Hochschule Augsburg University of Applied Sciences

Studienarbeit

Studienrichtung Informatik

Jorge Andrés Cuartas Monroy
Moritz Schächterle
Dominik Heimstädt

WM-Tippspiel

Fakultät für Informatik
Telefon: +49 821 5586-3450
Fax: +49 821 5586-3499

Hochschule Augsburg
University of Applied Sciences
Baumgartnerstraße 16
D 86161 Augsburg

Telefon +49 821 5586-0
Fax +49 821 5586-3222
http://www.hs-augsburg.de
poststelle@hs-augsburg.de

Dr. Helmut Merz
Internet Programmierung mit Python
Abgabe der Arbeit: 1.07.2010

Inhaltsverzeichnis

1 Aufgabenstellung

1.1 Einführung

Im Rahmen der Vorlesung „Internetprogrammierung mit Python“ ist eine Studienarbeit zu erstellen. Aus mehreren zur Auswahl stehenden Arbeiten ist die Wahl auf „Fußball-Tippspiel“ gefallen. Die Aufgabe darf im Team bearbeitet werden mit maximal drei Gruppenmitgliedern. Aus aktuellem Anlaß wird die Anwendug ein Fußball-WM Tippspiel mit der Möglichkeit auf die Begegnungen zu tippen und für richtige Tipps, Differenz oder Tendenz Punkte zu erhalten. Die Anwendung soll darüberhinaus noch ermöglichen einzusehen wieviele Punkte der User momentan hat und auf welchen Platz er steht.

Bei der Überlegung zur Wahl geeigneter Werkzueuge zur Erstellung der Anwendung, standen unter anderem Zope, Turbogears, Pylons und Django in der engeren Auswahl.

Für die Auswahl geeigneter Werkzeuge sind folgende Punkte wichtig. Arbeiten mit bekannten Techniken/Komponenten wie Python, mySQl, ORM[1], HTML, JavaScript und Apache. Darüberhinaus sollte die Einarbeitungszeit nicht zu groß sein, da aus Erfahrung, das Einarbeiten in neue Frameworks zeitaufwändig ist. Somit ist eine gute Dokumentation ein weiterer wichtiger Punkt für die Auswahl.

Mit allen oben erwähnten Frameworks kommt man sicher schnell und einfach ans Ziel. Letzenlich fiel die Wahl auf Django, da dieses Framework eine recht schnelle Entwicklung für unsere Studienarbeit verspricht. Unter anderem sticht die einfache Userverwaltung, der schnell zu konfigurierbare Adminbereich der zur Verfügunggestellt wird, der OR-Mapper, das Templatesystem und der DRY: Don’t repeat yourself Ansatz.

[1] Objekt-Relational Mapping

2 Web Framework

2.1 Django

Das Django Webframework eignet sich für die Erstellung von Webanwendungen. Es folgt dem MVC[1] Muster. Wobei bei Django die Modelle der Anwendung, die Objekte mit denen Django arbeitet mit Hilfe von OR-Mapping in entweder mySQL, PostgreSQL, Oracle oder SQLite gespeichert werden können. So wird die Datenpersistenz der Anwendung gewährleistet. Für die View sind Templates zuständig, die mit Hilfe einer eigener Templatesprache konfiguriert werden. Die Schnittstelle nach außen zum Server bieten die views (nicht zu verwechseln mit der View), die die Kontrolle über die Anwendung bieten. Diese beinhalten die Geschäftslogik und dienen als Verbindung zwischen den Modellen und den Templates.

2.2 Klassen

Für die Anwendung stand zuerst die Erstellung der Klassen bzw. der Datenbankmodelle die benötigt werden, um die vorgestellten Ziele erreichen zu können. Folgende Klassen werden für die Anwendung benötigt.

Im Zentrum stehen die Tipps, welche die User abgeben können. Die Tipps setzen sich zusammen aus den einzelnen Usern (Tipper) und den Spielbegegnungen. In diesem Fall die Spiele der WM 2010. Weiterhin setzen sich diese aus Den Mannschaften und weiteren Informationen, wie Spielzeit den Ergebnissen zusammen.

```
1  class Tipps(models.Model):
2      user = models.ForeignKey(User, unique=False)
3      begegnung = models.ForeignKey(Begegnung, unique=False)
4
5      toreHeim = models.IntegerField(max_length=2)
6      toreGast = models.IntegerField(max_length=2)
7      tippDatum = models.DateTimeField()
```

[1] Model View Control

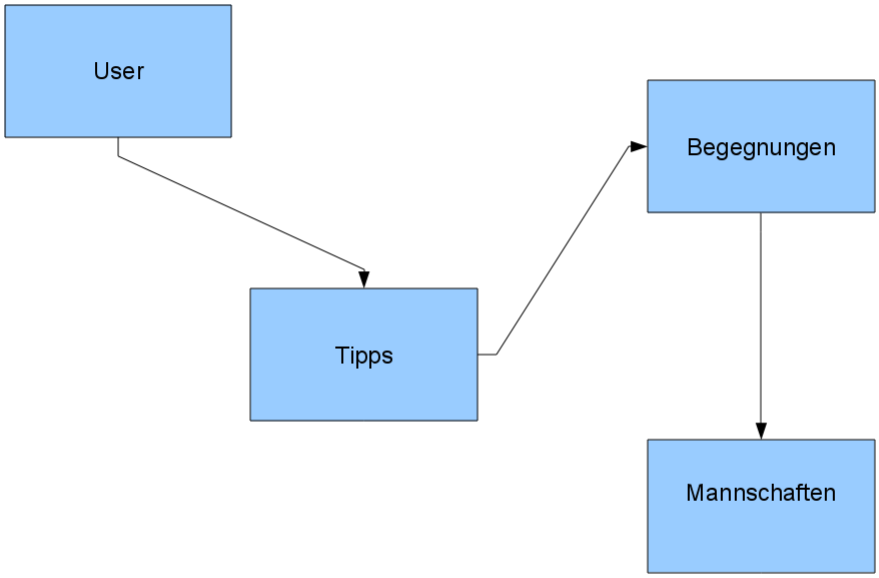

Abbildung 2.1: Benötigte Klassen

```
8
9       def __unicode__(self):
10          return u'%s %s' % (self.user, self.begegnung)
```

Listing 2.1: Modelle in Django

2.3 OR-Mapping

Mit Hilfe des eingebauten OR-Mappers in Django können die Tabellen aus den vorher definierten Klassen als Tabellen in die Datenbank gespeichert werden. Die Speicherung bzw. Synchronisation geschieht mit Django eigenen Boardmitteln.

```
1 python manage.py syncdb
```

Listing 2.2: Datenbanksynchronisation

Damit Python mit der Datenbank kommunizieren kann muss vorher ein Datenbank-Connector installiert werden. Django unterstützt Oracle, PostgreSQL, MySQL und SQ-Lite. Die Installation des Connectors ist abhängig von der Datenbank, die benutzt werden

soll. Getestet wurden mySQL und PostgreSQL hierbei bei der lokalen Entwicklung unter Ubuntu mySQL und auf dem Produktionsserver Postgres. Bei der Speicherung und Auslesen der Daten erwies sich mySQL toleranter, für den Produktionsserver mussten drei Views[2] für PostgreSQL angepasst werden, weil Exceptions von Django zurückgegeben wurden, die auf Probleme mit dem Speichern und Auslesen von Daten zurückzuführen waren.

2.4 Administration

Mit Hilfe der Django eigenen Administrationsoberfläche, die einfach als Applikation in der *settings.py*[3] installiert werden kann, können leicht die benötigten Einträge in die Datenbank geschrieben werden. Auch *Datetime-Felder* werden erkannt und es werden spezielle Widgets für die Eintragung von Datenfeldern zur Verfügung gestellt. In dem Zusammenhang wurde entschienden, die benötigten Informationen über Skripte, automatisch befüllen zu lassen genaueres im Kapitel **??** auf Seite **??**.

[2]Django funktion
[3]Datei beinhaltet alle Umgebungsvariablen des Projektes

3 Implementierung

3.1 Views

Die Geschäftslogik wird mit Hilfe der Views erstellt. Der Name ist hier etwas irreführend, weil man View mit der Ausgabe, also HTML verküpft. Die View ist die Schnittstelle, mit der die Anwendung mit dem Server/Client kommuniziert. Im wmTippspiel wird beim Aufruf der Startseite, automatisch die dafür zuständige View angesprochen. Diese führt Befehle aus, die ihre Ergenisse einem Template übergeben kann. Das Template generiert aus den übergebenen Informationen schließlich dann HTML, welches dem Server zur Weiterleitung an den Client zur Verfügung gestellt wird.

3.2 URL-Dispatching

In Django gibt es eine Kontrollinstanz, die ermöglicht, abhängig von der URL, auf die verschiedenen Views der Seite zu verweisen. Einstellungen können in der *urls.py* gemacht werden. Mit Hilfe regulärer Audrücke wird die URL untersucht. Der erste Ausdruck auf dem die URL passt, ermittelt und die dahinterliegende View ausgeführt.

```
1  urlpatterns = patterns('',
2      (r'^$', index ),
3      (r'^time/$', current_datetime ),
4      (r'^displaymeta/$', display_meta ),
5      #(r'^login/$', mylogin ),
6      (r'^accounts/logout/$', 'django.contrib.auth.views.logout', {'next_page':'/'}),
7      (r'^accounts/login/$', 'django.contrib.auth.views.login'),
8      # Example:
9      # (r'^wmTippspiel/', include('wmTippspiel.foo.urls')),
10
11     # Uncomment the admin/doc line below and add 'django.contrib.admindocs'
12     # to INSTALLED_APPS to enable admin documentation:
13     # (r'^admin/doc/', include('django.contrib.admindocs.urls')),
14
15     # Uncomment the next line to enable the admin:
16     (r'^admin/', include(admin.site.urls)),
17
18     (r'^tippspiel/', include('wmTippspiel.appWMTippspiel.urls')),
19
20 )
```

Listing 3.1: Auszug ursl.py

3.3 Django App

Im wmTippspiel wurde eine App[1] erstellt und ausgelagert, so kann die erstellte Applikation auch in anderen Internetseiten ohne viel Aufwand eingbunden werden.

```
1 (r'^tippspiel/', include('wmTippspiel.appWMTippspiel.urls')),
```

Die appWMTippspiel bildet den Kern des wmTippspiel-Projekts. Daneben sind weitere Applikationen, wie eine Registrierungs-Applikation (siehe Kapitel **??**) denkbar, damit sich User bequem auf der Seite registrieren können. Diese Applikation kann thematisch gut von der appWMTippspiel abgegrenzt werden und in anderen Projekten eingesetzt werden. Was zu dem Don't repeat Yourself Gedankten Djangos passt.

[1] Eigenständiger Teil einer Webanwendung, kann in verschiedenen Projekten eingebunden werden

4 Zugangsmechanismus

4.1 User-Authentifikation

In unserem Projekt stellte sich das User-Authentifikation-System als eines der drei größeren UseCases heraus. Wichtig erschien uns, dass der Besucher beim ersten Besuch ohne großen Zeitaufwand einen Account erstellen kann und dann gleich mit Tippen loslegen kann, also eine Registrierung mit automatischem Login. Später, bei jedem weiteren Besuch, erfolgt dann der gewohnte Login per Username und Passwort. Da die Registrierung, auch aufgrund unseres Layouts, nicht mit einem zweiten Passwortkontrollfeld ausgestattet ist, musste zusätzlich noch eine „Passwort vergessen?“-Funktionalität eingebaut werden.

Die Arbeit mit dem Web-Framework Django ermöglichte nun den Einsatz des Framework eigenen Authentifizierung-Werkzeuges, zu dem hier zunächst ein kleiner Einblick zu der Arbeitsweise gegeben werden soll. Django setzt dabei auf eine User-Session mit Cookie zur Speicherung der Session-ID, ein Verfahren, das mittlerweile bei so ziemlich jedem Framework oder CMS Standard ist. Als Aufgabenfelder dienen die Verwaltung von User, Gruppen und deren Berechtigungen. Da wir in unserem Projekt nur die Gruppierungen der normalen User und Superuser (nur für auto-generierten Admin-Bereich) benutzen und selbst keine eigenen Gruppen anlegen, wird hier nur das User-Model vorgestellt. Im Grunde genommen benötigen wir für dieses nur die User-Daten Username, Passwort, Emailadresse und ein paar versteckte Daten, wie „is_active“, „last_login“, etc. So gesehen, reicht uns also das Django eigene User-Model völlig aus und kann gleich so übernommen werden. Auch bei den Berechtigungen kommen wir relativ spartanisch zurecht, weil wir nur zwischen Inhalten unterscheiden, die der eingeloggte User sehen darf und solche, die jeder Besucher sehen darf (Admin-Bereich ist außen vor!). Hier reicht also auch nur eine einfache Prüfung, welche Django selbst in der View-Datei per „Shortcut“ übernimmt:

```
1 @login_required
2     def meinView(request):
```

```
3     bspVar = 1 ...
```

Listing 4.1: Decorator

4.2 Admin-Bereich für das User-Management

Zuallererst eine gute Nachricht: das Entwickeln des Admin-Bereichs (eine aufwendige Aufgabe, deren Mühen der normale Besucher nicht würdigt) kann man sich mit Django getrost sparen. Es generiert automatisch für jedes erstellte Model gewünschte Formulare in einem geschützten Admin-Bereich, der nur für die oben angesprochenen Superuser zugänglich ist. Einzelne Änderungen der Formulare oder Bedienbarkeit können mit Hilfe der Django Online-Dokumentation schnell gemacht werden, meistens sind sie aber nicht von Nöten.

4.3 Login-Werkzeug von Django

Auch beim Erstellen eines Logins kann man sich im Großen und Ganzen auf die Bibliothek django.contrib.auth.views.login verlassen.. Dies kann ganz einfach direkt in der url.py aufgerufen werden, wahlweise mit einem Template oder auch ganz spartanisch ohne. Hat man sich für ein Template entschieden, ist es hier möglich sich die Input-Felder (HTML) per Django's eigener Template-Sprache generieren zu lassen, oder, wie wir es gemacht haben, selbst mit HTML zu erstellen. Dies ermöglichte uns einen schnelleren Zugriff auf die Attribute des Input-Feldes, da wir Änderungen aus layouttechnischen Gründen benötigten. Ein interessantes Feature bietet die „next"-Funktion des Logins, die als GET-Parameter übergeben, dem Besucher besseres Navigieren erlaubt: z.B. möchte der nicht eingeloggte User in der Galerie ein Bild betrachten. Dies ist jedoch nur authentifizierten Usern gestattet. Deshalb wird ein next-Parameter mitgegeben, in welchem die Adresse zu dem Bild gespeichert wird, damit er nach erfolgreichem Login dort auch landet.

4.4 Logout bei Django

Für den Logout reicht ebenfalls die vorgegeben Funktion django.contrib.auth.views.logout in der urls.py einzustellen, da sie ja kein eigenes Template oder besondere Einstellungen bedarf.

4.5 Registrierung

Für die Registrierung selbst gibt es allerdings keine Hilfen seitens Django, da sie meist eine sehr individuelle Sache ist, z.B. möchte man bei einer Webshop Kontodaten, bei einer Online-Community vielleicht Geburtsdatum und Hobbies usw. wissen. Auch bieten sich hier besondere Sicherungsmöglichkeiten wie Captcha etc. an, was wir allerdings für unser kleines Tippspiel aufgrund der geringen Laufzeit nicht benötigten. Da wir einen schnellen Einstieg für den Teilnehmer uns wünschten, verzichteten wir außerdem auf eine Aktivierungsemail und loggten den User sofort nach der Registrierung ein.

4.6 Passwort vergessen

Da eine erleichterte Registrierung schnell zu fehlerhaften bzw. vergessenen Passwörtern führen kann, wurde noch eine „Passwort vergessen“-Funktion eingeführt, die dem Teilnehmer nach dem Eintragen seiner E-mailadresse, sein neues, zufallsgenerierte Passwort zuschickt. Bei der Entwicklung dieser Funktion kann man sich auch hier wieder auf Django verlassen, das ein Werkzeug zum E-mail versenden anbietet, als auch eine Funktion zur Erstellung von zufälligen Passwörtern, was jedoch bei uns selbst durch ein kleinen Einzeiler programmiert wurde.

5 Hilfsskripte

5.1 Auslesen der Mannschaften

Um die teilnehmenden Mannschaften nicht alle von Hand in die Datenbank eintragen zu müssen haben wir ein Skript geschrieben, dass mit BeautifulSoup den Quelltext der Fifa Homepage (http://de.fifa.com/worldcup/standings/index.html) einliest, die relevanten Informationen extrahiert und damit die Datenbank füllt. Dabei hat sich herausgestellt, dass BeautifulSoup nicht ganz so flexibel ist, wie es wünschenswert gewesen wäre. So sind die Informationen auf der Seite der Fifa z.B. in einer Tabelle angeordnet. Dabei wären Selektoren wie sie bei CSS3 vorkommen wünschenswert gewesen (z.B. 'tr td.l a strong'). So war es z.B. nicht möglich eine Tabellenzelle ausgehend von der Klasse der Zeile zu selektieren. Dies hätte das ganze dynamischer gemacht und wäre auch mit einem leicht veränderten Design von Seiten der Fifa zurecht gekommen. Da dies nicht möglich ist haben wir die Pfade ßtatischäufgebaut, was das Skript natürlich schlecht wiederverwendbar macht, da man für jede neue Seite die Pfade komplett neu aufbauen muss. Eine Alternative wäre hierbei lxml gewesen. Dort wird CSS3-Selektor Unterstützung versprochen. Das Problem dabei ist nur, dass lxml ein Pythonic binding für die Bibliotheken "libxml2ünd "libxsltïst. Diese Bibliotheken laufen allerdings unter C und da wir im "pure Pythonßoweit wie möglich bleiben wollten, haben wir uns (auch der Einfachheit halber um nicht noch einen Parser "lernenßu müssen) für BeautifulSoup entschieden. Dieses Skript wurde vor dem offiziellen Start des Tippspiels einmal zum Befüllen der Datenbank ausgeführt.

5.2 Auslesen der Spiel-Ergebnisse

Das Auslesen der Mannschaften per Skript zu realisieren war nur eine kleine Erleichterung, da es nur einmal ausgeführt wurde. Da wir aber nun schon ein Skript haben, welches mit der Struktur der Fifa Seite vertraut ist, haben wir dieses Skript angepasst um die Ergebnisse der Spiele auszulesen (von http://de.fifa.com/worldcup/matches/index.html).

Interessant hierbei war, dass bei manchen Spielen ein Halbzeitergebnis angegeben wurde und bei anderen nicht. Dieses stand allerdings in der gleichen Tabellenzeile wie das "normaleËrgebnis. Mit Hilfe von string.find() und anderen String-Funktionen konnten wir dies jedoch relativ zuverlässig filtern. Zur Identifikation der Spiele zogen wir das Datum samt Uhrzeit heran, was allerdings zu einem Problem wurde, als mehrere Spiele zur gleichen Zeit stattfanden. Hier musste noch auf die jeweiligen Mannschaften ausgewichen werden. Das grundlegende Problem besteht nunmal darin die Daten der Homepage mit denen der Datenbank abzustimmen und entsprechend zu updaten. Um das Skript nicht nach jedem Spiel per Hand ausführen zu müssen, haben wir einen Cronjob geschrieben, der das Skript in regelmäßigen Abständen ausführt und so die aktuellen Ergebnisse einpflegt.

6 Darstellung der Anwendung

6.1 Django Templatesprache

Django besitzt eine eigene Template-Sprache, die es ermöglichen soll, dass der Designer unabhängig vom Programmierer mit der Implementierung des Layouts beginnen kann. Über eine Schnittstelle übergibt Letzterer die benötigten Variablen an das Template, wo sie der Designer positionieren kann. Hier tritt ein grundlegendes Problem auf: zum Einen möchte der Designer bei jeder Seite nicht das komplette Layout neu einbauen müssen, da er sonst im schlimmsten Fall bei einer Layoutänderung, z.B. Copyright-Rechtsbestimmungen erneuert, mehrere hundert Dateien ändern müsste. Aus diesem Grund kann mit Hilfe der Template-Sprache eine Basis-Datei angelegt werden, in der die inhaltlichen Blöcke markiert werden, z.B.

```
1 {% block content %}
2 <h1Ü>berschrift</h1>
3 {% endblock %}
```

Listing 6.1: Beispiel

Nun kann man weitere Templates anlegen, die sich mit dieser Basis-Datei erweitern, jedoch einen eigenen Content-Block besitzen und somit den alten Block der Basis-Datei mit dem eigenen Inhalt ersetzen.

Weiter muss eine Template-Sprache dem Nutzer vielfältige Formatierungsmöglichkeiten bieten, welche dem Designer den Weg zu einer dynamischen Seite ebnet. Ein Hilfsmittel sind die bekannten Tags, wie IF-ELSE-Blocken und Schleifen, die Listen, Dictionaries oder Objekte auslesen. Ein weiteres Hilfsmittel sind die Filter, die z.B. prüfen, ob die gewünschte Variable existiert bzw. ein None oder False zurückgibt und dies dann mit vorgegebenen Defaultwert ersetzen. Häufig werden auch Tag-Filter, benutzt, um HTML-Tags zu beseitigen oder zu verändern.

7 Fazit

Das Erstellen des WMTippspiel hat viel Spaß gemacht aber hat auch auf der anderen Seite viel Zeit in Anspruch genommen. Aufgrund der Aktualität zur Weltmeisterschaft war es super zu sehen, wie die User der Anwendung mitgefiebert haben und der Seite treu geblieben sind. Die Arbeit mit Django ging reibungslos nur die Portierung auf dem Produnktionsserver hat etwas Zeit in Anspruch genommen. Da etwas Aufwand von Nöten war Apache und WSGI mit der Anwendung kommunizieren zu lassen. Aber nach anfänglichen Hürden stand das Projekt online.

Wir haben noch viele Ideen für die Anwendung, doch aus Zeitgründen konnten nicht alle Ideen umgesetzt werden. Die Ideen nehmen wir mit und vielleicht entsteht aus dem Projekt ein Bundesliga-Tippspiel für die nächste Saison.