Accelerating Information Technology Innovation

http://aiti.mit.edu

Cali, Colombia
Summer 2012
Lesson 2 – Django Models

MISTI

Google

MIT

What is a model?

- A **class** describing data in your application
- Basically, a class with **attributes** for each data field that you care about
- The schema for your data
  - A diagrammatic presentation

Why use Django models

- Avoid direct work with the database
- No need to handle database connections, timeouts, etc. Let Django do it for you.
- Class that extends models.Model

Django fields

- All you do is define a field type
  - Ex: active = models.BooleanField()
- Django handles the rest:
  - Bit value in sql database
  - Represented as a checkbox on a webpage
  - Validation of values

Before Models:

```
from django.shortcuts import render_to_response
import MySQLdb

def book_list(request):
    db = MySQLdb.connect(user='me', db='mydb',
                                        passwd='secret', host='localhost')
    cursor = db.cursor()
    cursor.execute('SELECT name FROM books ORDER BY name')
    names = [row[0] for row in cursor.fetchall()]

    db.close()

    return render_to_response('book_list.html', {'names': names})
```

After Models:

```
from django.shortcuts import render_to_response
from mysite.books.models import Book

def book_list(request):

    books = Book.objects.order_by('name')

    return render_to_response('book_list.html', {'books': books})
```

Django Model Syntax

```
class Musician(models.Model):
    first_name = models.CharField(max_length=50)
    last_name  = models.CharField(max_length=50)
    instrument = models.CharField(max_length=100)
    def __unicode__():
        return last_name+”, “+first_name

class Album(models.Model):
    artist       = models.ForeignKey(Musician)
    name         = models.CharField(max_length=100)
    release_date = models.DateField()
    num_stars    = models.IntegerField()
    def __unicode__():
        return name
```

Important Django field types

- BooleanField
  - Checkbox
- CharField(max_length)
  - Single-line textbox
- DateField
  - Javascript calendar
- DateTimeField
  - Javascript calendar, time picker

Important Django field types

- DecimalField(max_digits, decimal_places)
  - Decimal numbers
- EmailField
  - Charfield that validates email address
- FileField
  - File upload, stores path in database
- FloatField
  - Floating point numbers

Important Django field types

- ImageField ***Don’t use
  - Stores images
- IntegerField
  - Integer textbox
- PositiveIntegerField
  - Integer textbox for positive integers
- TextField
  - Multi-line textbox

Important Django field types

- TimeField
  - Time picker
- URLField
  - Textbox for URLs
- Anything you create

Django Relationship Fields

- ForeignKey(foreign class)
  - Many-to-one
- ManyToManyField(foreign class)
  - Uses a temporary table to join tables together
- OneToOneField(foreign class)
  - Enforces uniqueness (i.e. foreign key with unique=True)

Field options

- **null**: if True, empty fields will be stored as NULL in database.
- **blank**: if True, field is allowed to be blank. default is False.
- **choices**:
  - List or tuple of 2-tuples to use as field choices
  - Django will represent it with a drop-down instead of a textbox

```
class Foo(models.Model):
    GENDER_CHOICES = ( ('M', 'Male'), ('F', 'Female'), (‘NS’, ‘Not specified’) )
    gender = models.CharField(max_length=2, choices=GENDER_CHOICES)
```

More Field Options

default: default value for a field

primary_key: if True, this field is the primary key for the model

unique: if True, this will have to be unique throughout the table

verbose_field_name: provides a human readable file name

DateField and DateTimeField options

- **auto_now**
  - Any time the object is saved, the field will be updated with the current time.
- **auto_now_add**
  - The time will always be equal to the creation date of the object.

Model Methods

- __unicode__():
  - Equivilant of toString – used for auto-generated admin pages
- get_absolute_url()
  - Used for deciding URLs that reference a specific object

Django Model Syntax (example)

```
class Musician(models.Model):
    first_name = models.CharField(max_length=50)
    last_name  = models.CharField(max_length=50)
    instrument = models.CharField(max_length=100)
    def __unicode__():
        return last_name+”, “+first_name

class Album(models.Model):
    artist       = models.ForeignKey(Musician)
    name         = models.CharField(max_length=100)
    release_date = models.DateField()
    num_stars    = models.IntegerField()
    def __unicode__():
        return name
```

Creating Models Manually

```
>>> from music.models import Musician

>>> m1 = Musician(first_name='Jimi', last_name='Hendrix',
instrument='guitar')

>>> m1.save()

>>> m2 = Musician(first_name="Eric", last_name='Clapton',
      instrument='guitar')
>>> m2.save()

>>> Musician_list = Musician.objects.all()

>>> Musician_list
[<Musician: Hendrix, Jimi>, <Musician: Clapton, Eric>]

#remember the unicode!!
```

Filtering

```
>>>Musician.objects.filter(first_name=”Jimi”)
[<Musician: Hendrix, Jimi>]

>>>Musician.objects.filter(instrument=”guitar”)
[<Musician: Hendrix, Jimi>, <Musician: Clapton, Eric>]

#returns a QuerySet, not an individual Model Object

>>>Musician.objects.filter(last_name__contains=”Clap”)
[<Musician: Clapton, Eric>]

#double underscore!!
```

Getting

```
>>>Musician.objects.get(first_name=”Jimi”)
<Musician: Hendrix, Jimi>

#returns single object

>>>Musician.objects.get(instrument=”violin”)
Error! DoesNotExist

>>>Musician.objects.get(instrument=”guitar”)
Error! MultipleObjectsReturned

#use try/except when using “get”.
```

Ordering

```
>>>Musician.objects.order_by(-last_name)
[<Musician: Hendrix, Jimi>, <Musician: Clapton, Eric>]
```

- Easier way: add class **Meta** to Model class

```
class Musician(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    instrument = models.CharField(max_length=100)

    def __unicode__():
        return last_name+", "+first_name

    class Meta:
        ordering = [-last_name]
```

More Functionality

```
>>>m1.instrument=”drums”
>>>m1.save()
      #updates ALL rows, could lead to “race” condition

>>>Musicians.objects.filter(id=12).update(instrument=”bass”)
       #updates only “instrument” row
```

Chaining

```
>>>Musicians.objects.filter(instrument="guitar").order_by("-
last_name")
[<Musician: Hendrix, Jimi>, <Musician: Clapton, Eric>]
```

Rules of Django Models

1. When you update a model, ALWAYS RUN
   `python manage.py syncdb`
2. All classes extend models.Model
3. Models only live in **Apps**
4. Django doesn't save objects until you call **save()** method

```
>>>a1 = Album(...)
# a1 is not saved to the database yet!
>>>a1.save()
# Now it is saved.
```

Tips for Django Models

1. Keep code clean

2. Always create a __unicode__() method

3. Name your variables well

4. Don’t think too much about the database