Sahil Diwan - Local Diner

Blog PostsCheck out my posts

Find me on...

GitHubCheck out my code

TwitterCheck out my life

WebsiteCheck out my work

Making a Flask app using a PostgreSQL database and deploying to Heroku

This blog post is about creating a simple pre-registration page using the best (in my opinion) micro web-development framework for Python, Flask. We will be connecting our pre-registation app to a PostgreSQL database locally and in the cloud. Once we have our Flask app running locally, I will show you how to successfully deploy it to Heroku.

Notes:

- The source code for this project can be found here.
- A live demo of this project can be found on Heroku here.
- I am running Mac OS X and this tutorial (at least installation) will be specific.

Let’s get started.

`1.)` Installation and environment setup

We will need to install the following:

- pip
- virtualenv
- postgres.app
- And obviously Flask + Flask-SQLAlchemy

First, install pip by running the command

```
$ sudo easy_install pip
```

in your terminal.

Then, install virtualenv by running the command

```
$ sudo pip install virtualenv
```

Now, we can create our project directory and set up our virtual environment.

```
$ cd your/fav/directory
$ mkdir lovelypreregpage
```

To set up a virtual environment

```
$ cd lovelypreregpage
$ virtualenv venv
```

You should now have a `venv` folder within your `lovelypreregpage` project.

To start your virtual environment run the command

```
$ . venv/bin/activate
```

Next, we can begin setting up our database. Go to postgresapp.com and download/install `postgres.app` just like any other application. `Postgres.app` is super easy to set up, but don't forget to add the following to your `.bash_profile` found in your home directory

```
# Postgres.app
export PATH="/Applications/Postgres.app/Contents/Versions/9.3/bin:$PATH"
```

All we have to do now is launch `Postgres.app` (found in applications) and hit `open psql`. In your `bash` terminal, create a new database by running the command

```
$ createdb pre-registration
```

Now, open up the postgres terminal launched when hitting `open psql` and run the command

```
=# \list
```

You should see the database `pre-registration` listed in the table.

`2.)` Setting up our actual Flask app

First, create the following files and folders

```
$ mkdir static
$ mkdir templates
$ touch app.py
$ touch .gitignore
```

Open up your `.gitignore` file and add the following

```
venv
*.pyc
.DS_Store
```

Now, go into your `static` folder and add following two folders and their respective files

- `css` –> bootstrap.min.css, styles.css (found here)
- `js` –> bootstrap.min.js, scripts.js (found here)

Then, go into your `templates` folder and add the following two HTML files

- `index.html` found here
- `success.html` found here

A couple things to notice in the HTML files; links to static files in Flask are done a special way, and calling a method from a form action is done similarly.

Let’s install some Flask stuff now. Run the following commands in your terminal

```
$ pip install Flask
$ pip install psycopg2
$ pip install Flask-SQLAlchemy
```

Next, we have to write the python code to do things and connect our database. Add the following code to your `app.py` file

```
from flask import Flask, render_template, request
from flask.ext.sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://localhost/pre-registration'
db = SQLAlchemy(app)

# Create our database model
class User(db.Model):
    __tablename__ = "users"
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), unique=True)

    def __init__(self, email):
        self.email = email
    def __repr__(self):
        return '<E-mail %r>' % self.email

# Set "homepage" to index.html
@app.route('/')
def index():
    return render_template('index.html')

# Save e-mail to database and send to success page
@app.route('/prereg', methods=['POST'])
def prereg():
    email = None
    if request.method == 'POST':
        email = request.form['email']
        # Check that email does not already exist (not a great query, but works)
        if not db.session.query(User).filter(User.email == email).count():
            reg = User(email)
            db.session.add(reg)
            db.session.commit()
            return render_template('success.html')
    return render_template('index.html')

if __name__ == '__main__':
    app.debug = True
    app.run()
```

To create our database based off our model, run the following commands

```
$ python
>>> from app import db
>>> db.create_all()
>>> exit()
```

Now, run `python app.py` in your terminal. You should see `* Running on http://127.0.0.1:5000/`, go to http://127.0.0.1:5000/ and experience beauty.

*Bonus, a great postgres GUI client for Mac OS X is `Induction`. Just go here to download and install. Launch the app and enter `postgres://localhost/pre-registration` as your database URL. Next, browse your data. :)

`3.)` Deploy to Heroku

First, go to Heroku and install the Heroku toolbelt just like any other application. While you are at it, create an account on Heroku if you do not have one already. Make sure to add your ssh key(s). Then, before you can create a Heroku app from your existing project we need to turn it into a git repo. Go here to install git on your machine.

Let’s go ahead and create a couple more files we will need for deploying to Heroku

```
$ touch Procfile
$ touch requirements.txt
```

Now, run the following to commands with your virtual environment running

```
$ pip install gunicorn
$ pip install Flask-Heroku
$ pip freeze > requirements.txt
```

That last command is putting all our app requirements into `requirements.txt` so Heroku knows what it needs to install.

Add the following line to your `Procfile` which tells heroku what python file to execute

```
web: gunicorn app:app
```

Open up your `app.py file and edit your imports and database connections to look like this

```
from flask import Flask, render_template, request
from flask.ext.sqlalchemy import SQLAlchemy

from flask.ext.heroku import Heroku

app = Flask(__name__)
#app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://localhost/pre-registration'
heroku = Heroku(app)
db = SQLAlchemy(app)
```

Since we have Heroku and git installed, run the following commands while in your `lovelypreregpage` project

```
$ git init
$ git add .
$ git commit -m "initial commit"
$ heroku create name-of-your-app
```

The last command will automatically create a Heroku app using the name you give. Next, run the following command to push your Flask app up to Heroku

```
$ git push heroku master
```

Setting up a PostgreSQL database on Heroku is a lot like setting up a PostgreSQL database locally. Run the following commands

```
$ heroku addons:add heroku-postgresql:dev
$ heroku pg:promote HEROKU_POSTGRESQL_COLOR_URL
```

*It is important to switch `HEROKU_POSTGRESQL_COLOR_URL` with the color shown to you after running the `heroku addons:add heroku-postgresql:dev` command.

Now, run the following commands to create our database tables exactly as we did before

```
$ heroku run python
>>> from app import db
>>> db.create_all()
>>> exit()
```

Finally, go to `name-of-your-app.herokuapp.com` and see your working pre-registration page! It is not the sexiest website, but now that you know how to make it you can spend time on front-end shenanigans.

To see the live demo of this tutorial go to flask-postgres-heroku.herokuapp.com.

*Bonus, Heroku allows you to create Dataclips, which is similar to `Induction`. Create a `Dataclip` and run the query

```
select * from users
```

to see all the users that have pre-registered for your awesome app.