IWGS2_SS20_Zettel5

July 7, 2020

1 Informatische Werkzeuge in den Geistes- und Sozialwissenschaften II

1.1 Hausaufgabe 5 (Einführung in SQL-Datenbanken)

Erschienen: 30.05.2020
Abgabe bis: 07.06.2020

Bitte laden Sie Ihre Notebooks bis 23:59 Uhr am Abgabetag in Ihrer Übungsgruppe bei StudOn hoch.

Wenn Ihnen einige der hier verwendeten Konzepte unbekannt sind oder Sie nicht wissen, wie Sie fortfahren sollen, können Sie die Vorlesungsunterlagen zu Rate ziehen oder jederzeit Fragen im Forum oder auf Slack stellen, im Tutorium nachfragen, oder Ihrem Tutor eine Mail schreiben.

1.2 Aufgabe 5.1 (Aufgabenteilung, 20 Punkte)

Beatrice war die letzte Woche im Urlaub und Sie haben seitdem nichts von ihr gehört. Völlig unabhängig davon sind Sie seit etwa einer Woche merkwürdig entspannt. Ein wenig vermissen Sie jedoch die gemeinsamen Abenteuer. Außerdem würden Sie ihr gerne von der letzten Woche erzählen, als Sie praktisch ganz alleine die Welt gerettet haben. Sie wählen Beatrices Nummer. Es klingelt keine zwei Mal und Beatrice hat schon abgehoben. Sie erzählt von ihrem Urlaub. Sie habe einen echten "Programmier-Guru" kennengelernt und habe einiges gelernt, was sie gerne in dem Webserver ausprobieren würde. Unter anderem würde sie gerne "die Aufgaben besser verteilen".

Sie fragen, wie sie denn die Aufgaben zwischen sich und Ihnen aufteilen möchte. Noch während Sie die Frage stellen, dämmert es Ihnen. Beatrice meint nicht die Aufteilung von Aufgaben zwischen Ihnen beiden. Stattdessen möchte sie nicht mehr allen Code in einer einzigen Datei haben, denn das wird langsam unübersichtlich. Den Code sollte man auf zwei Skripte aufteilen und damit in "die jeweiligen Aufgaben aufteilen".

In den letzten Aufgaben haben wir all unseren Code in einer einzigen Datei geschrieben. Das ist in Ordnung für kleinere Programme. Da unser Server aber langsam wächst, weichen wir jetzt ein wenig von diesem Muster ab.

Mit dieser Aufgabe finden Sie zwei Python-Dateien, *image_database.py* und *image_server.py*. Die Aufgabenteilung zwischen den beiden Skripten ist die folgende: - *image_database.py* liest eine CSV-

Datei (im Ordner *metadata*) und baut daraus eine SQL-Datenbank, ähnlich wie wir es im letzten Aufgabenblatt gesehen haben. - *image_server.py* stellt unseren Bottle-Webserver dar. Dieser arbeitet nur noch auf der SQL-Datenbank und den Bildern (im Ordner *data*).

Die Skripte aufzuteilen hat den Vorteil, dass wir die SQL-Datenbank nur einmal bauen müssen und danach im Server immer verwenden können. So muss die Datenbank nicht bei jedem Start des Programms neu gebaut werden. Es macht außerdem die Skripte leichter verständlich, weil der jeweilige Code etwas kürzer wird.

In dieser Teilaufgabe kümmern wir uns um die Erstellung der Datenbank, also arbeiten am Skript *image_database.py*. Wir haben außerdem eine deutlich umfangreichere Metadata-Datei (*metadata/Marburg_Metadaten_Kirmes.csv*). Machen Sie sich mit dem neuen Format vertraut. Anders als die letzten Wochen sind hier die Einträge mit Semikolons getrennt, nicht mit Kommas. Das müssen Sie beachten, wenn Sie den Delimiter angeben.

Aufgabe: Im Skript *image_database.py* finden Sie 2 Stellen, die mit `TODO` markiert sind. An der ersten Stelle sollen Sie die Tabelle *Images* in der Datenbank erstellen. Ähnlich zur Metadata-Datei soll die Spaltenstruktur dieser Tabelle folgendes Format haben: - Title - Subtitle - Archive - Artist - Location - Date - Genre - Material - FileName - URL

Alle Spalten sollen Text speichern. Denken Sie daran, wie in der letzten Hausaufgabe auch schon, den Fall zu behandeln, dass die Tabelle schon existiert. In diesem Fall, löschen Sie die bisher vorhandende Tabelle einfach vorher (`DROP`).

Im zweiten `TODO` sollen Sie die Tabelle mit Daten füllen. Nutzen Sie dazu den `INSERT`-Befehl für SQL.

Wichtig: Testen Sie Ihr Programm und überprüfen Sie ob Ihre Datenbank richtig gefüllt wird. Im IWGS-Forum befindet sich ein Beitrag, in dem nochmal beschrieben wird, wie wir Datenbanken in externen Programmen inspizieren können.

1.3 Aufgabe 5.2 (Server, 20 Punkte)

Wie jede Woche, müssen Sie auch in dieser Aufgabe wieder den Port ändern, unter dem der Webserver gestartet wird. Dies ist wieder die letzte Zeile in *image_server.py*.

Wir geben eine weitere Datei in dieser Woche vor: *show_table.tpl*. Auch hier findet sich nichts, was wir nicht schon mehrmals in IWGS gemacht haben. Deshalb ersparen wir Ihnen diese Woche hier die Tipparbeit. Schauen Sie sich die Datei an und verstehen Sie was dort gemacht wird. Am Anfang der Datei befindet sich ein ausführlicher Kommentar. Machen Sie sich vor allem damit vertraut, welche Python-Daten diese Datei tatsächlich braucht (sie braucht nämlich nicht alle Spalten unserer Datenbank-Tabelle).

Aufgabe: Ihre Aufgabe ist es, *show_table.tpl* zu verwenden um den Inhalt der Datenbank anzuzeigen. Implementieren Sie dazu in *image_server.py* eine Route *show_all*. Diese Route soll aus der Datenbank alle Einträge (also alle Zeilen) abfragen. Stellen Sie dabei sicher, nur die Daten (die Spalten) abzufragen, die auch in *show_table.tpl* gebraucht werden. Auch die Reihenfolge muss übereinstimmen.

1.4 Aufgabe 5.3 (Datenbank durchsuchen, Teil 1, 30 Punkte)

Der große Vorteil einer Datenbank (im Vergleich zB zu einer Python-Liste) ist, dass wir sie effizient durchsuchen können. Dies machen wir uns in dieser Aufgabe zunutze. Im letzten Semester haben wir HTML-Formulare kennengelernt, in die BenutzerInnen Informationen eingeben können. Allerdings haben wir nie wirklich etwas mit den eingegebenen Daten gemacht. Das endet jetzt!

Aufgabe: Vervollständigen Sie die Datei *home.tpl*. Fügen Sie ein HTML-Formular ein. Das Formular soll zwei Textfelder enthalten, mit denen nach KünstlerInnen und/oder Bildtiteln gesucht werden kann. Außerdem soll das Formular einen Submit-Button enthalten. Zur Erinnerung: Der Submit-Button leitet auf eine andere HTML-Route weiter. Diese Route kann dann die eingegebenen Daten verarbeiten. In unserem Fall soll Ihr Submit-Button auf eine Route `/search` weiterleiten.

1.5 Aufgabe 5.3 (Datenbank durchsuchen, Teil 2, 30 Punkte)

Im letzten Schritt verarbeiten wir die eingegebenen Daten und durchsuchen damit unsere Datenbank. Dazu vervollständigen Sie die Route `/search`. Für diese haben wir schon etwas Code vorgegeben. Wichtig ist hier vor allem, dass die Funktion nicht als `@route` markiert ist, sondern als `@get`. Dies erlaubt es uns, die Daten, die ein User im letzten Schritt in die Eingabemaske eingegeben hat, abzufragen.

Wir nennen diese Art der Kommunikation *GET*-Requests. Diese erlauben die Datenübertragung (und damit die Übergabe von Parametern) vom Browser an die aufgerufene Webseite. Im Fall von *GET*-Requests werden die Parameter dabei über die URL übertragen. Wenn User beispielsweise als Suchbegriffe "david" als Künstler und "kirmes" als Bildtitel eingeben, wird die aufgerufene URL folgendes Format haben: http://jupyter.kwarc.info:32500/search?author=david&title=kirmes. Die Argumente starten hier mit einem Fragezeichen und mehrere Argumente werden mit & aneinandergehängt. Indem wir die Route mit `@get` markieren, erledigt Bottle das Extrahieren der übergebenen Werte für uns.

Aufgabe: Fragen Sie die Daten aus der Datenbank ab, die den Eingaben entsprechen. Um die Daten in der Datenbank zu filtern, verwenden Sie das SQL Schlüsselwort `WHERE`. In unserem Fall wollen wir nicht, dass BenutzerInnen den exakten Namen der KünstlerInnen oder den exakten Titel eines Bildes eingeben müssen. Deshalb verwenden wir außerdem das Schlüsselwort `LIKE`. Informationen dazu finden Sie hier: https://www.w3schools.com/sql/sql_like.asp.

Herauszufinden, wie genau `LIKE` funktioniert, ist Teil dieser Aufgabe. Beim Programmieren ist es ein immer wiederkehrendes Muster, die Syntax bestimmter Funktionalität herauszufinden. Es lohnt sich also, das hier zu üben.

Für das `LIKE`-Schlüsselwort kann man Platzhalter angeben (ähnlich wie in Regulären Ausdrücken). In SQL ist dies ein %-Zeichen. Weil dies in sqlite3 etwas seltsam gelöst ist, haben wir dies schon vorgegeben, in der Python-Variablen *queries*.