JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

A07: Python Webframeworks

Web2py

Softwareentwicklung
5BHIT 2017/18

Maximilian Müller, Martin Wölfer

Version 0.1
Begonnen am 10. Januar 2018
Beendet am 19. Januar 2018

Note:
Betreuer: RAFM & DOLD

Inhaltsverzeichnis

1 Aufgabenstellung

Suche dir mit einem/einer Partner/in ein Python Webframework aus und präsentiere deine Lösung!

1.1 Grundanforderungen(70%)

Hier werden die zu erwerbenden Kompetenzen und deren Deskriptoren beschrieben. Diese werden von den unterweisenden Lehrkräften vorgestellt.

Dies kann natürlich auch durch eine Aufzählung erfolgen:

- Installiere und konfiguriere eines der präsentierten Frameworks
- Überlege dir einen sinnvollen Anwendungsfall für das Framework und erstelle eine passende Datenbank dazu
- Erstelle eine simple Seite, welche Datensätze aus einer Datenbank anzeigt
- Protokolliere deine Vorgehensweise, aufgetretene Probleme etc.

1.2 Erweiterungen(30%)

- Verwende ein ansprechendes Design
- Das Bearbeiten von Datensätzen
- Das Löschen von Datensätzen
- Das Erstellen von neuen Datensätzen

1.3 Abgabe

Protokoll inkl. Arbeitsaufwand, Screenshots und Beschreibungen

2 Installation

Die Installation von web2py kann in 10 Sekunden erfolgen. Es wird lediglich ein `.zip` heruntergeladen von der offiziellen Seite (→ **Source Code**).

Der Source Code wird anschließend, und in diesem Ordner wird mit **python2** das File `web2py.py` ausgeführt:

```
1 cd web2py
  python2 web2py.py
```

Anschließend öffnet sich ein Fenster, bei welchem man angeben kann unter welcher Adresse man die Website und die Datenbank laufen haben will, in dem Anwendungsbeispiel **localhost**. Weiter kann ein Port ausgewählt werden, über welchem man web2py aufruft, standardmäßig wird der Port **8000** gewählt.

Zum Schluss muss noch ein Administrator Passwort gewählt, mit diesem verwaltet man jene Seite, welche momentan mit diesem Server aufgesetzt wird. Wichtig ist, jedes mal wenn der Server neu gestartet wird (d.h. `web2py.py` ausgeführt wird) wird nach einem neuen Passwort gefragt.

Abbildung 1: web2py grafisches interface

Nun kann **localhost:8000** aufgerufen werden, und man wird von der Standardseite empfangen.

3 Generell web2py

web2py ist ein sehr umfangreiches framework, und das merkt man sofort sobald man die defaultseite aufruft. Es gibt sehr viele Optionen und man ist anfänglicher weise leicht überwältigt. Bevor

überhaupt daran gedacht werden kann, eine app zu schreiben, muss sich web2py in der Theorie angeschaut werden.

3.1 Aufbau

Alle Applikationen in web2py sind mit dem **MVC-Modell**aufgebaut. Weiters ist wichtig, dass web2py einen Programmierstil voraussetzt, welcher **"coding-by-convention"** genannt wird. Dies bedeutet, dass der Programmierer sich immer an die Konventionen halten sollte, aber wenn nicht muss er dies explizit angeben. Durch diesen Stil wird dem Softwareentwickler sehr viel arbeit abgenommen weil sehr viele Beziehungen oder sogar Daten automatisch erzeugt werden.

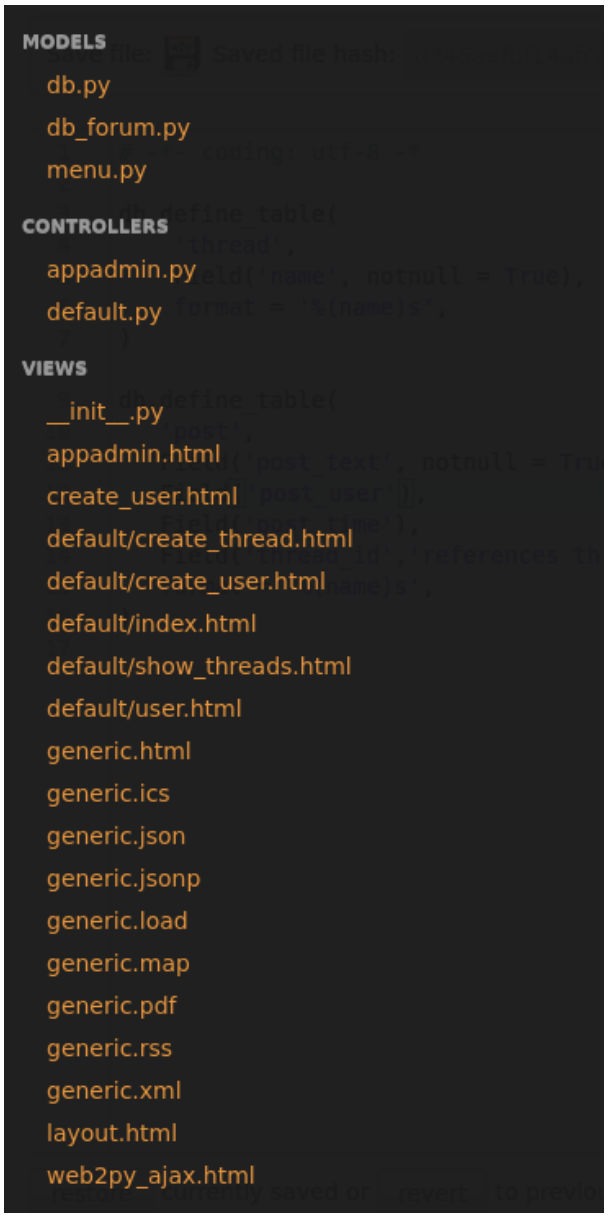

Abbildung 2: Grundsätzlicher Aufbau von web2py

3.2 Model

In web2py stellt das Model die Datenbank dar. Unter dem Bereich **Model** können `.py` Files erzeugt werden, in welchem die einzelnen Tabellen definiert werden können. Hierbei wird nicht normales `SQL` geschrieben, sondern es wird die Datenbank Abstraktionsschicht von web2py verwendet, auch **DAL** (**D**atabase **A**bstraction **Layer**) genannt. Dadurch können **CRUD** Anweisungen leichter ausgeführt werden.

Natürlich können auch hier, wie in SQL, constraints definiert werden um die Tabellen konsistent zu halten. Wichtig ist nur, dass wie in Django keine **Primary Keys** definiert werden müssen, da jede Tabelle automatisch ein Feld **Id** hat, welches automatisch inkrementiert wird mit jeder neuen Reihe.

Eine Tabelle besitzt folgende Optionen definiert zu werden:

```
db.define_table('person',
    Field('name'),
    id=id,
    rname=None,
    redefine=True
    common_filter,
    fake_migrate,
    fields,
    format,
    migrate,
    on_define,
    plural,
    polymodel,
    primarykey,
    redefine,
    sequence_name,
    singular,
    table_class,
    trigger_name)
```

Ein Feld, hat folgende Signatur

```
Field(fieldname, type='string', length=None, default=None,
    required=False, requires='<default>',
    ondelete='CASCADE', notnull=False, unique=False,
    uploadfield=True, widget=None, label=None, comment=None,
    writable=True, readable=True, update=None, authorize=None,
    autodelete=False, represent=None, compute=None,
    uploadfolder=None, uploadseparate=None, uploadfs=None, rname=None)
```

3.3 Controller

In web2py werden vom Controller die User-eingaben verwaltet und weitergeleitet - meistens an die View. Wichtig ist, jede Funktion welche im Controller definiert wird, kann im Browser aufgerufen werden, wahrscheinlich über eine REST-Schnittstelle.

Bsp:

Folgende Funktion wird in `default.py` definiert:

```
1   def hello_world():
        return "Hello World"
```

Liefert folgendes Ergebnis:

Abbildung 3: localhost:8000/app_name/default/hello_world

3.4 View

Als Model wird in web2py die Darstellung verstanden. Hierbei werden `.html` Dateien erstellt, welche aber durchaus Logik enthalten können, vor allem in Kommunikation mit dem Controller. Die Verwendung von HTML-Dateien in web2py erinnert stark an die Verwendung von `.xhtml` in Java.

Mit doppelt geschwungenen Klammern können Variablen aufgerufen werden, welche vom Controller in einem `dict` zurückgegeben wurden. Wie in Java, ist es auch hier möglich mit `extends` von einem HTML-File zu erben, um ein bestimmtes Layout über alle Seiten zu besitzen.

Bsp:

Folgende Funktion wird in `default.py` definiert:

```
2   def hello_world():
        return dict(msg="hello_world")
```

Und in `default/hello_world.html` wird folgendes definiert:

```
2   <h1>Vom Controller wurde folgendene Nachricht uebergeben: </h1>
    {{=msg}}
```

Und dies liefert folgendes Ergebnis:

4 Anwendungsfall

Idee ist es, eine sehr vereinfachte Form eines Forums zu programmieren.

4.1 Datenbank(Model)

Zuerst wurde sich ein einfaches Datenbankmodell überlegt, um die Daten abzuspeichern.

4.1.1 Modellierung

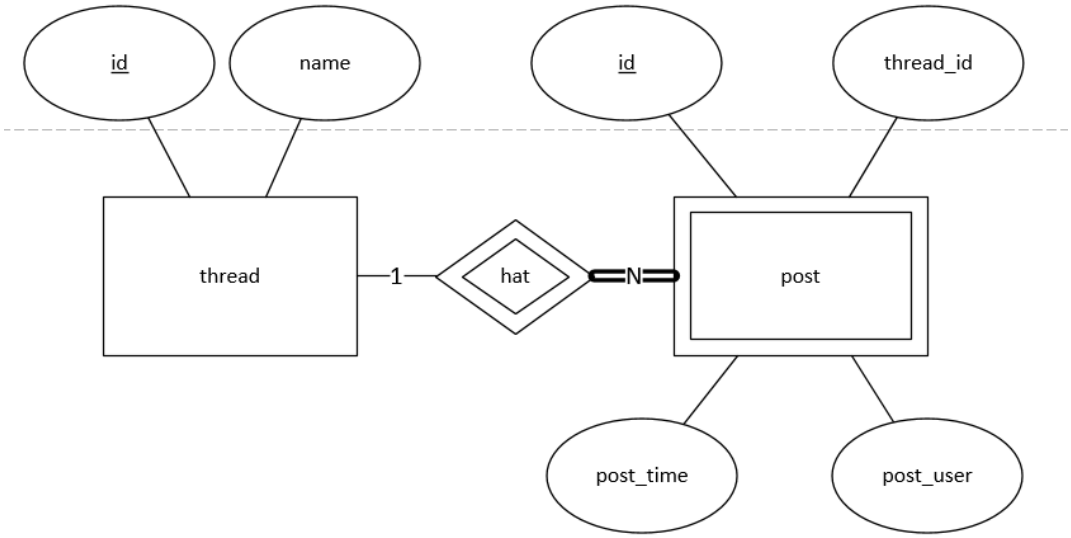

Abbildung 4: ERD-Diagramm der Datenbank

4.1.2 Umsetzung

```
db.define_table(
    'thread',
    Field('name', notnull = True),
    format = '%(name)s',
)

db.define_table(
    'post',
    Field('post_text', notnull = True),
    Field('post_user'),
    Field('post_time'),
    Field('thread_id','references thread'),
    format = '%(name)s',
)
```

4.2 Controller

Der Controller nimmt Daten aus der Datenbank und übergibt diese dem View.

In index() wird die Liste der Threads, deren Posts und ein Button zum erstellen eines neuen Threads übergeben.

```
def index():
    threads = db(db.thread).select()
    records = SQLFORM.grid(db.thread, db.thread(), deletable=True)
```



```
    form = SQLFORM(db.thread).process()
    form.element(_type='submit')['_value'] = T("Thread erstellen")
    posts = db().select(db.post.ALL)
    if form.accepted:
        redirect(URL("index.html"), client_side=True)
        response.flash = T("Neuer Thread wurde erstellt")
    return dict(threads=threads, records=records,form=form, posts=posts)
```

Wenn create_post() aufgerufen wird, fügt es der Datenbank zum jeweiligen Thread den neuen Post hinzu.

```
def create_post():
    thread = request.args[0]
    back_button = A(T('Back'), _href=URL('default', 'index', user_signature=True), _class='btn')
    form=SQLFORM(db.post, fields=['post_text','post_user'])
    form.vars.thread_id = thread
    form.process()
    return dict(form=form,thread=thread,back_button=back_button)
```

delete_post() löscht einen bestimmten Post aus einem Thread.

```
def delete_post():
    post = request.args[0]
    db(db.post.id==post).delete()
    redirect(URL("index.html"), client_side=True)
```

update_post() lässt einen einen Post bearbeiten.

```
def update_post():
    post = request.args[0]
    record = db.post(post)
    form=SQLFORM(db.post, record, fields=['post_text'])
    form.vars.thread_id = request.args[2]
    form.vars.post_user = request.args[1]
    form.process()
    db(db.post.id==request.args[3]).delete()
    back_button = A(T('Back'), _href=URL('default', 'index', user_signature=True), _class='btn')
    return dict(form=form, back_button=back_button)
```

delete_thread() löscht den spezifizierten Thread.

```
def delete_thread():
    thread = request.args[0]
    db(db.thread.id==thread).delete()
    redirect(URL("index.html"), client_side=True)
```

4.3 View

update_post.html stellt die Daten dar, die es von update_post() bekommt (Formular und button).

```
{{extend 'layout.html'}}
<h1>Einen Post updaten</h1>
{{=form}}
{{=back_button}}
```

index.html stellt die Threads und deren Posts Tabelarisch da. Die Daten dazu kommen aus index().

```
{{extend 'layout.html'}}

{{block header}}
<center style="background-color: #333; color:white; padding:30px">
  <h1>{{=request.application}}
</center>
```



```
{{end}}

{{=form}}
{{for thread in threads:}}
<table border="1" cellspacing="20" cellpadding="20" >
        <tr>
            <th>{{=thread.name}}</th>
            <th>User</th>
            <th>Time posted</th>
            <th><a class="btn" href="{{=URL('default','delete_thread', args=[thread.id])}}">{{=T('Thread
                loeschen')}}</a></th>
        </tr>
    {{for post in posts:}}
        {{if post.thread_id == thread.id:}}
        <tr>
            <td>{{=post.post_text}}</td>
            <td>{{=post.post_user}}</td>
            <td>{{=post.post_time}}</td>
            <td><a class="btn btn-secondary" href="{{=URL('default','delete_post', args=[post.id])}}">
                {{=T('Post loeschen')}}</a></td>
            <td><a class="btn btn-secondary" href="{{=URL('default','update_post', args=[post, post.
                post_user, post.thread_id, post.id])}}">{{=T('Post updaten')}}</a></td>
        </tr>
        {{pass}}
    {{pass}}
        <tr>
            <td><a class="btn btn-secondary" href="{{=URL('default','create_post', args=[thread.id])}}">
                {{=T('Post hinzufuegen')}}</a></td>
        </tr>
</table>
    </br>
{{pass}}
```

create_post.html lässt dich einen neuen Post erstellen.

```
{{extend 'layout.html'}}
<h1>Einen Post hinzufuegen</h1>
{{=form}}
{{=back_button}}
```

5 Ergebnisse

6 Probleme

6.1 Anfangs überwältigend

Wie bereits erwähnt, war es am Anfang sehr viel auf einmal. Besonders weil per ”Coding-by-convention” programmiert wird, ist es anfänglich schwer da sehr viel automatisiert ist und man nicht genau weiß ”was” automatisch generiert wird und was man händisch machen soll.

6.2 Seite neu laden

Ein weiteres Problem welches aufgetreten ist, nachdem ein neuer thread oder post erzeugt wird, soll die Seite neue geladen werden damit die neu eingesetzten Datensätze angezeigt werden. Dies wurde nach langer Recherche umgesetzt, indem schlicht und einfach mit `redirect` auf die momentane Seite ”umgeleitet” wird.

Literatur

Tabellenverzeichnis

Listings

Abbildungsverzeichnis

