- Studienarbeit Python -
WM-Tippspiel 2010

Moritz Schächterle, Dominik Heimstädt & Andrés Cuartas

6. Juni 2010

Inhaltsverzeichnis

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.

2 Django

Das Django Webframework eignet sich für die Erstellung von Webanwendungen. Es folgt dem MVC[2] 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.

[1] Objekt-Relational Mapping
[2] Model View Control

3 Modelle

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.

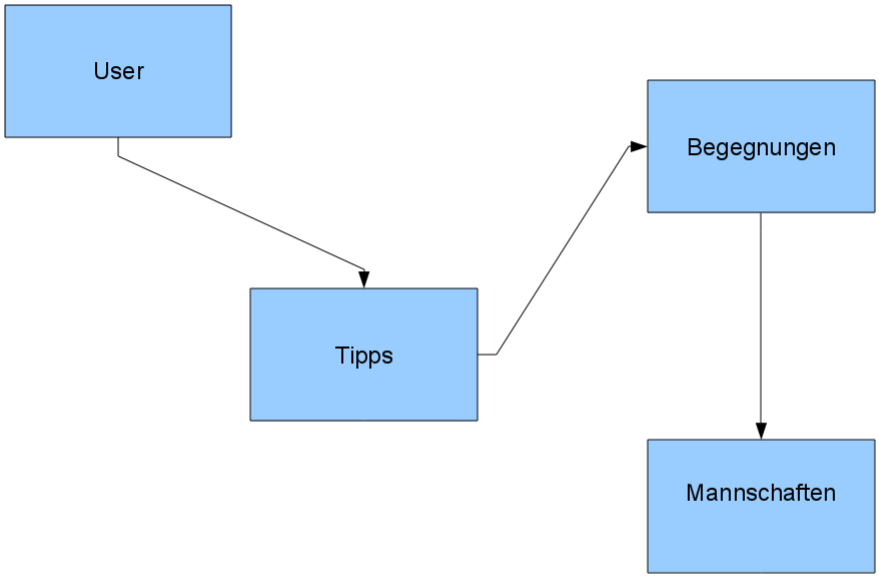

Abbildung 1: Benötigte Klassen

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()
8
9      def __unicode__(self):
```

```
10          return u'%s %s' % (self.user, self.begegnung)
```

Listing 1: Modelle in Django

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: Datenbanksynchronisation

Damit python mit der Datenbank kommunizieren kann muss vorher ein Datenbank-Connector installiert werden. Django unterstützt Oracle, PostgreSQL, MySQL und SQLite. 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[3] für PostgreSQL angepasst werden, weil Exceptions von Django zurückgegeben wurden, die auf Probleme mit dem Speichern und Auslesen von Daten zurückzuführen waren.

Mit Hilfe der Django eigenen Administrationsoberfläche, die einfach als Applikation in der *settings.py*[4] 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 6 auf Seite 6.

4 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] Django funktion
[4] Datei beinhaltet alle Umgebungsvariablen des Projektes

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.

```
urlpatterns = patterns('',
    (r'^$', index ),
    (r'^time/$', current_datetime ),
    (r'^displaymeta/$', display_meta ),
    #(r'^login/$', mylogin ),
    (r'^accounts/logout/$', 'django.contrib.auth.views.logout', {'next_page':'/'}),
    (r'^accounts/login/$', 'django.contrib.auth.views.login'),
    # Example:
    # (r'^wmTippspiel/', include('wmTippspiel.foo.urls')),

    # Uncomment the admin/doc line below and add 'django.contrib.admindocs'
    # to INSTALLED_APPS to enable admin documentation:
    # (r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
    (r'^admin/', include(admin.site.urls)),

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

)
```

Listing 3: Auszug ursl.py

Im wmTippspiel wurde eine App[5] erstellt und ausgelagert, so kann die erstellte Applikation auch in anderen Internetseiten ohne viel Aufwand eingbunden werden.

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

So bildet die appWMTippspiel die Kernapplikation der wmTippspiel-Projekts.

5 Registrierung

6 Hilfsskripte

7 Templates

8 Portierung auf Produktionsserver

9 Fazit

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