Entwurfsbeschreibung

Yannik Völker

26. Mai 2015

Inhaltsverzeichnis

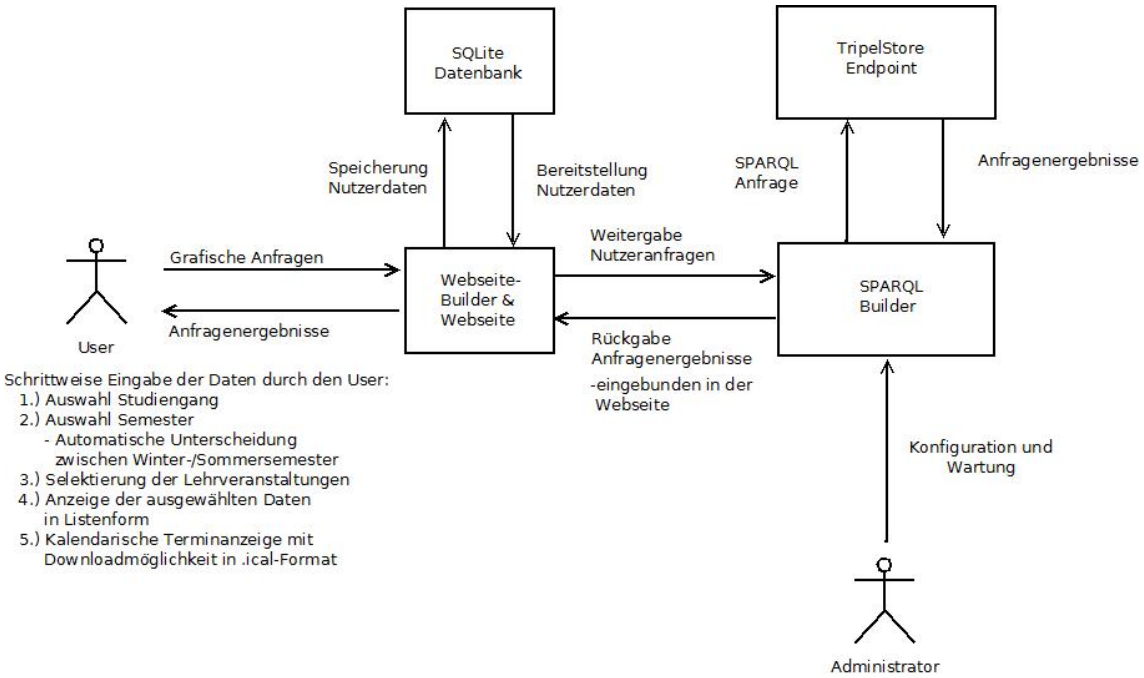
1 Allgemeines

Die jetzige Situation an der Universität Leipzig ist, dass man mit der Belegung der Module die Termine der einzelnen Veranstaltung teilweise von unterschiedlichen Quellen erhält. Zudem sind die existierenden Veranstaltungsübersichten (bspw. die der Fakultät für Informatik[1]) häufig zu unübersichtlich, sodass auch hier eine Stundenplan-Zusammenstellung aufwendig ist.

Das Ziel unseres Projektes ist deswegen die Erstellung personalisierter Stundenpläne auf Basis der Open-Data-Schnittstelle od.fmi.uni-leipzig.de. Diese Schnittstelle hält die Informationen zu den Veranstaltungen der Fakultät für Mathematik und Informatik als RDF-Datenbank. Diese wollen wir nutzen, um eine einfachere Möglichkeit des Stundenplanerstellens zu bieten.
Die Informationen über Kurse und Seminare werden aus der od-Schnittstelle ausgelesen und übersichtlich zusammengefasst, so dass ein Nutzer sich einen auf sich zugeschnittenen Stundenplan erstellen und am Ende als ical-Datei herunterladen kann. Realisiert wird dies als Webanwendung, welche platformübergreifende und komfortable Bedienung ermöglicht.

[1] `https://www.informatik.uni-leipzig.de/ifi/studium.html` (01.04.2015)

2 Produktübersicht



Nachdem der Webserver entsprechend konfiguriert und das Pythonprogramm dort hinterlegt wurde, kann man die Webseite mit jedem gängigen Browser erreichen. Der Nutzer wird zuerst auf eine Startseite gelangen, auf der eine kurze Einführung steht. Nun wird nach und nach immer feiner gesiebt, welche Veranstaltungen der Nutzer besucht. Zuerst wird nach dem Studiengang gefragt, danach nach dem Semester. Das Programm kann zwischen Sommer- und Wintersemester unterscheiden - also wird während einem Sommersemester beispielsweise nichts aus dem Wintersemester angezeigt.

Nun wird eine Vorauswahl der Lehrveranstaltungen getätigt. Zusätzlich kann man noch andere Module per Klick hinzufügen. Anschließend werden alle Module mit den Übungs- und Praktikaterminen gelistet, wobei man hier ebenfalls per Klick eine Wahl trifft. Der letzte Schritt ist die Anpassung der gewählten Lehrveranstaltungen, hier wird die Möglichkeit geboten einzelne Lehrveranstaltungen bereits vor offiziellem Vorlesungsende aufhören zu lassen. Schlussendlich gelangt der Benutzer auf die kalendarische Terminanzeige, welche im zwei Wochenformat gehalten ist. Dies wird als Download als .ical-Format angeboten.

All dies geschieht, ohne dass der Nutzer ein Konto anlegen oder sich einloggen muss. Zur

späteren Nachkonfiguration oder erneutem abrufen des Stundenplans durch den Benutzer wird jeder generierte Stundenplan verknüpft mit einer id gespeichert. Ein späterer Zugriff ist dann über eine generierte URL möglich. Dem Nutzer werden bei erneutem Aufruf auch Änderungen seit dem letzten Herunterladen am Stundenplan angezeigt.

3 Grundsätzliche Struktur- und Entwurfsprinzipien

Die Anwendung ist in Python geschrieben und verwendet Flask als minimalistisches Webframework. Zusätzlich werden unter anderem externe Module für SPARQL, iCallender und den Zugriff auf SQL-Datenbanken genutzt.

Die einzelnen Module des Programmes, welche jeweils eine klar umrissene Funktionalität erfüllen, liegen in seperaten dateien. Zusätzlich gibt es jinja2-Templates, über welche die verarbeiteten Daten in HTML-Seiten überführt werden.

Bei Auswahl von Modulen und Veranstaltungen werden stests SPARQL-Anfragen an den Triple-Store `od.fmi.uni-leipzig.de` gesendet, es findet kein caching statt.

Die Stundenpläne werden in einer SQL-Datenbank gespeichert, um bei wiederholter Benutzung der Anwendung auf die vorherige Auswahl zurück greifen zu können.

4 Struktur- und Entwurfsprinzipien einzelner Pakete

Im folgenden wird der genauere Aufbau der einzelnen Module erläutert. Dabei wird in Python-Module und Module für die HTML-Templates unterschieden. Zusätzlich wird dann noch der Aufbau der SQL-Datenbank beschrieben, welche für die Speicherung der Userdaten genutzt wird.

4.1 Python-Module

Das Modul `app.py` bildet den Kern des Programms. Die Inhalte der einzelnen Webseite werden hier mit hilfe anderer Module (`SPARQL-Modul` und `Zeit-Modul`) generiert und anschließend in die Forms, sowie in die Webseite angebunden. Jede Funktion in diesem

Modul, bis auf zwei Ausnahmen, welche die Datenbankverbindung herstellen bzw trennen, generiert die Daten für eine HTML-Seite.

Das Modul `forms.py` dient zur Konfiguration des externen moduls WTForms, welches von uns zur Objektorientierten Formgenerierung und insbesondere Eingabevalidation genutzt wird.

Die Abfragen an die Datenbank `od.fmi.uni-leipzig.de` finden im SPARQL-Modul `sparql_requests.py` statt. Für jede von uns Benötigte Abfrage enthält diesen Modul eine Funktion, welche Python-Parameter entgegen nimmt, mit diesen eine SPARQL-Abfrage erstellt und ausführt, so wie deren Ergebnisse in Python-Datenstrukturen umwandelt und zurückgibt.

Das Modul `ical.py` konvertiert Stundenplandaten in Kalenderformate, genauer entweder in das iCalendar-Format, oder in eine zweiwöchentliche Tabelle.

Das Modul `time-conversion.py` stellt Funktionen zur Verfügung um sich im Studienjahr zu orientieren, zum Beispiel um das Momentane Semester zu bestimmen, oder das nächste Vorlesungsende.

4.2 Config-Datei und Kommandozeilen-Parameter

im Verlauf des Projektes ist es ersichtlich geworden, dass die Anwesenheit einer Config-Datei und Kommandozeilen-Parameter notwendig sind. Es wurde daher eine Config-Datei und ein eigenes Modul zur Verarbeitung angelegt, mit der die Defaultwerte für den Host und den Port, der eingestellt werden soll, die URL, welche als Datenquelle angegeben wird und den Pfad der Datenbank, welche im Verlauf des Programmes angelegt wird. Zusätzlich zu der Config-Datei können der `app.py` Kommandozeilen-Parameter mitangegeben werden, welche ebenfalls die notwendigen Parameter für eine Ausführung im Terminal mit übergeben.

4.3 HTML-Templates

Die jinja2-Templates stellen den View dar, sie sind ein festes HTML-Gerüst, in welches übergebene Daten eingearbeitet werden.

Die Basis fast aller weiteren HTML-Templates bildet dabei die `base.html`. In diesem wir die Struktur der Website Aufgebaut, und anderen modulen innerhalb dieser ein paar Blöcke zum überschreiben zur Verfügung gestellt. Dies vermeidet Redundanzen in den restlichen Templates, da so nicht jedes bsp. das stylesheet einzeln einbinden muss.

Ein weiteres Hilfstemplate ist die `_formhelpers.html`. Dieses rendert Formularelemente in Form von Listen. Auch durch dieses Modul werden Redundanzen vermieden.

Über die Templates `semesterwahl.html` und `studiengangswahl.html` werden Webseiten erzeugt, auf denen der User dazu aufgefordert aus jeweils einer Liste mit Studiengängen und Semester, das für ihn jeweils Passende auszuwählen.

Im Template `addremovemodule.html` wird dann die dynamische An- und Abwahl von Kursen verarbeitet. Dies wird ermöglich durch eine Aktualisierung der Website mit der dazugehörigen Liste der bereits ausgewählten Kurse.

Alle weiteren Templates, welche hier nicht ausführlich beschrieben wurden, erzeugen dann jeweils eine Webseite. Diese werden nach bestätigter Kursauswahl durch eine SPARQL-Abfrage aus der Datenbank der `od.fmi.uni-leipzig.de` ausgelesen und dem User zur Auswahl präsentiert. Schließlich wird es dem User möglich sein, sich die ausgewählten Kurse in Form einer `.ical`-Datei herunter zu laden und diese in sein persönlich genutztes Kalenderprogramm zu importieren.

4.4 SQLite-Datenbank

5 Datenmodell

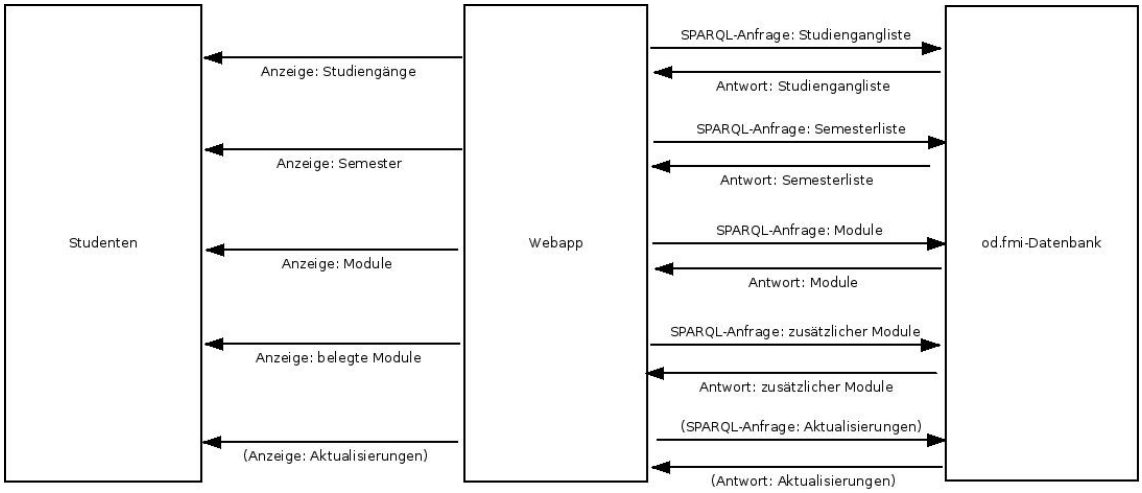


Abbildung: Datenmodell

Für unser Projekt ist es essentiell, dass bestimmte Daten, welche von User eingegeben werden, von uns gespeichert werden. Zu diesen Daten gehören neben den ausgewählten Kursen, auch der Studiengang und das Semester. Dies geschieht, damit der User beim erneuten Besuch auf unseren Websites alle Module und Termine nicht erneut auswählen muss.

Um die verschiedenen Nuzter zu unterscheiden, bekommt jeder User eine individuelle `ID`. Diese `ID` wird durch eine Methode zufällig generiert und mit allen Eingaben des Users in einer `SQLite-Datenbank` gespeichert.

Im folgenden wird der Aufbau der SQLite-Datenbank beschrieben, welche die Daten der Nutzer verwaltet. Das Schema der Datenbank mit den Informationen zu den Nutzern ist das Folgende:

```
CREATE TABLE `student` (
        `id`                    VARCHAR(40) NOT NULL,
        `studiengang`   VARCHAR(80),
        `semester`              SMALLINT,
```

```
        `module`                    BLOB,
        `lastdownload`   DATE,
        PRIMARY KEY(id)
);
CREATE TABLE `lv` (
        `uri`                       VARCHAR(80) NOT NULL,
        `lastchange`     DATE,
        `begin`                     VARCHAR(5),
        `end`                       VARCHAR(5),
        `room`                      VARCHAR(20),
        `talker`                    VARCHAR(?),
        PRIMARY KEY(uri)
);
CREATE TABLE `choice` (
        `id`                        VARCHAR(40) NOT NULL,
        `uri`                       VARCHAR(80) NOT NULL,
        `enddate`                   DATE,
        PRIMARY KEY(id, uri),
        FOREIGN KEY(id) REFERENCES student(id),
        FOREIGN KEY(uri) REFERENCES lv(uri)

);
```

`student` repräsentiert einen Studenten. Dabei ist `id` ein zufälliger sha1-hash der zum späteren Aufruf der gespeicherten Inhalte vorgesehen ist, `studiengang` enthält die URI des Studiengangs im SPARQL-Triple-Store. `module` beinhalt eine serialisierte Python-Liste, welche die vom Nutzer gewählten Module in Form der entsprechenden URIs beinhalten. `lastdownload` ist der Zeitpunkt zu dem dieser Stundenplan das letzte mal heruntergeladen wurde, durch ein vergleich mit `lastchange` kann herrausgefunden werden, ob sich der Stundenplan geändert hat.

Die Tabelle `lv` beinhaltet alle relevanten Informationen zu Lehrveranstaltungen, so wie das Datum ihrer Letzten Änderung (`lastchange`).

`choice` ist einen n-n-Verknüpfung zwischen `student` und `lv`, beinhaltet also welche LVs von welchen Studenten belegt werden, zudem noch das von den Studenten selbst festgelegte eventuell veränderte Enddatum in `enddate`.

6 Testkonzept

Da einige Funktionen innerhalb des Projektes sich auf Daten des Triplestores beziehen, gestaltet sich die Test-Automatisierung dieser Funktionen außerordentlich schwierig. Deswegen muss bei diesen Methoden, die hauptsächlich „sparql-requests.py“ betreffen, hauptsächlich auf die Richtigkeit der ausgegebenen Daten getestet werden. Da viele der Funktionen sowieso statisch sind, also keine Parameterübergabe stattfindet, ist die manuelle Verifizierung der Funktionen vereinfacht. Zudem sollten sich gerade diese Funktion nicht häufig ändern, sodass automatische Tests vernachlässigbar sind

Die restlichen Funktionen werden, wie bereits im Qualitätssicherungskonzept[2] beschrieben, mit automatisierten Tests verifiziert. Hierbei reichen für kleinere Funktionen die doctest-Funktionalitäten. Falls größere Funktionen entstehen, die umfangreichere Tests benötigten, wird hierfür unittest benutzt, welches ebenfalls bereits im Qualitätssicherungskonzept beschrieben wurde. In beiden Fälle sollte mit dem Tool „Coverage“ darauf geachtet werden, dass alle zu testenden Bereiche des Quelltextes abgedeckt werden.

Schließlich werden für die Systemtests das Zusammenspiel der Funktionen, die unter Anderem die Konformität des entstehenden iCalenders und der Webseite betreffen, getestet. Die dafür benutzten Werkzeuge wurden ebenfalls bereits im Qualitätssicherungskonzept beschrieben.

[2] unter „3. Testkonzept“ (S. 4ff)

7 Glossar

***.py** Dateiendung für Programme in der Programmiersprache Python

doctest Methode in Python, zur einfachen Erstellung von Modultests, der eigentliche Test befindet sich dabei in einem Docstring.

Dropdown-Liste Eine Dropdown-Liste ist ein grafisches Bedienelement, welches beim Aktivieren dem Nutzer mehrere Auswahlmöglichkeiten in einer Liste bietet

Flask ein auf Python basierendes Web-Framework

Forms ist ein Element von HTML, welches Formulare darstellt. Formulare können verschiedene Formate haben (bspw. mit Radiobuttons, Checkboxen, etc.)

Hash eine Prüfsumme, die dazu dient, Datensätze auf Gleichheit zu prüfen

iCalendar ein Datenformat zum Austausch von Kalenderinhalten

JSON-Format kompaktes Datenformat in einer einfachen lesbaren Textform zum Zweck des Datenaustauschs zwischen Datenanwendungen.

open data bedeutet die freie Verfügbar- und Nutzbarkeit meist öffentlicher Daten

Python Programmiersprache

Radiobutton Ein Radiobutton ist ein Bedienfeld einer grafischen Oberfläche, welche von einem Nutzer eine Auswahl verlangt

RDF Tripel von Daten, welche Beziehungen zwischen repräsentieren

SPARQL ist eine graph-basierte Abfragesprache für RDF

SQLite Programmbibliothek, die ein relationales Datenbanksystem enthält

Stylesheet ist ein Element von Webseiten, welches die Gestalt der Webseite bestimmt

Tag bezeichnet zumeist ein in Klammern eingeschlossenes Element, welches dazu dient, dazwischenliegende Textelemente zu kennzeichnen, bspw. den Titel einer HTML-Seite

Template ist eine Vorlage für die HTML-Seiten, die mit Hilfe von Flask zu vollständigen Webseiten

zusammengesetzt werden

Triplestore (Virtuoso) ist eine RDF-Datenbank, die eine SPARQL-Schnittstelle zur Verfügung stellt. Hier werden die Daten, die für die Stundenplan-Generierung nötig sind, abgefragt

URL (auch Uniform Resource Locator) identifiziert und lokalisiert eine Website oder auch andere Rescourcen in einem Computernetzwerk