Starting a Django Project - Real Python

Answering the question, “How do I setup a Django (1.5, 1.6, 1.7, or 1.8) Project from scratch?”

*Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.***Welcome!**

1. **Read over the Introduction and Setup sections.**
2. **Then choose your poison – Django 1.5, 1.6, 1.7, 1.8 – to setup a Django Project.**
3. **After the initial Project setup, move down to the Create an App section to setup a nice and easy app.**
4. **Then checkout the summary workflow for a quick-start guide to a Django Project.**

Cheers!

Introduction

This tutorial answers the question, *“How do I setup a Django Project from scratch?”*. Since you’re reading this, I assume (err, hope) you know that Django is a Python web framework built for rapid web development. We’ll go through the setup, detailing the basic installation procedures of Django and the dependencies required as well as a few additional libraries/extensions to get you started developing ASAP.

We’ll also look at a basic workflow you can use as soon as your project structure is setup.

Finally, be sure to check out the following videos:

> Although these videos are specific to Django 1.5, they’ll help you understand the basic workflow for Django 1.6, 1.7, and 1.8 as well.

Tutorial Requirements

You should have some Python experience and know basic Unix bash commands. If you’ve never used the command line before, please familiarize yourself with the following commands: *pwd*, *cd*, *ls*, *rm*, and *mkdir*.

For simplicity, all examples use the Unix-style prompt:

```
$ python manage.py runserver
```

(The dollar sign is not part of the command.)

Windows equivalent:

```
C:\> python manage.py runserver
```

Setup

What you need for a basic dev environment:

1. Python 2.7.x or 3.4.x
2. easy_install and Pip
3. Git
4. virtualenv
5. Django
6. Database (SQLite, MySQL, PostgreSQL, MongoDB, etc.)
7. South (for Django versions prior to 1.7)
8. Text editor (Sublime, vim, Komodo, gedit)

> **Note**: This tutorial utilizes Python version 2.7.8.

Python

Unix environments come pre-installed with Python. To check your Python version, run the command:

```
$ python -V
Python 2.7.8
$ python3 -V
Python 3.4.2
```

If you already have a 2.7.x version, move on to the next step. If not, download and install the latest 2.7.x version specific to your operating system.

easy_install and pip

Both easy_install and pip are Python Package Managers, which make it *much* easier to install and upgrade Python packages (and package dependencies).

To download easy_install, go to the Python Package Index (PyPI). You need to download setuptools, which includes easy_install. Download the package egg (.egg), then install it directly from the file.

Pip, meanwhile, is a wrapper that relies on easy_install, so you must have easy_install setup and working first before you can install pip. Once easy_install is setup, run the following command to install pip:

```
$ easy_install pip
```

Git

For version control, we'll be using git. You can check your current version, if you have git already installed, with the following command:

```
$ git --version
git version 2.3.0
```

If you do not have a version greater than 1.7.x installed, please download the latest version.

MySQL

SQLite comes pre-installed with Python, and most tutorials utilize SQLite – so let's push ourselves a bit and use MySQL instead.

First, install MySQL from here.

Next, start the server, and then setup a new database and a user:

```
$ mysql.server start
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
mysql> CREATE DATABASE django_db;
Query OK, 1 row affected (0.01 sec)
mysql>
mysql> quit
Bye
```

virtualenv

It's common practice to use a virtualenv (virtual environment) for your Python projects in order to create self-contained development environments (also called "sandboxes"). The goal of virtualenv is to prevent different versions of libraries/packages from messing with each other.

Think of virtualenv as a completely isolated container within your computer, where you can utilize any version of Python and install libraries/packages and it won't affect anything outside that container. It's like an isolated, soundproof room within your home where you can scream as loud as you want, about anything you want, and nobody else outside that room can hear it.

Install virtualenv with the following command:

```
$ pip install virtualenv
```

Django 1.5

First released on February 26, 2013, the most notable new features include:

1. *Configurable User Model*: Instead of being forced to use Django's definition of a "user", you can now roll your own user model, creating custom fields, like – links to social profiles, date of birth, favorite color, etc.
2. *Python 3 Support*: Django suggests not to use Python 3 in production just yet, since support is still "experimental". However, this feature is huge for the Python community as a whole, helping to put even more pressure to mirage to the much-improved Python 3.

Want to read the full release notes? Check out the official changes from Django here.

The latest release came on January 2, 2015: v1.5.12

Check out the accompanying videos to this tutorial:

Django Install

Set up your development structure:

```
$ mkdir django15_project
$ cd django15_project
$ virtualenv env
$ source env/bin/activate
```

You should see `(env)` before your prompt, `(env)$`, indicating that you're running within the 'env' virtualenv.

To exit the virtualenv, type the following command:

```
$ deactivate
```

Then reactivate when you're ready to work again.

Let's get Django installed:

```
$ pip install django==1.5.12
```

You can check the version by running the following commands:

```
$ python
>>> import django
>>> django.get_version()
'1.5.12'
>>>
```

Project setup

Setup Django project

```
$ django-admin.py startproject my_django15_project
```

This creates a new directory called “my_django15_project” with the basic Django directory and structures:

```
├── manage.py
└── my_django15_project
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

Version control

Before you start any developing, put your project under version control. First, add a new file called *.gitignore* within your “django15_project” directory, which is used to ignore unnecessary files from being added to the git repository.

Add the following to the file:

```
env
*.DS_Store
*.pyc
__pycache__
```

Now initialize (or create) a new Git repo and add your changes to staging and then to the local repo.

```
$ git init
$ git add -A
$ git commit -am "initial commit"
```

If you use BitBucket or GitHub (highly recommended), PUSH your files to your central repo.

Database settings

First, install MySQL-python, which is a database connector for Python:

```
$ pip install MySQL-python
```

Edit your *settings.py* file within your “my_django15_project” directory to add the following information about your database you setup earlier:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'django_db',
        'USER': 'root',
        'PASSWORD': 'your_password',
    }
}
```

Create your database tables and set up a superuser:

```
$ cd my_django15_project
$ python manage.py syncdb
```

Launch the development server:

```
$ python manage.py runserver
```

You should see the following output if Django and the database are setup correctly:

```
Validating models...

0 errors found
September 7, 2014 - 23:36:02
Django version 1.5, using settings 'my_django15_project.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Point your browser at http://127.0.0.1:8000 and you should see the Django “It worked!” page.

Back in your terminal, kill the server by pressing CONTROL-C.

South

South is used for managing changes to your database tables. As your application grows, and you need to add a field to a specific table, for example, you can simply make changes to the database via migrations with South. It makes life *much* easier.

Install South:

```
$ pip install south
```

pip freeze

Now that all your libraries are installed, use the following command to create a record of the installed libraries within the “my_django15_project” directory:

```
$ pip freeze > requirements.txt
```

This command is incredibly useful if you need to recreate your project from scratch and need to know the exact libraries/versions you need to install.

Commit your new changes to Git.

Set up your Django app

Create your new app:

```
$ python manage.py startapp myapp
```

Your project structure should now look like this:

```
├── manage.py
├── my_django15_project
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── myapp
    ├── __init__.py
    ├── models.py
    ├── tests.py
    └── views.py
```

Update the INSTALLED_APPS in your *settings.py* file:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.admin',
    'myapp',
    'south',
)
```

Here, we enabled the admin, `'django.contrib.admin'`, and added the apps, `'myapp'`, and `'south'`.

Open *urls.py* and uncomment these three lines:

```
from django.contrib import admin
admin.autodiscover()
url(r'^admin/', include(admin.site.urls)),
```

Sync your database again:

```
$ python manage.py syncdb
```

Create a database migration file with South:

```
$ python manage.py schemamigration myapp --initial
```

Migrate the database changes:

```
$ python manage.py migrate myapp
```

Fire up the server (`python manage.py runserver`), and then navigate to http://127.0.0.1:8000/admin. Enter your superuser's username and password to login to the admin page.

Commit to Git. For example:

```
$ git add -A
$ git commit -am "updated settings, created app, added south, enabled django admin"
```

Using git, the right way

> **NOTE**: This section is fairly advanced and completely optional. If this is your first time setting up Django, you can skip this section.

If you are not using GitHub or Bitbucket as a central repo, you should create a separate branch of your local repo for development.

First, create a new directory called "dev" within your "django15_project" directory, navigate into the newly created directory, and then clone (copy) your entire Django project:

```
$ git clone /path/to/your/project/
```

> For example: `/Users/michaelherman/desktop/django15_project`

This command creates an exact copy of your repo, which includes all your commits and branches. Always develop from this directory by creating separate branches for each major change to your project:

```
$ git branch <branchname>
```

Next switch to that branch:

```
$ git checkout <branchname>
```

You can always check to see what branches are available with this command:

```
$ git branch
```

After you are done developing, commit your changes:

```
$ git add -A
$ git commit -am "some message"
```

Now you want to merge your changes with the master branch:

```
$ git checkout master
$ git merge <branchname>
```

You can PUSH your changes to GitHub or Bitbucket if you use either service. Finally, navigate back to your main production folder and PULL the changes:

```
$ git pull
```

Example:

```
(env)$ cd dev
(env)$ cd django15_project
(env)$ git branch 06212013
(env)$ git checkout 06212013
(env)$ git add -A
(env)$ git commit -am "description of changes made"
(env)$ git checkout master
(env)$ git merge 06212013
(env)$ cd ..
(env)$ git pull /Users/michaelherman/desktop/django15_project/dev/django15_project
```

Workflow

Now that your app is set up, follow either of these simple workflows every time you want to make changes to your app:

Basic:Advanced:

1. Navigate to your project
2. Activate virtualenv
3. Create and checkout a new git branch
4. Develop
5. Commit changes
6. Merge the new branch with your master branch
7. PULL the changes into the production folder
8. Deploy
9. Deactivate virtualenv

Cheers! Any questions? Suggestions?

Let's create a basic app!

Django 1.6

First released on November 6, 2013, the most notable new features include:

1. *Python 3 Support*: Support for Python 3 is now official, without any restrictions so you can run it in production.
2. *Improved Database Transaction Management*: The API is leaner, cleaner, and simpler, making rollbacks and error handling much easier.
3. *New Test Runner*
4. *Persistent Database Connections*

Want to read the full release notes? Check out the official changes from Django here.

The latest release came on March 18, 2015: v1.6.11

Django Install

Set up your development structure:

```
$ mkdir django16_project
$ cd django16_project
$ virtualenv env
$ source env/bin/activate
```

You should see `(env)` before your prompt, `(env)$`, indicating that you're running within the 'env' virtualenv.

To exit the virtualenv, type the following command:

```
$ deactivate
```

Then reactivate when you're ready to work again.

With your virtualenv activated, install with Pip:

```
$ pip install django==1.6.11
```

You can check the version by running the following commands:

```
$ python
>>> import django
>>> django.get_version()
'1.6.11'
>>>
```

Project setup

Setup Django project

```
$ django-admin.py startproject my_django16_project
```

This creates a new directory called "my_django16_project" with the basic Django directory and structures:

```
├── manage.py
└── my_django16_project
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

Version control

Before you start any developing, place your project under version control. First, add a *.gitignore* file within your "django16_project" directory, which prevent unnecessary files from being added to the git repository.

Add the following to the file:

```
env
*.DS_Store
*.pyc
__pycache__
```

Now initialize (or create) a new Git repo and add your changes to staging and then to the repo.

```
$ git init
$ git add -A
$ git commit -am "initial"
```

If you use GitHub, PUSH your files to your central repo as well.

Database settings

First, install MySQL-python so that Python can communicate with MySQL:

```
$ pip install MySQL-python
```

Edit your *settings.py* file within your “my_django16_project” directory:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'django_db',
        'USER': 'root',
        'PASSWORD': 'your_password',
    }
}
```

Create your database tables and set up a superuser:

```
$ cd my
$ python manage.py syncdb
```

Launch the development server:

```
$ cd my_django16_project
$ python manage.py runserver
```

You should see the following output if all is well thus far:

```
Validating models...

0 errors found
September 7, 2014 - 23:36:02
Django version 1.6, using settings 'my_django16_project.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Navigate to http://127.0.0.1:8000 and you should see the familiar light-blue “It worked!” page.

Kill the server by pressing CONTROL-C back in the terminal.

South

South is used for handling database migrations – e.g, making changes to your database schema.

Install South:

```
$ pip install south
```

pip freeze

With the dependencies installed, use the following command to create a record of them, along with their subsequent versions, within the “my_django16_project” directory:

```
$ pip freeze > requirements.txt
```

This command comes in handy when you need to recreate your project from scratch. You can simply run `pip install -r requirements.txt` to install all your project’s dependencies.

Commit your new changes to Git.

Set up your Django app

Create your new app:

```
$ python manage.py startapp myapp
```

Your project structure should now look like this:

```
├── manage.py
├── my_django16_project
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── myapp
    ├── __init__.py
    ├── admin.py
    ├── models.py
    ├── tests.py
    └── views.py
```

Update the INSTALLED_APPS in your *settings.py* file to include South as well as your new app:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.admin',
    'myapp',
    'south',
)
```

Now as soon as you run `syncdb`, Django will be aware of both `south` and `myapp`.

Open *urls.py* and uncomment these three lines:

```
from django.contrib import admin
admin.autodiscover()
url(r'^admin/', include(admin.site.urls)),
```

Sync your database again:

```
$ python manage.py syncdb
```

Create a database migration file with South:

```
$ python manage.py schemamigration myapp --initial
```

Migrate the database changes:

```
$ python manage.py migrate myapp
```

Fire up the server (`python manage.py runserver`), and then navigate to http://127.0.0.1:8000/admin. Enter your superuser's username and password to login to the admin page. This is just a sanity check to ensure that all is working.

Commit to Git. For example:

```
$ git add -A
$ git commit -am "updated settings, created app, added south, enabled django admin, boom"
```

All set. Let's create a basic app!

Django 1.7

First released on September 2, 2014, the most notable new features include:

1. *Database Migrations*: Django now has built-in support for making database schema changes, which is based on South. This is big!
2. *Improved system checking tools for validating and checking your projects*
3. *Refactoring of how Django identifies and loads applications*

Want to read the full release notes? Check out the official changes from Django here.

The latest release came on May 1, 2015: v1.7.8

Django Install

Set up a development structure:

```
$ mkdir django17_project
$ cd django17_project
$ virtualenv env
$ source env/bin/activate
```

You should see `(env)` before your prompt, `(env)$`, indicating that your virtualenv is activated.

To deactivate the virtualenv:

```
$ deactivate
```

Then reactivate once you're ready to start developing again.

With your virtualenv activated, install Django with Pip:

```
$ pip install django==1.7.8
```

You can check the version by running the following commands:

```
$ python
>>> import django
>>> django.get_version()
'1.7.8'
>>>
```

Project setup

Setup Django project

```
$ django-admin.py startproject my_django17_project
```

This creates a new directory called “my_django17_project” with the basic Django directory and structures:

```
├── manage.py
└── my_django17_project
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

Version control

Before you start any developing, place your project under version control. First, add a *.gitignore* file within your “django17_project” directory, which prevent unnecessary files from being added to the git repository.

Add the following to the file:

```
env
*.DS_Store
*.pyc
__pycache__
```

Now initialize (or create) a new Git repo and add your changes to staging and then to the repo.

```
$ git init
$ git add -A
$ git commit -am "initial"
```

If you use GitHub, PUSH your files to your central repo as well.

Database settings

First, install MySQL-python so that Python can talk to MySQL:

```
$ pip install MySQL-python
```

Edit *settings.py* within your “my_django17_project” directory:

```
DATABASES = {
    'default': {
```

```
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'django_db',
        'USER': 'root',
        'PASSWORD': 'your_password',
    }
}
```

Create your database tables and set up a superuser:

```
$ cd my_django17_project
$ python manage.py migrate
$ python manage.py createsuperuser
```

Launch the development server:

```
$ python manage.py runserver
```

You should see the following output if all is well thus far:

```
Performing system checks...

System check identified no issues (0 silenced).
September 07, 2014 - 19:51:01
Django version 1.7, using settings 'my_django17_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Navigate to http://127.0.0.1:8000 and you should see the familiar light-blue “It worked!” page.

Kill the server by pressing CONTROL-C back in the terminal.

pip freeze

With the dependencies installed, use the following command to create a record of them, along with their subsequent versions, within the “my_django17_project” directory:

```
$ pip freeze > requirements.txt
```

This command comes in handy when you need to recreate your project from scratch. You can simply run `pip install -r requirements.txt` to install all your project’s dependencies.

Commit your new changes to Git.

Set up your Django app

Create your new app:

```
$ python manage.py startapp myapp
```

Your project structure should now look like this:

```
├── manage.py
├── my_django17_project
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── myapp
    ├── __init__.py
    ├── admin.py
    ├── migrations
    │   └── __init__.py
    ├── models.py
    ├── tests.py
    └── views.py
```

Update the INSTALLED_APPS in your *settings.py*:

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myapp',
)
```

Fire up the server (`python manage.py runserver`), and then navigate to http://127.0.0.1:8000/admin. Enter your superuser's username and password to login to the admin page. This is just a sanity check to ensure that all is working.

Commit to Git. For example:

```
$ git add -A
$ git commit -am "updated settings, created app, boom"
```

All set. Let's create a basic app!

Django 1.8

First released on April 1, 2015, the most notable new features include:

1. *New PostgreSQL specific functionality*: This added new PostgreSQL-specific ModelFields – the ArrayField, HStoreField, and Range Fields. Check out our blog post, Fun With Django's New Postgres Features, for more info.
2. Multiple template engines: Now you can use Jinja!

Want to read the full release notes? Check out the official changes from Django here.

The latest release came on May 1, 2015: v1.8.1

Django Install

Set up a development structure:

```
$ mkdir django18_project
$ cd django18_project
$ virtualenv env
$ source env/bin/activate
```

You should see `(env)` before your prompt, `(env)$`, indicating that your virtualenv is activated.

To deactivate the virtualenv:

```
$ deactivate
```

Then reactivate once you're ready to start developing again.

With your virtualenv activated, install Django with Pip:

```
$ pip install django==1.8.1
```

You can check the version by running the following commands:

```
$ python
>>> import django
>>> django.get_version()
'1.8.1'
>>>
```

Project setup

Setup Django project

```
$ django-admin.py startproject my_django18_project
```

This creates a new directory called “my_django18_project” with the basic Django directory and structures:

```
├── manage.py
└── my_django17_project
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

Version control

Before you start any developing, place your project under version control. First, add a *.gitignore* file within your “django18_project” directory, which prevent unnecessary files from being added to the git repository.

Add the following to the file:

```
env
*.DS_Store
*.pyc
__pycache__
```

Now initialize (or create) a new Git repo and add your changes to staging and then to the repo.

```
$ git init
$ git add -A
$ git commit -am "initial"
```

If you use GitHub, PUSH your files to your central repo as well.

Database settings

First, install MySQL-python so that Python can talk to MySQL:

```
$ pip install MySQL-python
```

Edit *settings.py* within your “my_django17_project” directory:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'django_db',
```

```
        'USER': 'root',
        'PASSWORD': 'your_password',
    }
}
```

Create your database tables and set up a superuser:

```
$ cd my_django18_project
$ python manage.py migrate
$ python manage.py createsuperuser
```

Launch the development server:

```
$ python manage.py runserver
```

You should see the following output if all is well thus far:

```
Performing system checks...

System check identified no issues (0 silenced).
May 19, 2015 - 09:52:02
Django version 1.8, using settings 'my_django18_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Navigate to http://127.0.0.1:8000 and you should see the familiar light-blue “It worked!” page.

Kill the server by pressing CONTROL-C back in the terminal.

pip freeze

With the dependencies installed, use the following command to create a record of them, along with their subsequent versions, within the “my_django17_project” directory:

```
$ pip freeze > requirements.txt
```

This command comes in handy when you need to recreate your project from scratch. You can simply run `pip install -r requirements.txt` to install all your project’s dependencies.

Commit your new changes to Git.

Create an App

Let's create a basic, single page app that uses markdown to display text.

Setup

With virtualenv activated, install the following package to render markdown.

```
$ pip install django-markdown-deux
```

Update the requirements file to include the new dependency:

```
$ pip freeze > requirements.txt
```

Add "markdown_deux" to *settings.py*:

```
INSTALLED_APPS = (

  ... snip ...

  'markdown_deux',
)
```

Create a new directory within the "my_django1*_project" directory called "templates", and then add the path to the* settings.py* file:

```
import os
SETTINGS_DIR = os.path.dirname(__file__)
PROJECT_PATH = os.path.join(SETTINGS_DIR, os.pardir)
PROJECT_ROOT = os.path.abspath(PROJECT_PATH)
TEMPLATE_DIRS = (
    os.path.join(PROJECT_ROOT, 'templates'),
)
```

Your project structure should look like …

This for Django 1.5 and 1.6:

```
├── manage.py
├── my_django1*_project
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── myapp
│   ├── __init__.py
```

```
│   ├── migrations
│   │   ├── 0001_initial.py
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
└── templates
```

Or this for Django 1.7:

```
├── manage.py
├── my_django17_project
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── myapp
│   ├── __init__.py
│   ├── admin.py
│   ├── migrations
│   │   ├── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
└── templates
```

Views and URLs

We will be following the Model-View-Controller (MVC) architecture structure. Django projects are logically organized around this architecture. However, Django's architecture is slightly different in that the views act as the controllers. So, projects are actually organized in a Model-Template-Views architecture (MTV). Yes, this is confusing.

Views

Add the following code to the *views.py* file:

```
from django.shortcuts import render_to_response
from django.template import RequestContext


def index(request):
    return render_to_response('index.html')
```

This function takes a parameter, `request`, which is an object that has information about the user requesting the page from the browser. The function's response is to simply render the *index.html* template.

URLs

Next, we need to add a new pattern to the *urls.py* file:

```
urlpatterns = patterns(
    '',
    url(r'^admin/', include(admin.site.urls)),
    url(r'^$', 'myapp.views.index')
)
```

For more information on how to generate url patters, please view from the official Django documentation.

Templates

Finally, we need to create the *index.html* template. Create a new file called *index.html* within the templates directory, and add the code found here.

Fire up the server. How does that look? Let's add some styles.

Replace the previous code with code here.

Better?

Conclusion

Finish up adding styles and content as you see fit. Then return to the Workflow section to finish the development process. If you need help, please purchase Real Python. Or just do it anyway to help support this open source project. Thank you. Cheers!.

Summary Workflow

The following is a basic workflow that you can use as a quick reference for developing a Django Project.

Setup

1. Within a new directory, create and activate a virtualenv.
2. Install Django.
3. Create your project: `django-admin.py startproject <name>`
4. Create a new app: `python manage.py startapp <appname>`
5. Add your app to the INSTALLED_APPS tuple.

Add Basic URLs and Views

1. Map your Project's *urls.py* file to the new app.

2. In your App directory, create a *urls.py* file to define your App's URLs.
3. Add views, associated with the URLs, in your App's *views.py*; make sure they return a `HttpResponse` object. Depending on the situation, you may also need to query the model (database) to get the required data back requested by the end user.

Templates and Static Files

1. Create a *templates* and *static* directory within your project root.
2. Update *settings.py* to include the paths to your templates.
3. Add a template (HTML file) to the *templates* directory. Within that file, you can include the static file with – `{% load static %}` and `{% static "filename" %}`. Also, you may need to pass in data requested by the user.
4. Update the *views.py* file as necessary.

Models and Databases

1. Update the database engine to *settings.py* (if necessary, as it defaults to SQLite).
2. Create and apply a new migration.
3. Create a super user.
4. Add an *admin.py* file in each App that you want access to in the Admin.
5. Create your models for each App.
6. Create and apply a new migration. (Do this whenever you make *any* change to a model).

Forms

1. Create a *forms.py* file at the App to define form-related classes; define your ModelForm classes here.
2. Add or update a view for handling the form logic – e.g., displaying the form, saving the form data, alerting the user about validation errors, etc.
3. Add or update a template to display the form.
4. Add a `urlpattern` in the App's *urls.py* file for the new view.

User Registration

1. Create a UserForm
2. Add a view for creating a new user.
3. Add a template to display the form.
4. Add a urlpattern for the new view.

User Login

1. Add a view for handling user credentials.
2. Create a template to display a login form.
3. Add a urlpattern for the new view.

Setup the template structure

1. Find the common parts of each page (i.e., header, sidebar, footer).
2. Add these parts to a base template
3. Create specific. templates that inherent from the base template.