bm Documentation

Release 0.1

jawahar

Aug 12, 2019

Contents

Contents:

CHAPTER 1

BM

budget management project

Note: Remember: use *dev-base.html'(by renaming it to 'base.html*) for development coz it is consitence for development.

1.1 Settings

Moved to settings.

1.2 Basic Commands

1.2.1 Setting Up Your Users

- To create a **normal user account**, just go to Sign Up and fill out the form. Once you submit it, you'll see a "Verify Your E-mail Address" page. Go to your console to see a simulated email verification message. Copy the link into your browser. Now the user's email should be verified and ready to go.
- To create an **superuser account**, use this command:

```
$ python manage.py createsuperuser
```

For convenience, you can keep your normal user logged in on Chrome and your superuser logged in on Firefox (or similar), so that you can see how the site behaves for both kinds of users.

1.2.2 Test coverage

To run the tests, check your test coverage, and generate an HTML coverage report:

```
$ coverage run manage.py test
$ coverage html
$ open htmlcov/index.html
```

Running tests with py.test

```
$ py.test
```

1.2.3 Live reloading and Sass CSS compilation

Moved to Live reloading and SASS compilation.

1.2.4 Celery

This app comes with Celery.

To run a celery worker:

```
cd bm
celery -A bm.taskapp worker -l info # may cause heroku(worker) crash.
celery -A b.taskapp worker --loglevel=info
```

Please note: For Celery's import magic to work, it is important *where* the celery commands are run. If you are in the same folder with *manage.py*, you should be right.

1.2.5 Email Server

In development, it is often nice to be able to see emails that are being sent from your application. If you choose to use MailHog when generating the project a local SMTP server with a web interface will be available.

To start the service, make sure you have nodejs installed, and then type the following:

```
$ npm install
$ grunt serve
```

(After the first run you only need to type `grunt serve`) This will start an email server that listens on `127.0.0.1:1025` in addition to starting your Django project and a watch task for live reload.

To view messages that are sent by your application, open your browser and go to `http://127.0.0.1:8025`

The email server will exit when you exit the Grunt task on the CLI with Ctrl+C.

1.2.6 Sentry

Sentry is an error logging aggregator service. You can sign up for a free account at https://sentry.io/signup/?code=cookiecutter or download and host it yourself. The system is setup with reasonable defaults, including 404 logging and integration with the WSGI.

Note: Important part for working in production.

1.2.7 Cross Domain Name/Client Side Domain Name

Must set *BM_CLIENT_CROSS_DOMAIN_NAME* for active cross domain in client and server side communication.

1.2.8 Client Side Redirection

BM_CLIENT_PASSWORD_RESET_URL set the url which help in *restting* password.

Note: Default redirection url is *reset*.

CHAPTER 2

Install

Better to Django CookieCutter in install in locally.

CHAPTER 3

Global Environment

Global environment is an important part for the any apps one way or other. These environment is classified into type.

This project uses *django cookiecutter* as its base template. Follow other setting Cookiecutter Settings to know more.

3.1 Must Set Enviroment

Remeber the intercommunication between client and server must be in secure connection.

Name	Detail desciption
BM_CLIENT_CROSS_DOMAIN_NAME	Domain name of the client site Eg: on hosting in github(github site) it MUST set as https://userName.github. com.[1]
Open Weather Map	
BM_OPEN_WEATHER_MAP_API	Get api key for the openweather website by signup
BM_DB_CONN_MAX_AGE(0mins)	Setting max connection timeout for DB.

3.2 Must Review Enviroment

This enviroment is kind of option but if misconfig can cause lots of headpain.

3.3 Optional Environment

The below environment adds the given django 3rd party or local apps(which is consiter as optional) to *INSTALLED_APPS*.

[1] For now, only one domain is allowed to set.

Name	Detail desciption
BM_OPTIONAL_BASE_APPS	Add the given apps to base
BM_OPTIONAL_LOCAL_APPS	Add the given apps to local
BM_OPTIONAL_TEST_APPS	Add the given apps to test
BM_OPTIONAL_PRODUCTION_APPS	Add the given apps to production
BM_CURRENT_USER_UPLOAD _CACHE_TIMEOUT(default 90)	This is used to set cache time out(in seconds)

CHAPTER 4

Packages

Package is storage place for the spending amount and its after maths process.

Contents:

4.1 MonthBudgetAmount

Month Budget Amount is the limit set by the user for setting up margin limit in the spending amount in current month.

4.2 Packages Setting

..note:: Do remeber in development mode on creating super user. The package setting object will not be created automatically. Please create manully by using *localdata* command under **'**fixtures/package_settings.json'(set corresponding user's id number properly).

Package setting is model object which contain the nessary data for the controlling package model.

4.3 Upload Flat Files

Uploading flat file (such as Excel, CSV, etc) with specific columns and some data.

Warning: For time begin paytm uploading is supported.

`bm.packages.flat_file_interface.base_excel_interface.BaseExcelClass` is a abstract class. By using this class a default pandas interface has been created.

CHAPTER 5

Weather

Weather package is the interface which use's Open Weather

Contents:

CHAPTER 6

Deploy

To deploy in production there three way can followed Pythonanywhere, Heroku and Docker.

Note: It is highly recommented to deploy with server which support ASIG such in Heroku to support Django Channel.

CHAPTER 7

Developing with Docker

You can develop your application in a Docker container for simpler deployment onto bare Linux machines later. This instruction assumes an Amazon Web Services EC2 instance, but it should work on any machine with Docker > 1.3 and Docker compose installed.

7.1 Setting up

Docker encourages running one container for each process. This might mean one container for your web server, one for Django application and a third for your database. Once you're happy composing containers in this way you can easily add more, such as a Redis cache.

The Docker compose tool (previously known as fig) makes linking these containers easy. An example set up for your Cookiecutter Django project might look like this:

```
webapp/ # Your cookiecutter project would be in here
    Dockerfile
    ...
database/
    Dockerfile
    ...
webserver/
    Dockerfile
    ...
production.yml
```

Each component of your application would get its own Dockerfile. The rest of this example assumes you are using the base postgres image for your database. Your database settings in *config/base.py* might then look something like:

```
DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.postgresql_psycopg2',
            'NAME': 'postgres',
            'USER': 'postgres',
```

(continues on next page)

(continued from previous page)

```
            'HOST': 'database',
            'PORT': 5432,
        }
    }
```

The Docker compose documentation explains in detail what you can accomplish in the *production.yml* file, but an example configuration might look like this:

```
database:
    build: database
webapp:
    build: webapp:
    command: /usr/bin/python3.4 manage.py runserver 0.0.0.0:8000 # dev setting
    # command: gunicorn -b 0.0.0.0:8000 wsgi:application # production setting
    volumes:
        - webapp/your_project_name:/path/to/container/workdir/
    links:
        - database
webserver:
    build: webserver
    ports:
        - "80:80"
        - "443:443"
    links:
        - webapp
```

We'll ignore the webserver for now (you'll want to comment that part out while we do). A working Dockerfile to run your cookiecutter application might look like this:

```
FROM ubuntu:14.04
ENV REFRESHED_AT 2015-01-13

# update packages and prepare to build software
RUN ["apt-get", "update"]
RUN ["apt-get", "-y", "install", "build-essential", "vim", "git", "curl"]
RUN ["locale-gen", "en_GB.UTF-8"]

# install latest python
RUN ["apt-get", "-y", "build-dep", "python3-dev", "python3-imaging"]
RUN ["apt-get", "-y", "install", "python3-dev", "python3-imaging", "python3-pip"]

# prepare postgreSQL support
RUN ["apt-get", "-y", "build-dep", "python3-psycopg2"]

# move into our working directory
# ADD must be after chown see http://stackoverflow.com/a/26145444/1281947
RUN ["groupadd", "python"]
RUN ["useradd", "python", "-s", "/bin/bash", "-m", "-g", "python", "-G", "python"]
ENV HOME /home/python
WORKDIR /home/python
RUN ["chown", "-R", "python:python", "/home/python"]
ADD ./ /home/python

# manage requirements
ENV REQUIREMENTS_REFRESHED_AT 2015-02-25
RUN ["pip3", "install", "-r", "requirements.txt"]
```

(continues on next page)

(continued from previous page)

```
# uncomment the line below to use container as a non-root user
USER python:python
```

Running *sudo docker-compose -f production.yml build* will follow the instructions in your *production.yml* file and build the database container, then your webapp, before mounting your cookiecutter project files as a volume in the webapp container and linking to the database. Our example yaml file runs in development mode but changing it to production mode is as simple as commenting out the line using *runserver* and uncommenting the line using *gunicorn*.

Both are set to run on port *0.0.0.0:8000*, which is where the Docker daemon will discover it. You can now run *sudo docker-compose -f production.yml up* and browse to *localhost:8000* to see your application running.

7.2 Deployment

You'll need a webserver container for deployment. An example setup for Nginx might look like this:

```
FROM ubuntu:14.04
ENV REFRESHED_AT 2015-02-11

# get the nginx package and set it up
RUN ["apt-get", "update"]
RUN ["apt-get", "-y", "install", "nginx"]

# forward request and error logs to docker log collector
RUN ln -sf /dev/stdout /var/log/nginx/access.log
RUN ln -sf /dev/stderr /var/log/nginx/error.log
VOLUME ["/var/cache/nginx"]
EXPOSE 80 443

# load nginx conf
ADD ./site.conf /etc/nginx/sites-available/your_cookiecutter_project
RUN ["ln", "-s", "/etc/nginx/sites-available/your_cookiecutter_project", "/etc/nginx/
↪sites-enabled/your_cookiecutter_project"]
RUN ["rm", "-rf", "/etc/nginx/sites-available/default"]

#start the server
CMD ["nginx", "-g", "daemon off;"]
```

That Dockerfile assumes you have an Nginx conf file named *site.conf* in the same directory as the webserver Dockerfile. A very basic example, which forwards traffic onto the development server or gunicorn for processing, would look like this:

```
# see http://serverfault.com/questions/577370/how-can-i-use-environment-variables-in-
↪nginx-conf#comment730384_577370
upstream localhost {
    server webapp_1:8000;
}
server {
    location / {
        proxy_pass http://localhost;
    }
}
```

Running *sudo docker-compose -f production.yml build webserver* will build your server container. Running *sudo docker-compose -f production.yml up* will now expose your application directly on *localhost* (no need to specify the port number).

7.3 Building and running your app on EC2

All you now need to do to run your app in production is:

- Create an empty EC2 Linux instance (any Linux machine should do).
- Install your preferred source control solution, Docker and Docker compose on the news instance.
- Pull in your code from source control. The root directory should be the one with your *production.yml* file in it.
- Run *sudo docker-compose -f production.yml build* and *sudo docker-compose -f production.yml up*.
- Assign an Elastic IP address to your new machine.
- Point your domain name to the elastic IP.

Be careful with Elastic IPs because, on the AWS free tier, if you assign one and then stop the machine you will incur charges while the machine is down (presumably because you're preventing them allocating the IP to someone else).

7.4 Security advisory

The setup described in this instruction will get you up-and-running but it hasn't been audited for security. If you are running your own setup like this it is always advisable to, at a minimum, examine your application with a tool like OWASP ZAP to see what security holes you might be leaving open.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search