Intro to Django (1.6.2)

Mustafa Khattab
https://github.com/mkhattab

March 18, 2014

Project: Cat Gif Aggregator

We’re going to build a web app that allows users to view and rate user-submitted cat gifs links.

https://github.com/mkhattab/iepug_django_intro

But first...what is Django?

- Web framework born 2003 in Lawrence, Kansas (no place like home) for a publishing site, Lawrence-Journal World
- DRY: don’t repeat yourself
- MVC: model-view-controller

  (we’ll get back to this later)

But why?

use Django, that is.

Features: everything plus the kitchen sink

- ORM: object relational mapper (w/ support for most of the major RDBMS’)
- Admin interface
- URL Routing
- Template system
- Cache support
- Internationalization and localization
- Authentication
- Sessions
- Forms
- Security: CSRF & clickjacking protection, signed cookies, etc.
- Email
- Syndication, pagination, static files management, and more!

source: djangoproject.com

Rapid Application Development

Provided you stay within the bounds of the framework.

Used By

- Mozilla
- Disqus
- Instagram
- Pinterest
- Rdio
- many others

source: djangoproject.com

“I’m convinced. I’m going to use Django for every project!”

Hold on...not a good idea. We’ll talk about this later.

Installation

- Via PIP
  - `pip install django`
- Within a virtualenv
  - `virtualenv django`
  - `source django/bin/activate`
  - `pip install django`

Creating our project

- Via `django-admin.py`
    - `django-admin.py startproject catgifs`

`tree catgifs`

```
catgifs/
├── catgifs              -- root project module
│   ├── __init__.py
│   ├── settings.py      -- app settings
│   ├── urls.py          -- root URL conf
│   └── wsgi.py          -- WSGI handler
└── manage.py            -- CLI swiss army knife
```

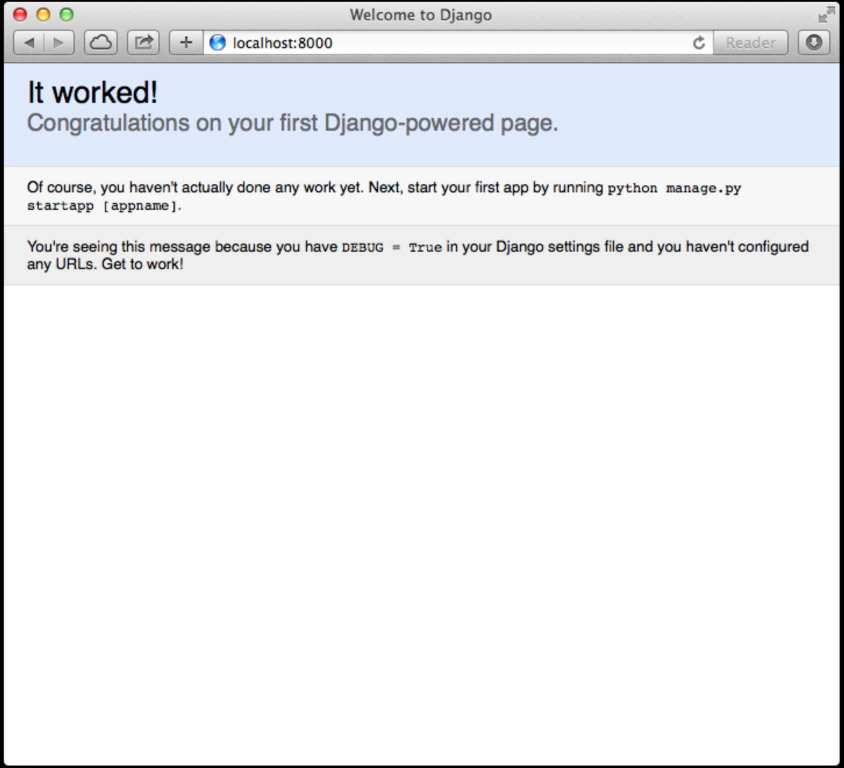
manage.py --help

```
Available subcommands:

[auth]
    changepassword
    createsuperuser

[django]
    check
    cleanup
    compilemessages
    createcachetable
    dbshell
    diffsettings
    dumpdata
    flush
    inspectdb
    loaddata
    makemessages
    runfcgi
    shell
    sql
    sqlall
    sqlclear
    sqlcustom
    sqldropindexes
    sqlflush
    sqlindexes
    sqlinitialdata
    sqlsequencereset
    startapp
    startproject
    syncdb
    test
    testserver
    validate

[sessions]
    clearsessions

[staticfiles]
    collectstatic
    findstatic
    runserver
```

Sanity check

- Verify we have a working installation
  - `manage.py runserver`
- By default, Django will start a HTTP server on port 8000
- See `manage.py runserver --help` for more options

You should see this



It worked!

Congratulations on your first Django-powered page.

Of course, you haven't actually done any work yet. Next, start your first app by running `python manage.py startapp [appname]`.

You're seeing this message because you have `DEBUG = True` in your Django settings file and you haven't configured any URLs. Get to work!

Creating our app

- Apps are how we group different components or modules in a project.
- A project can have multiple apps and apps can be shared across multiple projects.
- Via `manage.py`
    - `manage.py startapp gifs`

`tree catgifs`

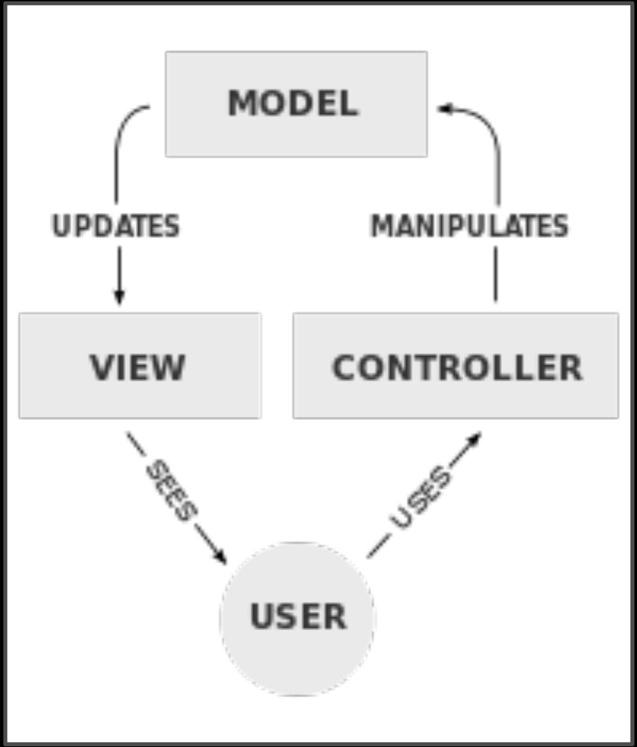
```
.
├── catgifs                     -- root project module (slide #11)
│   ...
├── gifs                        -- gifs app module
│   ├── __init__.py
│   ├── admin.py                -- admin interface config
│   ├── models.py               -- database models
│   ├── tests.py                -- self-explanatory
│   └── views.py                -- request handling logic
└── manage.py
```

Aside: MVC

- Model
  - application data, relationships, logic, functions.
- View
  - representation of data defined Model, usually.
- Controller
  - logic handling application state and view presentation.

source: wikipedia.org

MVC...



source: wikipedia.org

In Django, MVC → MTV

- Models simply are called Models
- View is the Template
- Controller is the View
- Confusing but you’ll adjust

MTV or MVC?: `tree catgifs`

```
.
├── catgifs
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py              ---> URL routing & dispatch; Controller
│   ├── wsgi.py
│   └── [templates/]         ---> Django templates; View
├── gifs
│   ├── __init__.py
│   ├── admin.py
│   ├── models.py            ---> Model (duh)
│   ├── tests.py
│   ├── views.py             ---> Application & Presentation logic; Controller
│   └── [templates/]         ---> App-specific templates; View
└── manage.py
```

`settings.py`

Contains project configuration: database settings, installed apps, static file directory paths, and more.

Database settings

```
--- settings.py (line #58)
DATABASES = {
          'default': {
             'ENGINE': 'django.db.backends.sqlite3',
             'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
           }
       }
```

We’ll stick with SQLite for this project. However, we have the following options for `‘ENGINE’`:

- PostgreSQL: `‘django.db.backends.postgresql_psycopg2’`
- MySQL: `‘django.db.backends.mysql’`
- Oracle: `‘django.db.backends.oracle’`

Installed apps

```
--- settings.py (line #32)
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'gifs',  # add our app to the tuple
)
```

- Django installs quite a few apps by default.
- It’s important to run `manage.py syncdb` on project creation and everytime after installing a new app (or model).

`manage.py syncdb`

- Creates initial database.
- Synchronizes new apps and models.
- Is idempotent, so you can run the command multiple times without worry.
- Unfortunately, it only supports synchronizing non-existing tables, not new/modified columns or constraints (coming soon in 1.7) -- Checkout South.
- We haven’t defined any models in our app, yet.

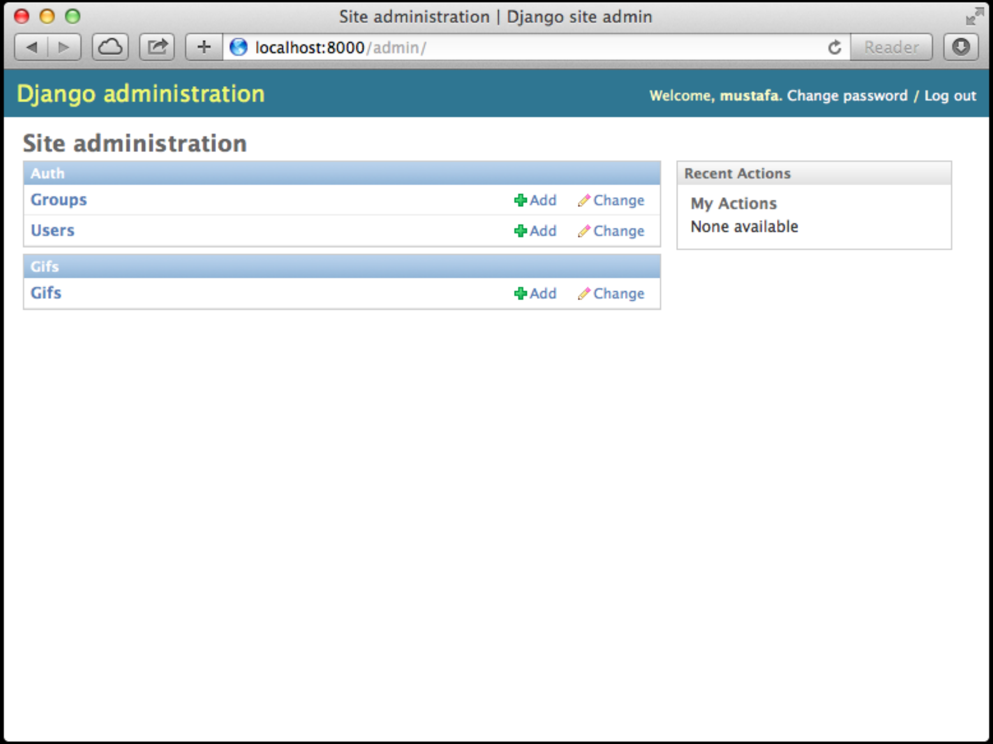
Aside: ORM

- Stands for Object Relational Mapping.
- Python Class definitions map to SQL table definitions.
- Class variables map to SQL columns.
- Caveat: knowledge of the ORM is not a substitute for knowledge of SQL, database design (e.g. 1-to-1, 1-to-*), normalization (e.g. 1NF, 2NF), etc.
- Always remember to run `manage.py syncdb` after adding a new model and make sure your app is listed in `INSTALLED_APPS`

Defining our first model

gifs/models.py

Aside: Admin interface



Site administration | Django site admin

localhost:8000/admin/

Django administration

Welcome, mustafa. Change password / Log out

Site administration

Auth		
Groups	Add	Change
Users	Add	Change

Gifs		
Gifs	Add	Change

Recent Actions

My Actions

None available

Admin interface

- Is bundled with Django and enabled by default.
- By default, you can access the admin via `/admin/`
- Supports customization, see Django docs.
- It’s awesome!
- Register your models in `admin.py`

Views & URL Dispatch

- Views can be defined as functions or classes.
- URIs map to Views
  - `/some/thing ---> view_callable`
- Building elegant URIs are important.
- URL patterns are defined as regular expressions in `urls.py`

Cat GIFs index view

`gifs/views.py`

Templates

- Very simple, yet powerful syntax.
  - conditionals: if, else, ifchanged, etc.
  - iteration: for loops
  - filters: e.g some_var|upper_case
- Supports inheritance
  - For example, let’s say:
  - base.html -- defines header, footer & content block
    - index.html -- inherits base.html & can override parent blocks (or placeholders)

GIFs Index template

gifs/templates/index.html (better reusability)
or
catgifs/templates/index.html

We need to accept user submissions

We can do this via `Forms`

Forms

- Defined similarly like Models.
- It can:
  - Display an HTML form with automatically generated form widgets.
  - Check submitted data against a set of validation rules.
  - Redisplay a form in the case of validation errors.
  - Convert submitted form data to the relevant Python data types.

source: docs.djangoproject.com

Let’s define our submission form

gifs/forms.py

Update our index view

To accept POST requests

Bonus: implement ratings

- Determine the best way to define ratings in `gifs/models.py`
- Remember to run `manage.py syncdb` after defining any new models.
- Django documentation is your friend.

So, why not use Django?

Despite it’s awesomeness

Why not Django?

- Django is an excellent choice for many if not most *traditional* web applications.
- If you’re building an app that is/does:
  - Live chat; HTTP chunked responses/persistent connections.
  - Rich Internet Application (RIA), via e.g. Ember.js, Angular.js --- Django is probably overkill.
  - Web services/APIs only --- again Django is probably overkill.

Resources

- Django documentation is invaluable. It’s all you really need.
    - docs.djangoproject.com
- Books
    - Pro Django
    - Two Scoops of Django: Best Practices