Praktikum 3

Webanwendung (REST-Variante) "Mitarbeiterqualifizierung"

Einleitung

Sie haben im Praktikum 2 die fachliche Aufgabenstellung "Webanwendung Mitarbeiterqualifizierung" bearbeitet. Der Schwerpunkt der Implementierung lag im Praktikum 2 auf der Serverseite.

Im Praktikum 3 sollen Sie bei identischer fachlicher Aufgabenstellung ("Webanwendung Mitarbeiterqualifizierung") die Anwendung als "Single-Page-Application" erstellen und serverseitig REST-Interfaces verwenden.

Neue fachliche Vorgaben gibt es nicht. Datenmodell und prinzipielle Gestaltung der Benutzungsschnittstelle bleiben gleich.

Beachten Sie die Hinweise zur Implementierung der Benutzungsschnittstelle sowie zur Dokumentation.

Anforderungen an die Umsetzung

Single-Page-Application und REST

Erstellen Sie die Webanwendung "Mitarbeiterqualifizierung" als Client-Server-Anwendung, die die Architekturprinzipien *REST* (*Representational State Transfer*) und *Single-Page-Application* berücksichtigt:

- es wird einmalig das Dokument `index.html` als Webseite geladen
- weitere Inhalte werden dynamisch in Container, die im Markup des Dokuments `index.html` vorhanden sind, geladen
- alle notwendigen Ressourcen - CSS-Stilregeln in externen CSS-Dateien, JavaScript-Code in externen Quelldateien - werden durch das Dokument `index.html` referenziert und damit einmalig mit dem Laden des Dokuments geladen
- die Kommunikation zwischen Client und Server erfolgt asynchron im Hintergrund mit Hilfe der `Fetch-API` (die ursprüngliche Idee wurde *AJAX* genannt):

- Anforderungen, ggf. mit Daten, werden im Hintergrund asynchron an den Server gesendet:
  - gemäß den Prinzipien des REST-Ansatzes muss hierbei die HTTP-Methode angegeben werden
  - Daten werden im Message-Body als Key-Value-Paare übertragen (Standard, wie dies bei einfachen Formularen mit der Methode POST erfolgt)
- Daten, die vom Server an den Client übertragen werden, werden grundsätzlich gemäß der *JSON*-Konvention strukturiert.

Die Aufbereitung der Daten und die Erzeugung des Markups erfolgt auf dem Client mit einer Template-Engine. Die Templates selber werden als statische Ressource vom Server geliefert.

Method-Dispatching

Serverseitig wird *Method-Dispatching* verwendet: die HTTP-Methode entscheidet, welche Methode aufgerufen wird. In der Dokumentation zu `cherrypy` wird im Tutorial 7 gezeigt, wie man das *Method-Dispatching* verwendet (siehe http://docs.cherrypy.org/en/latest/tutorials.html?highlight=dispatching#tutorial-7-give-us-a-rest). Es ist **nicht** erforderlich, die im Abschnitt "Advanced" der Dokumentation zu `cherrypy` beschriebene Vorgehensweise "RESTFul-style dispatching" zur Auswertung komplexerer Pfade in den URI anzuwenden.

Konstruktion der REST-Anfragen

Zur Konstruktion der REST-Anfragen kann man z.B. von folgenden Überlegungen ausgehen (siehe auch: Datenmodell Praktikum 2, Implementierung Datenbasis):

- es gibt vier unabhängige Klassen:
  - `Mitarbeiter`
  - `Weiterbildung`
  - `Qualifikation`
  - `Zertifikat`
- `Teilnahme` ist eine Beziehung zwischen `Mitarbeiter` und `Weiterbildung` mit den Attributen `status` und `abschluss`
- weitere Beziehungen gibt es zwischen
  - `Mitarbeiter` und `Qualifikation` (verfügt über)
  - `Mitarbeiter` und `Zertifikat` (verfügt über)
  - `Weiterbildung` und `Qualifikation` (verfügt über)

- `Weiterbildung` und `Zertifikat` (verfügt über)
- `Teilnahme` und `Qualifikation` (erwirbt)
- `Teilnahme` und `Zertifikat` (erwirbt).

Dementsprechend kann man folgende Ressourcen definieren:

1. `Mitarbeiter`
2. `Weiterbildung`
3. `Qualifikation`
4. `Zertifikat`
5. `Teilnahme`
6. `Relation`

Bei 1. bis 4. haben die HTTP-Methoden diese Bedeutungen:

- `GET`
  - ohne Id: alle Elemente der Ressource liefern
  - mit Id: nur dieses Element der Ressource liefern
- `POST`: neues Element mit den gelieferten Daten erzeugen
- `PUT`: bestehendes Element (identifiziert per Id) mit den gelieferten Daten aktualisieren
- `DELETE`: bestehendes Element (identifiziert per Id) löschen

Die Verwendung von `Teilnahme` als eigene Ressource ermöglicht es, bei `Relation` ausschließlich zweiwertige Relationen zu verwenden. Die Methoden haben bei `Teilnahme` folgende Bedeutung:

- `GET`
  - ohne Id: alle Elemente der Ressource `Teilnahme` liefern
  - mit Mitarbeiter-Id: Teilnahmen des Mitarbeiters liefern
  - mit Weiterbildung-Id: Teilnahmen der Weiterbildung liefern
- `POST`: eine neue Teilnahme erzeugen (erfordert beide Ids)
- `PUT`: Angaben zur Teilnahme aktualisieren (erfordert beide Ids): Erfolg der Teilnahme, Teilnahmestatus
- `DELETE`: Teilnahme löschen (wenn das fachlich möglich ist) (erfordert beide Ids)

Bei 6. muss die Relation durch Angabe der beiden miteinander in Beziehung stehenden Ressourcen definiert werden:

- `GET`

    - ohne Id: alle Elemente der Relation liefern
    - mit einer Id: Relation einseitig auswerten
    - mit zwei Ids: Relations-Element liefern, ggf. mit den zugeordneten Attributen
- `POST` : eine neue Relation erzeugen (erfordert beide Ids)
- `PUT` : Angaben zur Relation aktualisieren (erfordert beide Ids) (falls es fachliche Attribute gibt)
- `DELETE` : Relation löschen (wenn das fachlich möglich ist) (erfordert beide Ids).

Die Übersichten zu den Mitarbeiterdaten und den Weiterbildungsdaten, die die Teilnahmen, Qualifizierungen und Zertifikate zeigen sollen, können als eigenständige *Ressource* aufgefasst werden. Ebenso können die Auswertungen als *Ressourcen* aufgefasst werden.

Beschreibung der REST-Anfragen (Beispiele)

Beispiel 1

Pflege Mitarbeiterdaten

Methode	URI	Reaktion
GET	/mitarbeiter/	alle Mitarbeiterdaten (Liste) anfordern
GET	/mitarbeiter/:mid	Daten eines Mitarbeiters gemäß :mid anfordern
POST	/mitarbeiter/ + Daten	Daten eines neuen Mitarbeiters speichern, :mid zurückgeben
PUT	/mitarbeiter/:mid + Daten	Daten des Mitarbeiters gemäß :mid ändern
DELETE	/mitarbeiter/:mid	Daten des Mitarbeiters gemäß :mid löschen (incl. aller abhängigen Daten)

Methode	URI	Reaktion

Methode	URI	Reaktion
GET	/mitarbeiteruebersicht/:mid	Übersicht (Teilnahmen, Qualifikationen, Zertifikate) für Mitarbeiter gemäß :mid anzeigen

Pflege Weiterbildungen

Methode	URI	Reaktion
GET	/weiterbildung/	alle Weiterbildungen (Liste) anfordern
GET	/weiterbildung/:wbid	Daten einer Weiterbildung gemäß :wbid anfordern
POST	/weiterbildung/ + Daten	Daten einer neuen Weiterbildung speichern, :wbid zurückgeben
PUT	/weiterbildung/:wbid + Daten	Daten der Weiterbildung gemäß :wbid ändern
DELETE	/weiterbildung/:wbid	Daten der Weiterbildung gemäß :wbid löschen (incl. aller abhängigen Daten)

Methode	URI	Reaktion
GET	/weiterbildunguebersicht/:wbid	Übersicht (Teilnahmen, Qualifikationen, Zertifikate) für Weiterbildung gemäß :wbid anzeigen

Beispiel 2

Relation Mitarbeiter - Qualifikation

Methode	URI	Reaktion
GET	/mitarbeiterqualifikation/	alle Qualifikationen aller Mitarbeiter anfordern
GET	/mitarbeiterqualifikation/?mid=:mid	alle Qualifikationen eines Mitarbeiters gemäß :mid
GET	/mitarbeiterqualifikation/?mid=:mid&qid=:qid	eine Qualifikation eines Mitarbeiters gemäß :mid/:qid anfordern
POST	/mitarbeiterqualifikation/?mid=:mid + Daten	Daten einer neuen Qualifikation eines Mitarbeiters gemäß :mid speichern, :qid zurückgeben
PUT	/mitarbeiterqualifikation/?mid=:mid&qid=:qid + Daten	Daten der Qualifikation eines Mitarbeiters gemäß :mid/:qid ändern
DELETE	/mitarbeiterqualifikation/?mid=:mid&qid=:qid	Daten der Qualifikation eines Mitarbeiters gemäß :mid/:qid löschen (incl. aller abhängigen Daten)

Beispiel 3

Auswertungen

Methode	URI	Reaktion
GET	/auswertungmitarbeiter/	Auswertung für alle Mitarbeiter anfordern
GET	/auswertungweiterbildungen/	Auswertung für alle Weiterbildungen anfordern
GET	/auswertungzertifikate/	Auswertung für alle Zertifikate anfordern

Die clientseitig zu verwendenden Templates werden mit folgender Anforderung geladen:

Methode	URI	Reaktion

Methode	URI	Reaktion
GET	/template/	Templates

Mit Doppelpunkt gekennzeichnete Elemente sind variable Werte. Die Daten bei PUT- und POST-Anfragen werden im Message-Body übertragen.

Die Ergebnisse der Anfragen werden stets im JSON-Format zurückgeliefert. Der HTTP-Statuscode wird verwendet, um auf korrekt durchgeführte oder nicht ausführbare Anfragen hinzuweisen (z.B. 200 = Ok, 404 = angefragte Daten sind nicht vorhanden).

Implementierung der Benutzungsschnittstelle

Anforderungen an die Gestaltung

Nehmen Sie die Gestaltung mit Hilfe von CSS vor. Berücksichtigen Sie dabei:

- verwenden Sie Farbschemata (Vorder-/Textfarben und Hintergrundfarben), die die Lesbarkeit sicherstellen
- ordnen Sie Schalter in Listen- und Detailsichten so an, dass sich eine übersichtliche und dem Bedienablauf entsprechende Anordnung ergibt
- ordnen Sie Führungstexte (HTML-Element `label`) und die zugehörigen Interaktionselementen übersichtlich an.

Tabellen (HTML-Element `table`) dürfen grundsätzlich nicht zur Erzielung eines Layouts verwendet werden.

Benutzen Sie dazu andere Elemente, die Sie mit absoluter und/oder relativer Positionierung und/oder mit Flexboxes und/oder Grids anordnen. **Die Positionierung mit `float` darf nicht verwendet werden**.

Das Layout des Inhaltsbereichs müssen Sie mit Hilfe von Flexboxes oder Grids definieren.

Strukturierung Datenpflege und Bedienung

Die Benutzungsschnittstelle verwendet bei der Pflege von Daten im wesentlichen zwei verschiedene Darstellungsformen:

- Listensichten:
  - Darstellung der Informationen in einer Liste, die in der Regel als Tabelle (HTML-Element `table`) realisiert wird

    - **einzelne Zeilen werden mit einem Mausklick auf die Zeile ausgewählt**; die Auswahl mit Radiobuttons oder Checkboxen oder ähnlichen Hilfsmitteln ist **nicht** zulässig
    - Operationen, die auf eine ausgewählte Zeile angewendet werden können (z.B. ändern), werden als separate Schalter angeboten; die Verwendung von Schaltern / Links in jeder Zeile ist **nicht** zulässig
- Detailsichten:
    - Darstellung von Informationen in Formularen
    - Überprüfung von Eingaben mit den in HTML5 vorgesehenen Mitteln sowie zusätzlichen, in JavaScript realisierten Prüfungen, falls erforderlich.

Verwendung von JavaScript-Komponenten zur Client-Implementierung

Sie **müssen** zur Implementierung die folgenden JavaScript-Komponenten verwenden:

- Eventservice:
    - clientseitiger asynchroner Datenaustausch zwischen JavaScript-Komponenten
    - Implementierung in `evs.js`
- Template-Engine:
    - Implementierung in `tco.js` und `tmg.js`
- Wrapper für die Nutzung der Fetch-API:
    - Implementierung in `req.js`.

Die Quellen sowie eine Beschreibung finden Sie in dieser Archivdatei.

Sie finden dort auch eine Anwendung ("Demonstrator"), die die Verwendung der einzelnen JavaScript-Module demonstriert.

Implementierung der Datenbasis

Berücksichtigen Sie folgende Vorgaben:

- verwenden Sie für jeden Strukturtyp (z.B. "Mitarbeiter", "Weiterbildung" etc.) ein eigenes Verzeichnis im Verzeichnis `data` und benennen Sie es passend
- verwenden Sie für jedes Datenobjekt zur Speicherung eine einzelne JSON-Datei
    - verwenden Sie bei der Speicherung den Parameter `indent=3`, damit die Inhalte leicht lesbar sind
    - benennen Sie die Datei in der Form `<id>.json`

    - verwenden Sie für die Bildung der eindeutigen Identifikation UUID-Zeichenketten (UUID: Universally Unique Identifier) (siehe https://docs.python.org/3/library/uuid.html); verwenden Sie UUID1 oder UUID4
- verwenden Sie für die Relationen Schlüssel z.B. in der Form `<typ1>_<id1>_<typ2><id2>`, so dass Sie mit Mustervergleichen einfach auf Teilmengen zugreifen können.

Weitere Anforderungen

- Webclient:
    - Verwendung HTML5 (XML-konforme Notation)
    - Überprüfung des Markup mit Hilfe der w3c-Validator-Dienste
    - Präsentation mit CSS, ausgelagert in eine externe CSS-Datei
        - **die Verwendung von CSS-Frameworks wie z.B. "bootstrap" ist nicht zulässig**
    - Verwendung JavaScript, ausgelagert in eine oder mehrere externe JS-Dateien
        - **die Verwendung von (anderen) JavaScript-Frameworks ist nicht zulässig**
        - verwenden Sie Klassen, die Sie ab ES5 mit dem Schlüsselwort `class` definieren können
        - es **müssen** die oben angegebenen JavaScript-Komponenten verwendet werden
    - die direkte Einbettung von Markup in JavaScript-Code ist nicht zulässig
        - Sie **müssen** die oben angegebene Template-Engine verwenden
    - Sprache Benutzungsschnittstelle: Deutsch
- Webserver:
    - Verwendung Python 3
    - Verwendung Framework `cherrypy`
    - Method-Dispatching einsetzen
    - eine sinnvolle Aufteilung in Module, die den unterschiedlichen Aufgaben (RequestHandler ["Controller"], Views, Database) gerecht wird; sehen Sie für jede Klasse ein eigenes Modul vor
    - Datenspeicherung: siehe oben.

Nutzen Sie die Möglichkeiten objektorientierter Programmierung!

Verzeichnisstruktur

Verwenden Sie folgende Verzeichnisstruktur und erstellen Sie die angegebenen Dateien:

```
web
   /p3
       /mq                    <--- server.py
           /app               <--- Module der serverseitigen Lösung
           /content           <--- html-Datei(en), css-Datei(en), js-Datei(en); ggf.
weitere Unterverzeichnisse
           /data              <--- Daten in JSON-formatierten Dateien; ggf. weitere
Unterverzeichnisse
           /doc               <--- Dateien der Dokumentation
           /templates         <--- Vorlagen für clientseitige Templates
           /test              <--- Test-Skripte/-daten für Tests der REST-Interfaces
und Testergebnisse
```

Test REST-Interfaces

Zum Test der REST-Schnittstellen verwenden Sie eine der beiden nachfolgend beschriebenen Werkzeuge:

- (Möglichkeit 1) Erweiterung **REST Client** für Visual Studio Code
- (Möglichkeit 2) Werkzeug **cURL**. Varianten dieses Werkzeugs für gängige Betriebssysteme und Hinweise zur Anwendung finden Sie unter
  - https://curl.haxx.se, insbesondere https://curl.haxx.se/docs/manual.html
  - http://alvinalexander.com/web/using-curl-scripts-to-test-restful-web-services
  - oder werten Sie eine Suche bei Google z.B. mit den Stichworten "cURL REST" aus.

Erstellen Sie für die einzelnen Ressourcen und Methoden Testskripte an und speichern Sie diese zusammen mit den Ergebnisse im Verzeichnis `test`.

Anforderungen an die Dokumentation

Erstellen Sie eine Dokumentation, die Ihre Lösung beschreibt. Legen Sie dazu in einem Unterverzeichnis `doc` die Datei `mq.md` an. Sehen Sie folgende Gliederung vor:

- einleitend: Namen / Matrikelnummern der Mitglieder Ihrer Gruppe / 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
- Konfiguration
- 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 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" sollen (!) genutzt werden.

Die Dokumentation muss in Deutsch verfasst werden!

Bewertung / Testat

Zur Bewertung Ihrer Lösung im Hinblick auf die mögliche Erteilung des Testats müssen Sie den von Ihnen erstellten Quellcode Ihrer Web-Anwendung vorlegen und erläutern.

Sie müssen die Lauffähigkeit Ihrer Lösung und die Durchführung der Validierungen (Markup, CSS-Stilregeln) 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