene Archiv, entpacken Sie es in ein temporäres Verzeichnis und installieren Sie es mit dem Befehl `python setup.py install` (falls Sie zwei Python-Versionen haben, geben Sie `python3` an).

Verwendung

Sie erstellen die Template-Dateien wiederum als UTF-8 kodierte Texte mit einem Texteditor (Ablage im Verzeichnis `template`).

Beispiel eines `mako`-Templates (z.B. Auszug aus einer Template-Datei `liste.tpl`):

```
## coding: utf-8
<table id="idList">
   <tr>
     <th>Name</th>
     <th>Typ</th>
   </tr>

   ## man verwendet hier Zugriff auf das Dictionary "data_o"

   % for key_s in data_o:
   <tr id="r${key_s}">
     <td>${data_o[key_s]['name']}</td>
     <td>${data_o[key_s]['typ']}</td>
   </tr>
   % endfor
</table>
## Mako-Kommentare verwenden zwei #-Zeichen
```

Die Kontrollflussanweisungen werden durch ein Prozentzeichen gekennzeichnet und sind ähnlich den Anweisungen in Python aufgebaut. Platzhalter, die durch den Wert des jeweiligen Ausdrucks ersetzt werden, beginnen mit einem Dollar-Zeichen. Der eigentliche Ausdruck wird durch geschweifte Klammern begrenzt.

Auf Einrückung muss nicht ausdrücklich geachtet werden, sehr wohl aber auf eine übersichtliche Schreibweise!

Das Modul `view.py` sieht dann etwa so aus (Ausschnitt):

```
# coding: utf-8

import os.path

from mako.template import Template
from mako.lookup import TemplateLookup

#-------------------------------------------------------
class View_cl(object):
#-------------------------------------------------------

   #-------------------------------------------------------
   def __init__(self, path_spl):
   #-------------------------------------------------------
      # Pfad hier zur Vereinfachung fest vorgeben
      self.path_s = os.path.join(path_spl, "template")
      self.lookup_o = TemplateLookup(directories=self.path_s)

   # ... weitere Methoden

   #-------------------------------------------------------
   def create_p(self, template_spl, data_opl):
   #-------------------------------------------------------
      # Auswertung mit templates
      template_o = self.lookup_o.get_template(template_spl)
      # mit der Methode render wird das zuvor 'übersetzte' Template ausgeführt
```

```
27          # data_o sind die im Template angegebenen Daten
28          # data_opl die übergebenen Daten
29          markup_s = template_o.render(data_o = data_opl)
30          return  markup_s
31
32       #-------------------------------------------------------
33       def createList_px(self, data_opl):
34       #-------------------------------------------------------
35          return self.create_p('liste.tpl', data_opl)
36
37    # EOF
```

In der Variable `data_opl` wird hier im Beispiel ein Dictionary übergeben, das dann in der Templatedatei zur Erzeugung des Markups einer Tabelle verwendet wird.

Hilfsmittel / Quellen

Hier finden Sie nochmals eine Zusammenstellung der Hilfsmittel und Quellen, die Sie verwenden sollen:

Hilfsmittel	Quelle	Beschreibung / Hinweise
w3c-Validator-Dienste (Markup)	http://validator.w3.org/	Überprüfung der Korrektheit des Markup
w3c-Validator-Dienste (CSS)	http://jigsaw.w3.org/css-validator/	Überprüfung von CSS-Stilregeln
mako Template Engine	http://www.makotemplates.org	in python implementierte Template-Engine, die python-Code erzeugt
JSON	http://www.json.org	Beschreibung des Datenformats JSON (die dort angegebenen Implementierungen benötigen Sie nicht; sie sind bereits in Python enthalten)
json	(Python-Standard-Modul)	Umwandeln python-Datenstrukturen <-> JSON
pandoc	http://pandoc.org/	Konverter, hier für markdown -> html5