Introduction to Django

Chris Williams

Clarkson Open Source Institute

chris@christopher-williams.net

April 6th, 2011

What is Django?

Python Web Framework

1. Model-View-Controller (technically Model-View-Template) framework for easy web application development
2. Excellent for rapid prototyping, but also sufficient for production applications and deployments
3. Simple enough to learn that a real application can be written while just starting out
4. Comes with "batteries included": Standard sub-applications for common functionality

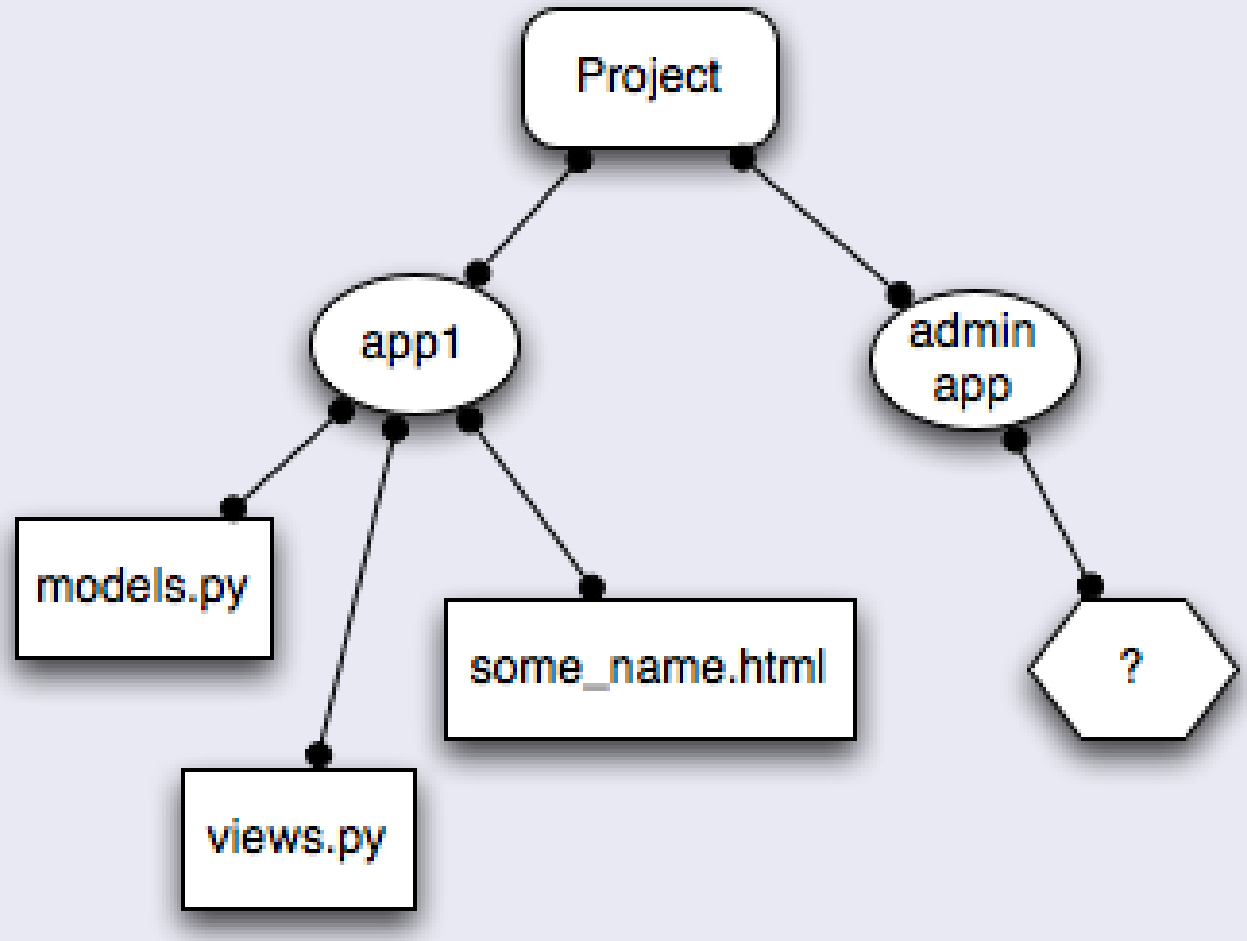
High-level Overview

Major Components of a Django Application

1. Models: Defines database schema as a class (Object-Relational Model, or ORM)
2. Views: Defines functionality of pages or resources, Python function
3. Templates: Bits of HTML with special tags that allow for simple logic and variable interpolation

High-level Overview

Structure of a Django Project



Examples

Example of a Model

```
class Person(models.Model):
  """Person model that stores basic information."""
  name = models.CharField(max_length=100)
  age = models.IntegerField()
  address = models.CharField(max_length=200)
  city = models.CharField(max_length=100)
  state = models.CharField(max_length=2,choices=STATES_LIST)

  def __unicode__(self):
    return self.name
```

Examples

Example of a View

```
def person_index(request):
  """Renders a list of people in the DB."""
  people = Person.objects.all()
  return render_to_response('person_index.html',
      {'people': people})
```

Examples

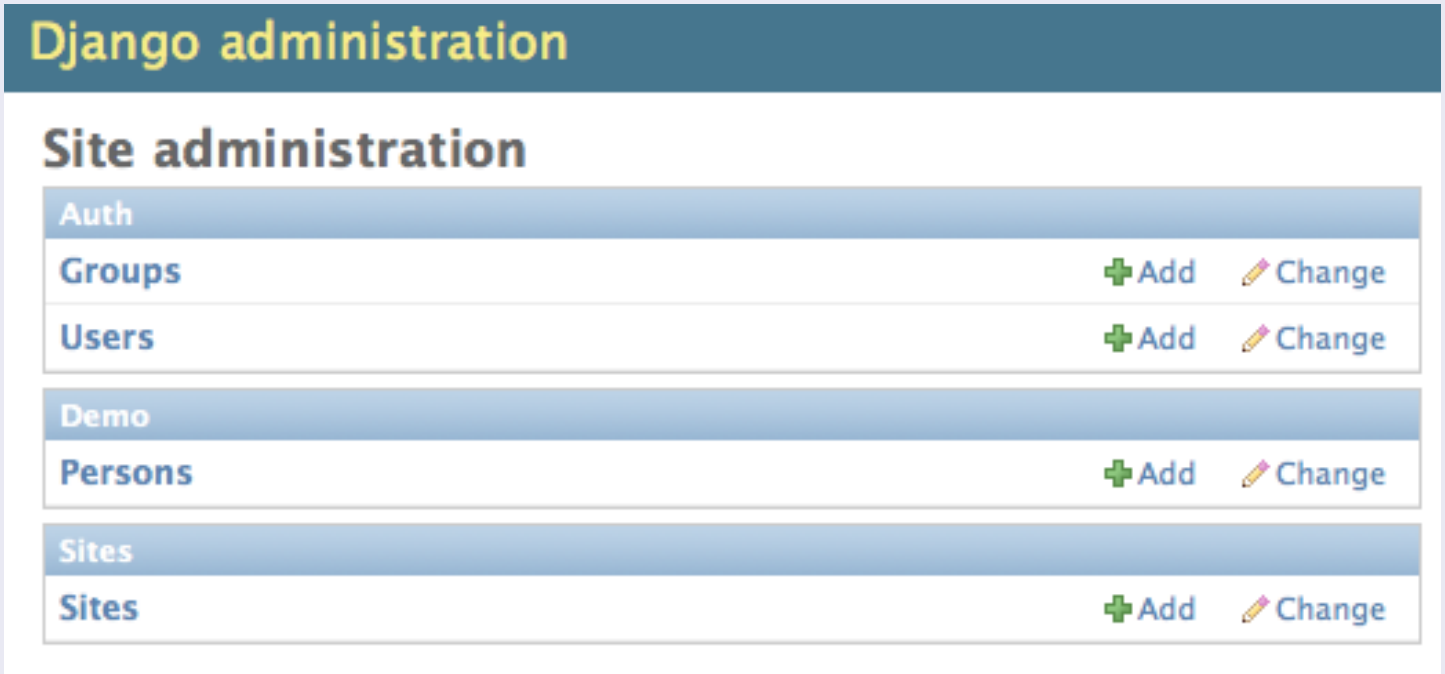
Example of a Template

```
{% extends 'base.html' %}
{% block title %}List of People{% endblock %}

{% block body %}
  <h1>People</h1>
  <ul>
    {% for person in people %}
      <li>{{ person.name }} ({{ person.age }})</li>
    {% endfor %}
  </ul>
{% endblock %}
```

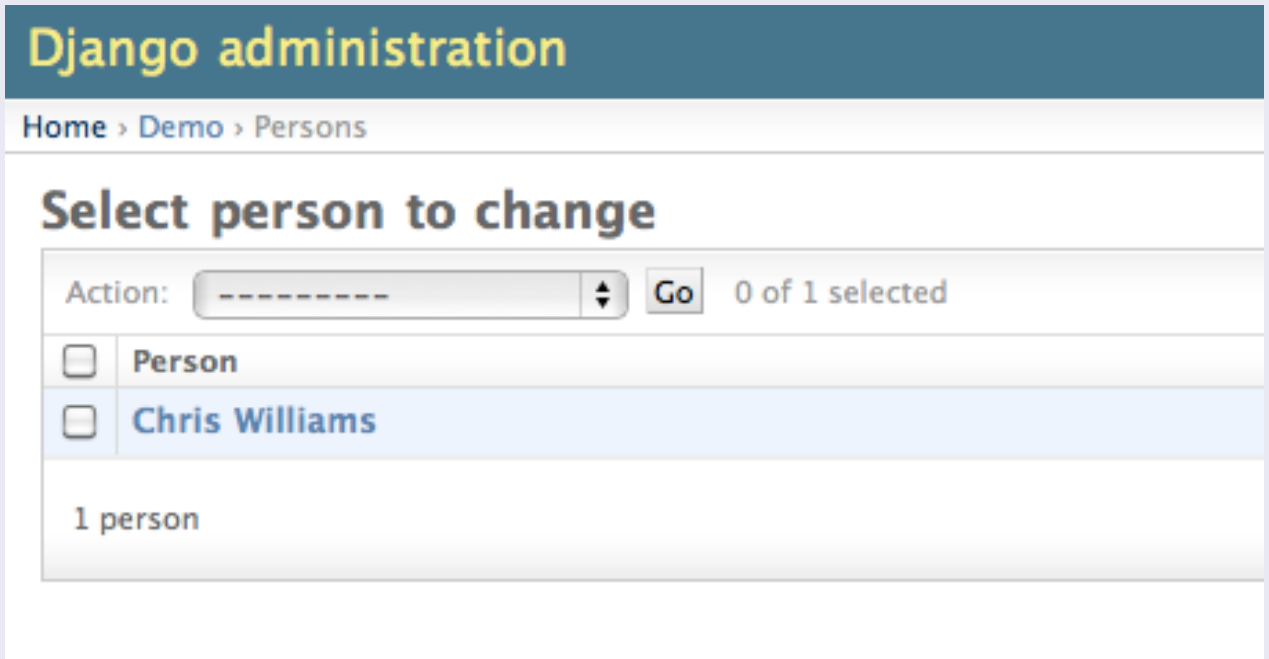
Screenshots

Admin Home



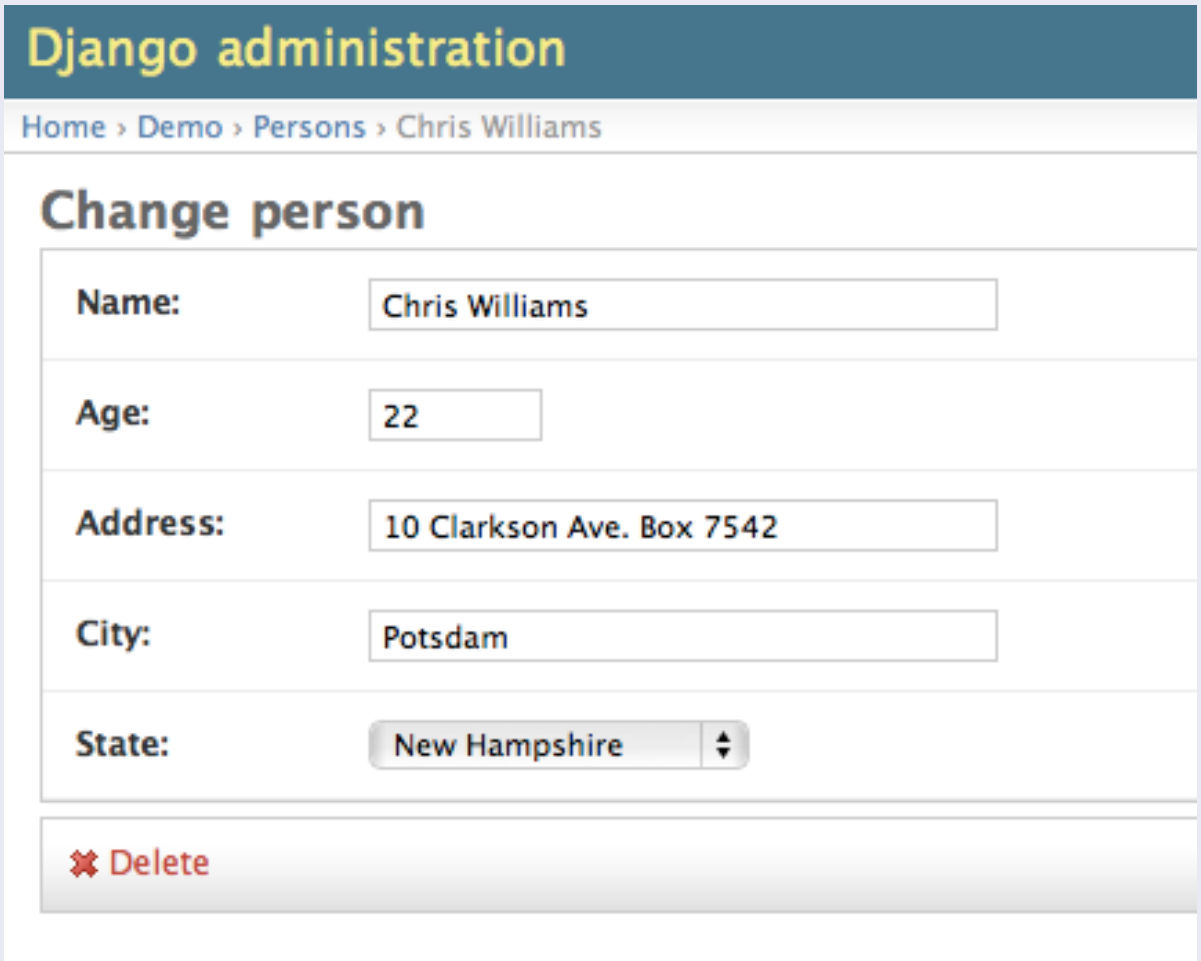
Screenshots

Person List



Screenshots

Edit Person



Digging Deeper into a Django Project

Installing Django

On most UNIX-likes, we use *easy_install*:
$ sudo easy_install django

Creating a Project

$ django-admin.py startproject < myproject >
This creates a new directory with four files: *__init__.py*, *manage.py*, *settings.py* and *urls.py*

Digging Deeper into a Django Project

An Empty Project

1. *__init__.py*: Blank file, makes the directory into a Python module (can be safely ignored)
2. *manage.py*: Script for managing the project (starting development server, syncing database, etc.)
3. *settings.py*: Contains project settings (database information, etc.)
4. *urls.py*: Maps URLs (e.g. */admin/*) to views (e.g. *django.contrib.admin.views.index*)

Digging Deeper into a Django Project

urls.py

1. List of tuples, containing a URI path (*/app/view/*) and a view (*myapp.views.myview*)
2. Allows changes to the architecture of the site without changing URLs ("Cool URLs Don't Change")
3. Allows segmentation of URLs into their respective apps, especially useful for "pluggable" apps

Digging Deeper into a Django Project

Creating a New App

1. Command is *python manage.py startapp [appname]*
2. Creates a new subdirectory named *appname*
3. Make sure to add *appname* to the *installed_apps* tuple in *settings.py*!

Contents on an Empty App

1. *__init__.py*: Blank file, makes the directory into a Python module (can be safely ignored)
2. *models.py*: Defines your models (database tables)
3. *tests.py*: Contains unit tests for the app
4. *views.py*: Contains view functions

Digging Deeper into a Django Project

Third-party Apps

A primary reason for the "app" architecture is to encourage code reuse. There are *many* third-party pluggable apps out there:

A Few Major Apps

1. *django-registration*: Covers logging in/out, registration, forgotten passwords, *etc.*
2. *django-pagination*: Easily splits things into pages (think Google search results)
3. *django-notification*: Provides a nice framework for presenting one-off messages to the user
4. *django-cas-consumer*: Allows for authentication against a Central Authentication Service (shameless plug ☺)

Questions?