Django Base Site Documentation

Brent O'Connor

May 25, 2019

Contents

CHAPTER 1

Contents

1.1 Setup and Usage Instructions

1.1.1 Setup Instructions

Note: If you want to use Docker then user the *Docker Instructions* or if you want to use Vagrant then use the *Vagrant Instructions*. Otherwise, continue with these instructions.

Before you begin make sure you've setup and installed Pipenv and Virtualenvwrapper. You can use the Django base site without Virtualenvwrapper however you won't be able to use the `workon` command.

Change the directory to where you want keep your django projects.

```
$ cd ~/Sites
```

In the same directory run the following commands to download the template.

```
$ export PROJECT_NAME=example
$ curl -LOk https://github.com/epicserve/django-base-site/archive/master.zip && unzip␣
↪master
$ mv django-base-site-master $PROJECT_NAME
$ cd $PROJECT_NAME
```

Setup your virtualenv with pipenv and install the project requirements.

```
$ pipenv install --dev --python $(which python3)
$ export SECRET_KEY=$(python -c "import random; print(''.join(random.SystemRandom().
↪choice('abcdefghijklmnopqrstuvwxyz0123456789%^&*(-_=+)') for i in range(50)))")
$ cat > .env <<EOF
DEBUG=on
SECRET_KEY='$SECRET_KEY'
EMAIL_HOST='smtp.planetspaceball.com'
```

(continues on next page)

(continued from previous page)

```
EMAIL_HOST_USER='skroob@planetspaceball.com'
EMAIL_HOST_PASSWORD='12345'
DEFAULT_FROM_EMAIL="President Skroob <skroob@planetspaceball.com>"
EOF
$ pipenv shell
```

Remove all unnecessary example configs and template files.

```
$ make clean
```

Setup your database:

```
$ chmod +x manage.py
$ ./manage.py migrate
```

At this point your base site should be setup and you can now run your dev server.

```
$ ./manage.py runserver
```

1.1.2 Usage

Running the development server

After following the *Setup Instructions* you can work on your project again by doing the following.

```
$ workon example
$ ./manage.py runserver
```

How to edit and build the SCSS and Javascript source files:

First from the root of the project install gulp and the node requirements. This requires that your first install node.

```
$ npm install -g gulp
$ npm install
```

Then you can run `gulp` which will watch for changes to your SCSS and Javascript files changes in the `./src` directory.

```
$ gulp
```

1.2 Using Vagrant

> **Warning:** Vagrant support has been deprecated and is no longer tested and will eventually be removed.

If you want to try out the `django-base-site` using Vagrant then you first need to install vagrant of course. Then you can do the following to get things running.

```
$ TARGET_DIR=~/Sites/ BRANCH=master PROJECT_NAME=example
$ cd $TARGET_DIR
$ curl -L "https://github.com/epicserve/django-base-site/archive/$BRANCH.zip" | tar␣
↪zx -C $TARGET_DIR && mv "django-base-site-$BRANCH" $PROJECT_NAME
```

(continues on next page)

(continued from previous page)

```
$ cd $PROJECT_NAME
$ vagrant up
```

After running `vagrant up` you'll need to wait (~5 minutes) while your new Vagrant box is provisioned.

When it has finished provisioning your Vagrant box you should be able to run `vagrant ssh` to ssh to your new box. From there you can run `drs` to start the Django runserver. You should now have the django development server running in your Vagrant box. You can now open http://127.0.0.1:8000 in your local web browser and you should be able to see the message, "You've successfully setup a Django base site. Start Coding!".

Now you can just edit your `django-base-site` files locally in the `~/Sites /example` directory and Django's runserver that's running in the Vagrant box will detect any changes that are made.

1.3 Using Docker

If you want to try out the `django-base-site` using Docker then you first need to install docker. Then you can go through the following steps.

1. First download the `django-base-site` wherever you want your new project.

```
$ TARGET_DIR=~/Sites/ BRANCH=master PROJECT_NAME=example && \
cd $TARGET_DIR && \
curl -L "https://github.com/epicserve/django-base-site/archive/$BRANCH.zip" | tar zx -
↪C $TARGET_DIR && mv "django-base-site-$BRANCH" $PROJECT_NAME &&  \
cd $PROJECT_NAME
```

2. Create your `.env` file.

```
$ export SECRET_KEY=$(python -c "import random; print(''.join(random.SystemRandom().
↪choice('abcdefghijklmnopqrstuvwxyz0123456789%^&*(-_=+)') for i in range(50)))") && \
cat > .env <<EOF
DEBUG=on
DATABASE_URL=postgres://postgres@db:5432/postgres
SECRET_KEY='$SECRET_KEY'
EMAIL_HOST='smtp.planetspaceball.com'
EMAIL_HOST_USER='skroob@planetspaceball.com'
EMAIL_HOST_PASSWORD='12345'
DEFAULT_FROM_EMAIL="President Skroob <skroob@planetspaceball.com>"
ALLOWED_HOSTS=*
INTERNAL_IPS=192.168.99.100,127.0.0.1,0.0.0.0,localhost,172.18.0.1
REDIS_HOST=redis
EOF
```

3. Build your service images.

```
$ docker-compose build
```

4. Run migrations and Create a super user.

```
$ docker-compose run web python manage.py migrate && \
docker-compose run web python manage.py createsuperuser
```

5. Run the Django runserver.

```
$ docker-compose up
```

1.3.1 Debugging

ipdb

Set a trace like you normally do in your code and then go to that view in your browser and then stop `docker-compose up`.

```
import ipdb; ipdb.set_trace()
```

Then you can run the following to work interactive debugging shell.

```
docker-compose run --service-ports web
```

PyCharm

- Follow Jetbrain's guide.
- Make sure you pull up the site using http://localhost:8000/ instead of http://127.0.0.1:8000/.
- You'll also need to add 172.18.0.1 to your INTERNAL_IPS.

1.3.2 Running Gulp

- When you run `docker-compose up` it starts the node service which should run gulp.
- If you want to run gulp by itself, you can run it with a command like, `docker-compose run --rm node`.

1.3.3 Common Gotchas

- You need to start your runserver using *docker-compose up* instead of *docker-compose run web python manage.py runserver 0.0.0.0:8000* or you won't be able to access your site from your browser.
- Installing *django-debug-toolbar* can ignore the Django version you've specified in your *Pipfile* and instead Django 2 because django-debug-toolbar uses "Django" with a capital D in it's requirements when other packages use "django" in lowercase. To work around this install everything except *django-debug-toolbar* and then added it last to your *Pipfile*.

1.3.4 Common Commands

Command	Description
docker-compose up	Starts up all of your services according to how they were defined in your docker-compose.yml file.
docker-compose down	Stops containers and removes containers, networks, volumes, and images created by *up*.
docker volume ls	List the volumes that have been created

1.3.5 References

- A Brief Intro to Docker for Djangonauts

1.4 Using AWS S3 for Static and Media Files

If you want to use AWS S3 for storing your static and media files then you can follow these steps to get things setup and going.

1.4.1 Setup an AWS User and S3 Bucket

1. Login to your AWS Console and create an IAM user for your Django project, making sure to only give the user programmatic access. Make sure you save the public access key and the secret key, because you'll need those for your Django settings.
2. Create an S3 Bucket. If you want to eventually point a CNAME DNS record to this bucket then make sure your bucket name is a fully qualified domain name.
3. Give the user you created access to the bucket. Create a policy like the following and add it to your user. You could name it Something like, MyDjangoProjectNameS3Access.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::django-base-site"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:PutBucketCORS",
                "s3:DeleteObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::django-base-site/*"
        }
    ]
}
```

4. Add the following CORS configuration to your bucket.

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<CORSRule>
    <AllowedOrigin>*</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
    <AllowedMethod>HEAD</AllowedMethod>
    <MaxAgeSeconds>3000</MaxAgeSeconds>
```

(continues on next page)

(continued from previous page)

```
    <AllowedHeader>Authorization</AllowedHeader>
</CORSRule>
</CORSConfiguration>
```

1.4.2 Update your Django Project Settings

1. Add the following to your Pipfile. Update the version numbers for whatever is current.

```
django-storages = "<1.7,>=1.6.5"
"boto3" = "<1.6,>=1.5.8"
```

2. Install the requirements.

```
$ pipenv install
```

3. Add `storages` to your `INSTALLED_APPS`.

```
INSTALLED_APPS = [
    ...
    'storages',
]
```

4. Add the following to your .env settings.

```
DEFAULT_FILE_STORAGE=apps.base.storage.MediaS3Storage
STATICFILES_STORAGE=apps.base.storage.StaticS3Storage
AWS_ACCESS_KEY_ID=<your key goes here>
AWS_SECRET_ACCESS_KEY=<your secret key goes here>
AWS_STORAGE_BUCKET_NAME=<your bucket name goes here>
```

1.4.3 Usage

If you set the `DEFAULT_FILE_STORAGE` to apps.base.storage.MediaS3Storage in the `.env` file, your django project will use Django Storages to save your static and media files to S3. If you want you can use your local filesystem during development, then just make sure you don't set the `DEFAULT_FILE_STORAGE` setting locally and only set it on your production instances.

After every static file change, you'll have to run `manage.py collectstatic` to upload your static files to S3. This usually is done programmatically using a tool like Fabric.

1.5 Install Pre-commit Hook

After you have setup your project by following the *Setup and Usage Instructions*, it's a good idea to install the flake8 pre-commit hook, which will prevent you from committing code with lint errors.

To install it, run the following in the root of your project:

```
$ flake8 --install-hook
```

Then run the following to figure out your virtualenv python location:

```
$ which python
```

Use path you just got to replace `#!/usr/bin/env python` with `#!/path/to/virtualenvs/example/bin/python` in `.git/hooks/pre-commit`

1.6 Website Pre-Launch Check List

- Make a favicon
- Test all pages in all major browsers
- Check all meta data
- Check to make sure you have a styled 404 and 500 error pages
- Make sure you're using page caching
- Make sure you have google Analytics installed
- Add a print style sheet
- Proofread content
- Double check your links (http://validator.w3.org/checklink/)
- Test all forms for validation and functionality
- Optimize the site for performance, http://developer.yahoo.com/performance/rules.html

CHAPTER 2

About

The Django Base Site is a Django site that is built using the best Django practices and comes with all the common Django packages that you need to jumpstart your next project.

CHAPTER 3

Install Requirements

Before setting up a new project make sure you have the following installed:

- Python 3.5 or newer
- Pipenv
- virtualenv

It's not a requirement, but it is recommended that you install Python using Pyenv with the virtualenvwrapper plugin.

CHAPTER 4

Features

- Bootstrap 4
- Celery
- Coverage
- Custom User Model
- Django 2
- Django Compressor
- Django Crispy Forms
- Django Debug Toolbar
- Django REST framework
- Django-allauth
- Django-environ for 12factor inspired environment variables
- Docker Support
- Gulp for building SASS and JS with Browserify for requiring modules and Babel for transpiling ES6/ES2015.
- Pipenv
- Vagrant Support
- Sample configs for Apache, Gunicorn, Nginx and Upstart

CHAPTER 5

Contribute

1. Look for an open issue or create new issue to get a dialog going about the new feature or bug that you've discovered.
2. Fork the repository on Github to start making your changes to the master branch (or branch off of it).
3. Make a pull request.