Elina Magamaeva/ Matrikelnummer: 2048934/ GitHub Name: elmag100

Dokumentation Restaurant-Bewertungsportal

Die folgenden Seiten sollen das Projekt zur Entwicklung eines Web-Portals zur Bewertung von Restaurants im Raum Hamburg hinsichtlich genutzter Technologien, Datenquellen und Konzepte beschreiben und erläutern. Im ersten Kapitel wird die Projektidee zusammengefasst, das zweite Kapitel beschreibt die verwendeten Software-Komponenten und das dritte Kapitel fasst den Konzeptions- und Entwicklungsprozess zusammen. Im letzten Kapitel wird, eingehend auf die Aufgabenstellungen der Einleitung, das Projekt reflektiert, bewertet und über Möglichkeiten der Erweiterung diskutiert.

Einleitung

Ziel des Projektes war es, ein Webportal zu entwickeln, das eine Übersicht aller Restaurants der Hamburger Innenstadt bereitstellt. Zusätzlich sollte basierend auf den Restaurantbeschreibungen eine grobe Kategorisierung der Restaurants vorgenommen werden. Jedes Restaurant hat eine Detailseite mit näheren Informationen wie Adresse, Beschreibung, Link zur Website. Auf dieser Detailseite sollten registrierte und eingeloggte Nutzer auch Kommentare und Bewertungen vornehmen können. Je Kategorie sollte man die Restaurants dann ihrer Bewertung nach sortiert einsehen können, wobei zu jedem Restaurant ein Mittelwert aller seiner Bewertungen angezeigt werden würde. Pro Kategorie sollten zunächst die Top-5 Restaurants vorgeschlagen werden und bei Wunsch dann alle weiteren abrufbar sein. Zusätzlich war eine Restaurantsuche nach Stichwort, Adresse, Stadt, Postleitzahl und Restaurantname vorgesehen.

Als Datenquelle diente ein Datensatz[1] aller Hamburger Restaurants, zu finden im Transparenzportal Hamburg[2] (ehem. Open Data Portal) mit etwa 160 Einträgen.

---

[1] http://suche.transparenz.hamburg.de/dataset/restaurants-in-hamburg

[2] http://transparenz.hamburg.de/open-data

Grundlagen

Im Folgenden werden die benutzten Sprachen, Frameworks und Tools genannt und in wenigen Worten vorgestellt.

Python

Der Webserver wurde komplett in Python[3] geschrieben. Die verwendete Version war Python 2.7.

Flask

Um die Entwicklung des Servers zu vereinfachen, wurde das Framework Flask[4] benutzt, welches durch integrierte Mechanismen für Routing, Datenbank-Interaktion und Templating viel Arbeit erspart. Da Flask über das pip-Tool installiert werden kann, ist auch das Setup des Servers, bzw. der Programmumgebung sehr simpel.

Jinja2

Jinja2[5] ist die verwendete Template-Engine, die Operationen wie Schleifen oder Variablentransformationen innerhalb von HTML-Templates ermöglicht. Jinja2 ist Teil des Flask-Frameworks.

SQLite

SQLite ist ein simples, dateibasiertes, relationales Datenbanksystem. Die meisten SQL-Befehle werden auch in SQLite ausgeführt, allerdings sind komplexere Operationen wie Joins, Sub-Queries oder Transaktionen nur eingeschränkt bis gar nicht möglich. Für den Zweck aber völlig ausreichend.

---

[3] https://www.python.org/

[4] http://flask.pocoo.org/

[5] http://jinja.pocoo.org/docs/dev/

HTML/CSS

Das Frontend-Layout und Styling wurde in HTML und CSS geschrieben. Dabei wurden die HTML5-Standards[6] weitestgehend eingehalten.

Bootstrap 3

Um eine uneingeschränkte Benutzung der Website, auch auf kleineren Geräten (Stichwort: Responsive Design), zu gewährleisten wurde das CSS-Framework Bootstrap[7] in der Version 3 verwendet. Dabei wird die Website in jeweils 12 Spalten unterteilt und alle Elemente können über relative Größen, die Vielfache von 1/12 der gesamten Seitenbreite sind je nach Bildschirmgröße passend skaliert und angeordnet werden. Im dritten Kapitel wird näher auf dieses Konzept eingegangen.

jQuery 3

Das Javascript-Frontend-Framework jQuery 3[8] ist nötig, damit Bootstrap auch auf kleineren Geräten richtig funktioniert, beispielsweise die dynamische Anpassung des Menüs. Ansonsten wurde kein Gebrauch von jQuery gemacht.

---

[6] https://www.w3.org/TR/html5/

[7] http://getbootstrap.com/

[8] https://jquery.com/

Projektbericht

Dieses Kapitel beschreibt den Projektverlauf, angefangen von der Konzeption über die Wahl der Software und Frameworks bis zur Implementierung. Dabei werden Bilder, Codeausschnitte und Diagramme zur Hilfe genommen.

Konzept

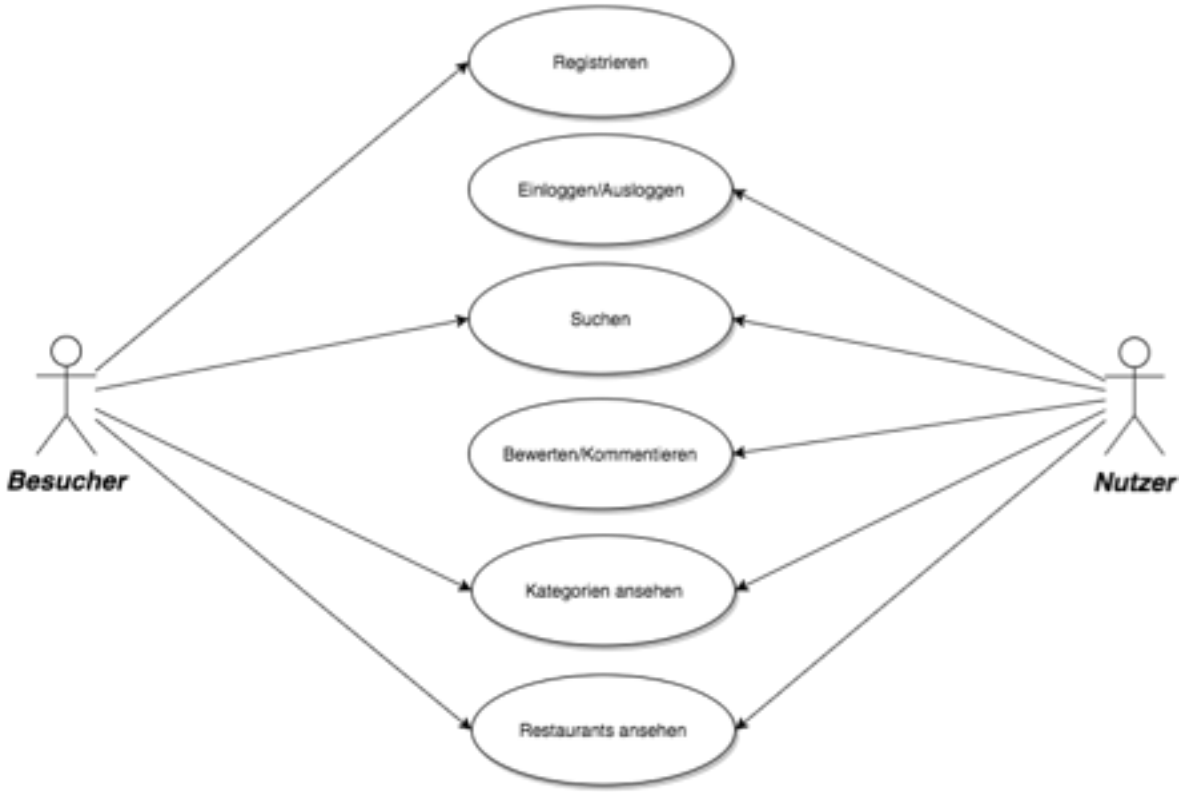
Das folgende Diagramm verbildlicht die Tabellen und die Relationen (ohne Kardinalitäten) der SQLite-Datenbank

Um die in der Einleitung beschriebene Datenarchitektur abzubilden reichen drei Tabellen aus:

restaurant - Enthält Datensätze zu einem Restaurant.
user - Wird bei der Registrierung eines Benutzers angelegt.
rating - Enthält eine Bewertung von 1-5, sowie einen Kommentar. Zusätzlich Verknüpfung zu dem entsprechenden Nutzer und Restaurant.

Auf allen ID-Spalten der Tabellen sollte in einer produktiven Umgebung ein Index angelegt sein, da dieser das Anfragen und Suchen auf Tabellen enorm beschleunigen kann. In dem gegebenen Umfang, hätte sich eine Indizierung jedoch keinen fühlbaren Unterschied gemacht, daher wurde darauf verzichtet.

Dieses Usecase-Diagramm zeigt die Privilegien/Funktionen der verschiedenen Nutzergruppen (Eingeloggter Benutzer, nicht eingeloggter Besucher), sowie die Seitenstruktur des Portals:



Man sieht, dass ein Besucher sich zunächst registrieren muss, bevor er sich einloggen und damit auch Kommentare verfassen und Bewertungen vornehmen kann.

Datenimport

Die Daten aus dem Transparenzportal Hamburg lagen im CSV-Format vor, daher musste diese zunächst in ein geeignetes Format gebracht werden, um in die SQLite-Datenbank importiert zu werden. Dazu kam der Python-Interne CSV-Reader zum Einsatz, um die Rohdaten Zeilenweise einzulesen. Dabei mussten Unterschiede im Zeichensatz der Python-Umgebung und der CSV-Datei mittels eines Kommentars

```
# -*- coding: utf-8 -*-
```

am Anfang des Programms angeglichen werden.

Ein typischer Datensatz sah in Rohform etwa so aus:

```
"Teheran","Adenauer Alle 70, 20097 Hamburg","Die persischen Spezialitäten im Teheran sind eine willkomene Abwechslung zur mediterranen und mitteleuropäischen Küche.","http://www.hamburg-tourism.de/erleben/essen-trinken/alle-restaurants-im-blick/teheran/"
```

Man sieht, dass beispielsweise die Adress-Spalte zusammengefasst Straße, Postleitzahl und Stadt des Restaurants enthält. Um einem Mindestgrad der Prinzipien der Datenbank-Normalisierung[9] gerecht zu werden, sollten zumindest Straße, Ort und Postleitzahl in separaten Spalten gespeichert werden. Mithilfe eines regulären Ausdrucks

```
(.*), ([0-9]{5}) (.*)
```

konnte man die Daten der Adresse leicht voneinander trennen

```
1.      [0-16]   `Adenauer Alle 70`
2.      [18-23   `20097`
        ]
3.      [24-31   `Hamburg`
        ]
```

und entsprechend in die Tabelle *restaurants* importieren.

[9] https://en.wikipedia.org/wiki/Database_normalization

Kategorisierung

Anschließend musste ein Konzept zur Kategorisierung der Restaurants erarbeitet werden. Durch mehrere Versuche haben sich die folgenden Bereiche als repräsentative Kategorien angeboten:

- Asiatisch
- Italientisch
- Café/Frühstück
- Spanisch
- Fisch & Meeresfrüchte
- Französisch

Die Kategorisierung selbst erfolgt durch die Indexierung der Restaurant-Beschreibungen im weitesten Sinne nach der Textwort-Methode, die an der folgenden Tabelle verdeutlicht wird.

Beschreibung enthält	**Kategorie**
'asia', 'asien', 'vietn', 'korea', 'fernost', 'fernöstlich', 'indisch', 'thai', 'japan', 'china', 'chinesisch'	Asiatisch
‘pizza’, ‘italien’	Italienisch
‘café’, ‘cafe’, ‘frühstück’, ‘kaffee’	Café/Frühstück
‘spanien’, ‘spanisch’, ‘tapas’	Spanisch
‘fisch’, ‘meeres’	Fisch & Meeresfrüchte
‘frankreich’, ‘französisch’	Französisch

Mithilfe dieser Kriterien wird versucht, alle Restaurants in eine Kategorie einzuordnen. Die einzelnen Stichworte werden auch als Prä- und Suffixe erkannt, sodass ebenfalls Textausschnitte wie

“koreanische Spezialitäten”

oder

“Thunfisch”

als relevant erkannt werden. Restaurants, die in keine der Kategorien fallen, werden in keiner weiteren Sonstiges/Andere-Kategorie abgelegt.

Aus technischer Sicht sind die Kategorien keine Entitäten im Datenbankmodell, denen Restaurants zugeordnet sind. Vielmehr manifestieren sie sich dadurch, dass eine Datenbankanfrage auf eine bestimmte Kategorie genau die Restaurants selektiert, in deren Beschreibungen sich Übereinstimmungen mit den entsprechenden Stichwortmengen finden, sie werden also erst zum Zeitpunkt der Anfrage festgelegt.

Restaurantsuche

Die Suche erlaubt es Nutzern, per Stichwort nach Restaurants zu suchen. Dabei werden entsprechend des Suchtextes die Namen, Beschreibungen und Adressen aller Restaurants abgeglichen und die Übereinstimmungen angezeigt. Die Suche funktioniert allerdings nicht mit mehreren Stichworten, da bei dem Abgleich der Suchtext nicht per Stichwort, sondern zusammenhängend ausgewertet wird.

Registrierung / Login und interaktive Funktionen

Es sollte im Portal die Möglichkeit geben, sich als Nutzer zu registrieren, um Bewertungen und Kommentare abgeben zu können. Dazu war ein einfaches Benutzer-Modell mit Name, Email und Passwort vorgesehen. Sowohl Email als auch Passwort werden in der Datenbank der Einfachheit halber und im Klartext gespeichert. Beim Login werden Name und Passwort mit den Einträgen in der User-Tabelle abgeglichen und bei Erfolg der Benutzer eingeloggt.

Dazu wird eine sogenannte Session angelegt, die auf dem Rechner des Nutzers eine temporäre Datei (Cookie) mit einer eindeutigen Nummer platziert, die der Browser bei jedem Aufruf der Seite mit an den Server schickt. Dieser kann dadurch erkennen, dass der Nutzer eingeloggt, die Session also aktiv ist. Sobald der Nutzer sich ausloggt, wird die Nummer des Cookies zurückgesetzt und die Session damit geschlossen.

Während der Session hat der Möglichkeit Kommentare mit einer Bewertung zu verfassen. Nach dem Absenden der Bewertung wird die ID des der Session zugeordneten Benutzers zusammen mit der Bewertung und der ID des Restaurants abgespeichert. Für der Übersicht der Kategorien wird über alle Bewertungen eines Restaurants ein Mittelwert errechnet, nach dem die Restaurants absteigend sortiert angezeigt werden.

Templates

Jede Unterseite des Portals hat ihre eigenes individuelles Template, welches im Ordner `/templates` liegt und Aussehen und Struktur der Seite beschreibt. Diese Templates sind größtenteils in HTML geschrieben, enthalten aber teilweise Ausdrücke, die von Jinja2 in der Funktion `render_template` interpretiert werden. Es gibt ein Template (`layout.html`), das als Vorlage für alle anderen dient und sich wiederholende Elemente, wie die Navigation oder den `<head>`-Bereich der Seite enthält, um redundanten Code in den anderen Templates zu verhindern. Mit dem Tag `{% block <name> %} … {% endblock %}` werden bestimmte Abschnitte der Seite (Navigation, Hauptinhalt, Footer) namentlich definiert.

Ein Template, welches von dem Haupttemplate erbt, wird mit `{% extends "layout.html" %}` eingeleitet. Danach kann über Ausdrücke der Art `{% block <name> %}{{ super( ) }}{% endblock %}` auf zuvor definierte Blöcke des darüberliegenden Templates referenziert werden.

Die zweite nennenswerte Besonderheit der Templates neben dem hierarchischen Aufbau ist die Verwendung des Bootstrap-Frameworks, dessen Konzept in den

Grundlagen bereits umrissen wurde. Bootstrap ist ein CSS-Framework, das auf vielen Webseiten Anwendung findet, die für sowohl Mobile- als auch Tablet- oder Desktop-Nutzung optimiert sind. Die Idee ist, die verfügbare Bildschirmbreite in 12 gleich große Kacheln[10] (columns) zu unterteilen, anhand derer man die Größe von Website-Elementen relativ (`1/12, 2/12, 6/12, 9/12, …, 12/12`) definieren kann. Zusätzlich kann man dies in Abhängigkeit von verschiedenen Bildschirmgrößen (`xs, sm, md, lg, xl`) macht. Möchte man beispielsweise, dass eine Kachel bei einem Handybildschirm (Größe `xs`) über die ganze Breite (`12/12`) angezeigt wird, gibt man dem HTML-Objekt die Klasse `col-xs-12`. Bei einem großen Bildschirm (Größe `lg`) hat man allerdings mehr Platz und zeigt es nur über die halbe Breite (`6/12`) an, sodass man noch etwas in der Reihe platzieren kann, also vergibt man zusätzlich noch die Klasse `col-lg-6`. Bootstrap übernimmt bei wechselnden Bildschirmbreiten automatisch die Anpassung der einzelnen Elemente entsprechend der `col-*-*` - Beschreibungen. Die untenstehende Grafik zeigt die Varianten des Kachel-Systems

 [11]

Im Projekt findet diese Methode auf fast jeder Seite Anwendung, beispielsweise bei der Übersicht der Kategorien oder der Restaurants.

Routing

Das Routing ist dank Nutzung des Flask-Frameworks ebenfalls sehr einfach. Die Seite hat folgende Unterseiten, die durch sog. Routen (englisch Routes) erreicht werden können:

- /

[10] http://getbootstrap.com/css/#grid

[11] http://www.tutorialrepublic.com/lib/images/grid-system-illustration.jpg

- /restaurants
- /detail/<*restaurant-id*>
- /rate
- /categories
- /category/<*category-name*>
- /login_form
- /login
- /register_form
- /register
- /logout
- /user/<*user-id*>
- /search_form

Jede dieser Routen korrespondiert mit einer Funktion, die im Python-Script definiert ist. Flask ermöglicht die Verknüpfung über eine einfache Zeile über der Funktionsdefinition:

```
@app.route('/user/<int:user_id>')
def user(user_id): # Informationen über einen Benutzer
      # [...]
```

Die wird automatisch nach dem angegebenen Schema ausgewertet und die User-ID wird der Funktion als Parameter übergeben. Innerhalb der Funktion steckt die Logik jeder Seite. In der Regel sind das Überprüfungen von Parameter, eine oder mehrere SQL-Abfragen und die Weitergabe von Parametern an das Template mithilfe der Funktion `render_template`.

Schluss

Dieses Kapitel, soll das Projekt reflektieren, Probleme ansprechen die während der Entwicklung aufgetreten sind, Ansätze für mögliche Erweiterungen geben und die Aufgabenstellungen aus der Einleitungen kommentieren.

Probleme bei der Entwicklung

Im Laufe der Entwicklung sind einige kleinere Probleme sowohl in technischer, als auch konzeptioneller Hinsicht aufgetreten.

Beispielsweise war die Inkonsistenz in der Text-Codierung zwischen Datenquelle und Python zunächst nicht als solche zu erkennen und die Lösung war ebenfalls nicht offensichtlich. Ein ähnliches Problem trat auch beim Import in die Datenbank auf. Dazu mussten Texte, die Umlaute und andere Sonderzeichen enthielten erst dekodiert werden, bevor man sie übergeben konnte.

Inhaltliche Fragen stellten sich beispielsweise bei der Kategorisierung der Restaurants. So war es zunächst ein Testen verschiedener Kombinationen von Stichworten, bis man eine hinreichend große Menge von Restaurants erhielt, die man einer eigenen Kategorie zuordnen kann. Anschließend musste diese Zuordnung auch implementiert werden. Diese hätte man auch im Datenbankschema festlegen können, allerdings wurde die beschriebene on-the-fly-Lösung dem Umfang des Projektes mehr als gerecht. Lediglich eine kleine strukturelle Schwäche ist dieser Lösung geschuldet. Auf der Detailseite eines Restaurants gibt es einen “zurück”-Button, der den Nutzer auf die vorherige Seite zurückbringt. Verfasst ein Nutzer einen Kommentar und/oder eine Bewertung, wird die Seite anschließend neu geladen. Dadurch erzeugt man aufgrund der Art der Implementierung eine zyklische Referenz und der Button verliert seine Funktion. Eine andere Implementierung war leider nicht möglich, da ein Restaurant aufgrund der Datenbank-Struktur keine feste Kategorie hatte und somit eine rückwärtige Referenz von der Detailseite zur entsprechenden Kategorie-Übersicht nicht möglich war. Funktional gesehen nicht richtig, aber möglich wäre da die Verlinkung des “zurück”-Button zur der Übersichtseite. Da man aber über das Menü bereits zu Übersicht navigieren kann ist diese Option überflüssig.

Sicherheitsprobleme & Erweiterungsmöglichkeiten

Die Anwendung ist nicht dafür entwickelt worden, um im größeren Stil performant und sicher zu sein. Beispielsweise sollte man Passwörter grundsätzlich nicht in Klartext abspeichern, da potenzielle Hacker ein leichtes Spiel hätten, Benutzerdaten zu klauen und zu missbrauchen. Daneben ist auch die verwendete SQLite-Datenbank für größere Datenmengen konzeptionell nicht die richtige Wahl. Auch die Implementierung der Session lässt es relativ einfach zu, sich unter falschem Namen anzumelden.
Aus den angesprochenen Problemen ergibt sich bereits eine Reihe von Erweiterungsmöglichkeiten. Man könnte im Sinne der Skalierbarkeit die SQLite-Datenbank durch eine MySQL-Datenbank[12] ersetzen, die auch bei größeren Datenmengen noch hohe Geschwindigkeit aufweist. Das Datenbankschema kann man dahingehend erweitern, dass Kategorien eine eigene Entität sind, die in einer Relation mit den zugehörigen Restaurants stehen.

Die Suche erkennt nur Textausschnitte, die exakt mit dem Suchtext übereinstimmen. Sie könnte dahingehend erweitert werden, dass Stichworte aus dem Suchtext separiert werden und die Suche einzeln für jedes Wort durchgeführt wird. Anschließend werden die Ergebnismengen vereint und man kann entsprechend der absoluten Vorkommen in den einzelnen Teilsuchen ein Relevanz-Ranking erstellen. Anknüpfend an den Vorschlag, die Kategorien in das DB-Modell zu integrieren, kann man ebenfalls eine Kategoriensuche implementieren.

Unter Zuhilfenahme anderer öffentlicher Datenquellen kann man die Anwendung um weitere Städte erweitern. Es gibt ebenfalls öffentliche Datenschnittstellen, mit denen man das heutige Wetter[13] oder eine Ansichtskarte der Stadt[14] in die Anwendung integrieren kann.

---

[12] https://www.mysql.de/

[13] http://openweathermap.org/api

[14] https://developers.google.com/maps/documentation/javascript/?hl=de

Appendix

Hinweise zur Installation / Ausführung

Nach dem Entpacken des Archives ist keine weitere Installation der Anwendung notwendig. Zur Einrichtung und Ausführung wird allerdings Python 2.7 und pip benötigt.

Danach werden folgende Befehle in einer Konsole nacheinander ausgeführt:

1. Mittels ~ `cd <pfad>` in den Installationsordner navigieren
2. Falls das Flask Framework noch nicht installiert ist
    a. ~ `pip install flask` und warten bis Flask installiert ist
3. Falls das Flask Framework bereits installiert ist
    a. ~ `pip install flask --upgrade` um sicherzustellen, dass die Version aktuell ist
4. Nun muss Flask in einer Umgebungsvariable mitgeteilt werden, welche Anwendung ausgeführt werden soll: ~ `export FLASK_APP=application.py` Hierbei ist es wichtig, dass die Konsole im gleichen Ordner, wie die Datei `application.py` ist.
5. Im letzten Schritt muss der Server nun gestartet werden. Je nach Installationsumgebung reicht der Befehl ~ `flask run` dafür aus, sicher funktioniert jedoch ~ `python -m flask run`.
6. Wenn alles geklappt hat, sollte in der Konsole folgender Text bestätigen, dass der Server läuft:

```
* Serving Flask app "application"
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

7. Nun kann die Anwendung in einem beliebigen Browser unter der Adressse http://127.0.0.1:5000/ aufgerufen werden.
8. Wer die Registrierung überspringen möchte kann sich mit dem Benutzernamen *Tester* und dem Passwort *123* anmelden.
9. Viel Spaß mit der Anwendung!