IWGS1_WS1920_Zettel12

July 7, 2020

1 Informatische Werkzeuge in den Geistes- und Sozialwissenschaften I

1.1 Hausaufgabe 12 (Projekt.2, 100 Punkte)

Erschienen: 25.01.2020
Abgabe bis: 02.02.2020

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.

1.1.1 Zusammenfassung Projekt

Im Übungszettel 10 haben Sie bereits damit begonnen die notwendigen Komponenten einer Web Application für ein *Choose-Your-Own-Adventure*-Abenteuer, ein so genanntes Spielbuch , zu entwickeln. Zettel 10 hat sich hauptsächlich mit dem Auslesen der nötigen Informationen aus Dateien in einem geeigneten Format beschäftigt. Es verbleibt die Funktionalität, die Informationen in einer Webapp anzuzeigen, sodass Interaktion ermöglicht wird.

Dieser Übungszettel bildet den Abschluss für das Projekt. Sie bekommen von uns Story-Dateien für das interaktive Abenteuer *"Beatrice Beispiel und die Magische Musterlösung"* gestellt, welche Sie natürlich beliebig ersetzen oder erweitern können (siehe auch Aufgabe P2.5). Ihre WebApp sollte unabhängig vom konkreten Inhalt dieser Dateien funktionieren.

Für die Aufgaben auf diesem Zettel wird abermals stark Gebrauch von der Bibliothek `bottle` gemacht. Wenn Sie damit noch nicht vertraut sind, können Sie Zettel 11 (Musterlösung verfügbar ab 27.01.2020), das verfügbare `bottle`-Tutorial oder Ihre Tutoren konsultieren.

1.1.2 Projekt P2.1 (Startseite, 10 Punkte)

Der erste Schritt zur WebApp ist die Startseite, auf der Besuchende landen und darüber informiert werden, was sie erwartet. Da diese Seite statisch (d.h. immer gleich) ist, ist die Route für die WebApp außerdem relativ einfach zu schreiben.

Wir wollen ermöglichen, als unterschiedliche Charaktere zu spielen (auch wenn das nur minimale Auswirkungen auf die Beispielgeschichte hat, vgl. Aufgabe P1.5). Momentan haben Sie noch nicht die Techniken, den Nutzer selbst einen Namen eingeben zu lassen, der dann entsprechend verwendet wird, also werden wir drei Möglichkeiten stellen, die sich mit einfachen Links realisieren lassen.

Aufgabe: Fügen Sie der Datei `cyoa_server.py` eine Route `/` hinzu, die eine Startseite mit einem kurzen erklärenden Text zurück gibt.
Sie soll außerdem drei Links enthalten, mit denen die Geschichte begonnen werden kann. Ein Link soll auf `/story/Beatrice/start` verlinken (für Menschen, die als Beatrice Beispiel spielen wollen), einer auf `/story/Max/start` (für Max Mustermann) und der dritte auf `/story/Nova/start` (für Nova Neuling).

1.1.3 Projekt P2.2 (Template für Kapitel, 40 Punkte)

Der nächste Schritt, nachdem die Information ausgelesen wurde, ist, sie zu präsentieren. Wir werden uns den Fakt zunutze machen, dass (abgesehen von der Startseite) alle anderen Seiten in unserer Spielbuch-WebApp die gleiche Struktur haben (Titel, Inhalte und Handlungsoptionen). Das bedeutet, dass wir eine *Template-Datei* schreiben und für *alle* Kapitel verwenden können, indem wir lediglich den angezeigten Text ändern. Das erspart uns im Vergleich zur Alternative (einzelne HTML-Seiten für jedes einzelne Kapitel schreiben oder generieren) nicht nur enorm viel Arbeit, es erleichtert auch die spätere Wiederverwendbarkeit und Erweiterung der WebApp.

Die `bottle`-Bibliothek stellt uns für diese Zwecke die SimpleTemplate Engine (oder kurz: `stpl`) zur Verfügung. Der größte Vorteil an dieser Engine ist, dass sie uns erlaubt, mit Python-Code *programmatisch* die HTML-Seite aufzubauen. Die wichtigsten Arten, dies zu tun, sind hier kurz angerissen und werdem im Tutorial ausführlicher behandelt. Das Prinzip ist jedoch immer das gleiche. Beim Aufruf einer Template-Datei (die zunächst Mal identisch zu einer normalen HTML-Datei sein kann) können Sie mehrere Python-Variablen übergeben, deren Inhalt dann Teil der HTML-Datei werden kann.

- Setzen Sie einen Variablennamen wie etwa `title` in der Template-Datei in geschweifte Klammern (`{{title}}`), wird der Inhalt der Variable beim Aufruf dort als Text eingefügt.
- Mit der Syntax `%` können Sie eine Zeile Python-Code beginnen.
- Mit der Syntax `<%` ... `%>` können Sie mehrere Zeilen Python-Code einbinden.

Mit diesen Werkzeugen können Sie in der Template-Datei quasi frei programmieren. Weil nun HTML aber (anders als Python) keine Rücksicht auf Whitespace nimmt, gibt es einen großen Unterschied: Code-Blöcke (wie If-Abfragen, Schleifen o.Ä.) können nicht mehr allein durch Einrückung beendet werden, sondern benötigen ein `end` keyword.

Dazu ein Beispiel! Angenommen Sie rufen folgende Template-Datei `example.tpl` auf:

```
<html>
    <body>
    <h1>{{title}}</h1>
    % length = len(animals)
    Diese {{length}} Tiere habe ich sehr gern:
    <ul>
        % for animal in animals:
        <li>{{animal}}</li>
```

```
        % end
    </ul>
    <%
        # This is an entire block of python code!
        if length > 10:
            footer = "Das sind ziemlich viele Tiere!"
        else:
            footer = "Auf Wiedersehen!"
    %>
    {{footer}}
    </body>
</html>
```

Ein typischer Aufruf aus Python wäre vielleicht der folgende:

```
from bottle import template

@route('/example')
def example():
    title = "Meine Lieblingstiere!"
    favourites = ["Faultiere","Haie","Rabenvögel","Fangschreckenkrebse"]
    return template('example.tpl', title=title, animals=favourites)
```

Wie erwartet ist der Quellcode der resultierenden HTML-Datei folgender (Python-Code wird ausgeführt, aber erscheint selbst nicht im fertigen HTML):

```
<html>
    <body>
    <h1>Meine Lieblingstiere!</h1>
    Diese 4 Tiere habe ich sehr gern:
    <ul>
        <li>Faultiere</li>
        <li>Haie</li>
        <li>Rabenvögel</li>
        <li>Fangschreckenkrebse</li>
    </ul>
    Auf Wiedersehen!
    </body>
</html>
```

Das gleische Schema sollen Sie jetzt für Ihre WebApp umsetzen. Dabei hat es mehrere Vorteile, auch HTML-Klassen zu nutzen. Manchen Tags in HTML, zum Beispiel `<div>` können Klassen übergeben werden, um mehrere Elemente eines Tags später voneinander unterscheiden zu können. Dies hat erst einmal nichts mit *Klassen* im Sinner von objektorientierter Programmierung zu tun. Hier ein Beispiel:

```
<div class="foobar">Normaler Text</div>
```

Aufgabe: Schreiben Sie eine HTML-Template-Datei `chapter.tpl`, die den Titel, den Text und die Handlungsoptionen, die ihr übergeben werden, sinnvoll einbindet. Für den Titel soll eine `<h1>`-Überschrift angelegt werden, der Text soll in einem `<div>`-Tag der `contents` angezeigt werden und

die Handlungsoptionen in einem `<div>`-Tag der Klasse `options`. Dies wird relevant werden, sobald wir in P2.4 CSS einbinden.

1.1.4 Projekt P2.3 (Routing für Kapitel, 20 Punkte)

Der letzte kritische Schritt zur funktionierenden WebApp ist noch eine Route, die die einzelnen Kapitel ausliest und die Templatedatei aus P2.2 mit den korrekten Werten aufruft.

Sie können für das Auslesen auf die Funktionen `chap.get_title()`, `chap.get_options()` und `chap.get_contents()` zugreifen, die auf der Musterlösung von Zettel 10 basieren und von der Datei `cyoa_server.py` bereits importiert werden. Alle diese Funktionen nehmen einen Dateinamen entgegen. Dabei ist zu beachten, dass der Name des Unterorders in diesem Dateinamen enthalten sein muss. Übergeben Sie also zum Beispiel `"./story/start.txt"` und nicht einfach `"start.txt"`.

Bedenken Sie dabei, dass Sie im Rückgabewert `chap.get_contents()` (der zunächst aus einer Liste von Strings besteht) noch wie auf Zettel 10 den String `"NAME"` durch den Inhalt der übergebenen Variable `name` ersetzen müssen.

Aufgabe: Fügen Sie der Datei `cyoa_server.py` eine Route `/<name>/<chapter>` hinzu. Diese soll mit den vorgestellten Funktionen die relevanten Informationen aus der Datei `story/<chapter>.txt` auslesen und damit Ihre Template-Datei aufrufen. Wenn keine entsprechende Datei gefunden wird, soll ein Text angezeigt werden, der zurück auf die Startseite verweist.

1.1.5 Projekt P2.4 (Stylesheet, 15 Punkte)

Die Webseiten, wie sie jetzt aktuell angezeigt werden (schwarzer Text auf weißem Grund, keinerlei interessante Formatierung), sind noch etwas unaufregend und unübersichtlich. Besser für das sogen. User Experience wäre es, wenn schon optisch eine Trennung zwischen z.B. Kapiteltext und Handlungsoptionen bestehen würde. Eine Möglichkeit, HTML-Seiten darart aufzufrischen sind *Cascading Style Sheets* (CSS). Das sind externe Dateien, die aus der HTML-Datei verlinkt werden können, mit Spezifikationen wie bestimmte HTML-Elemente angezeigt werden sollen.

Um ein Stylesheet in das HTML, was Ihr Server liefert, einzubingen, ergänzen Sie einfach das `<head>`-Tag wie folgt:

```
<head>
    <!-- Andere head-Elemente hier... -->
    <link rel="stylesheet" href='/static/story.css'>
</head>
```

Damit an `/static/story.css` auch tatsächlich eine CSS-Datei gefunden wird, benötigen Sie einerseits die Datei `story.css` und andererseits die folgende Route in Ihrem Server:

```
from bottle import static_file

@route('/static/<filepath>')
def server_static(filepath):
    return static_file(filepath, root='.') # "." => put file into same directory as cyoa_serve
```

Hinweis: Wenn Sie sich noch unsicher mit CSS fühlen oder nicht weiter kommen, können Sie (neben Tutorium, Slack, Forum und Emails an die Tutoren) im Internet zahlreiche gute Quellen für diese weit verbreitete Technologie finden. Wie auch schon zu Python und HTML , hat die Website `w3schools.com` auch für CSS ein hilfreiches Tutorial inklusive kleinerer Übungen.

Aufgabe: Fertigen Sie eine CSS-Datei `story.css` an, mit der Sie das Aussehen der HTML-Seiten dieses Projektes verändern können. Diese Datei sollte mindestens eine neue Hintergrundfarbe setzen und die `div`-Klassen `contents` und `options` in irgendeiner Weise verändern (wie genau ist Ihnen überlassen). Binden Sie diese sowohl auf Ihrer Startseite als auch in Ihrem Template für Kapitel ein.

1.1.6 Projekt P2.5 (Erweiterung der Story, 15 Punkte)

Die Geschichte, so wie Sie sie erhalten, ist noch nicht vollständig. Aktuell ist noch offen, was der Hauptrolle passiert, wenn Sie sich dazu entscheidet mit jemandem aus dem Fachbereich Okkultismuskunde zu der magischen Musterlösung zu konsultieren. Der Link von `uni-take.txt` nach `talk-scientist.txt` führt ins Leere, weil das Ziel (noch) nicht existiert.

Ihre Aufgabe ist es, das zu ändern, und für diese Abzweigung Kapitel hinzuzufügen. Der Inhalt dieser ist Ihnen komplett frei gestellt. Für diese Aufgabe müssen Sie kein Python programmieren, sondern Textdateien im passenden Format anlegen und mit Inhalt füllen.

Aufgabe: Erweitern Sie die Geschichte duch hinzufügen von `talk-scientist.txt` und mindestens einer weiteren Kapitel-Datei. `talk-scientist` soll dabei einen Titel und mindestens zwei Optionen enthalten und entsprechend verlinken. Mit diesen zusätzlichen Kapiteln soll die Geschichte keine dieser ”offenen Enden” mehr enthalten.