Flask Cheat Sheet

A cheat-sheet for creating web apps with the Flask framework using the Python language.

Contents

Creating a Simple App

- Create a module called `app.py` :

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
        return 'Hello, World!'

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

Structuring an Application with Blueprints

- Example project tree for a project called `project` :

```
run.py
project/
        __init__.py
        config.py
        forms.py
        models.py
        admin/
                __init__.py
                routes.py
        main/
                __init__.py
                routes.py
        templates/
                index.html
        static/
                css/
                        style.css
```

- In `run.py` :

```
from threechan import app

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

- In `project/__init__.py` :

```
from flask import Flask
from project.main.routes import main
from project.admin.routes import admin

app = Flask(__name__)

app.register_blueprint(main, url_prefix='/')
app.register_blueprint(admin, url_prefix='/admin')
```

- In `project/main/routes.py` :

```
from flask import Blueprint

main = Blueprint('main', __name__)

@main.route('/')
def index():
        return "Hello, World! This is the main page."
```

- In `rwochan/admin/routes.py` :

```
from flask import Blueprint

admin = Blueprint('admin', __name__)

@main.route('/')
def index():
        return "Hello, World! This is the admin page."
```

Creating Object-Based Configuration

- Create `project/config.py` :

```
class BaseConfig(object):
        SECRET_KEY = os.environ.get('SECRET_KEY') or 'abcdef123456'
        DEBUG = False
        TESTING = False

class DevelopmentConfig(BaseConfig):
        DEBUG = True
        TESTING = True

class TestingConfig
        DEBUG = False
        TESTING = True
```

- And then in `__init__.py` include:

```
from flask import Flask
from threechan import config

app = Flask(__name__)
app.config.from_object(config.DevelopmentConfig)
```

Using the Jinja2 Template Engine

- Rendering a template from `main/routes.py` :

```
from flask import Blueprint, render_template

@main.route('/')
def home():
        posts = [
                {
                        "body": "Hello, this is a post",
                        "timestamp": "This is a date and time",
                }
        ]

        return render_template("home.html", posts=posts)
```

- Using Jinja2 inside `templates/home.html` :

```
{% for posts in posts %}
        <div>
                {{ post.body }} <br>
                {{ post.timestamp }}
        </div>
{% endfor %}
```

- Using filters:

```
{{ body|upper }}
```

- Creating custom filters:

```
@main.template_filter('trim_upper')
def string_trim_upper(value):
        return value.strip().upper()
```

- Using `extends`:

in **index.html**

```
{% extends 'base.html' %}

{% block content %}
<p>
        Hello, world!
</p>
{% endblock %}
```

in **base.html**

```
<!DOCTYPE html>
<html lang="en">
        <head>
                <meta charset="UTF-8">
                <meta name="viewport" content="width=device-width initial-scale=1">
        </head>
        <body>
                {% blockcontent %}{% endblock %}
        </body>
</html>
```

- Using `includes`:

```
{% includes '_post.html' %}
```

- Adding a stylesheet from `templates/base.html`:

```
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
```

Creating Models with SQLAlchemy

- In `project/models.py`:

```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class Post(db.Model):
        id = db.Column(db.Integer, primary_key=True)
        body = db.Column(db.String(256))

        def __init__(self, body):
                self.body = body

        def __repr__(self):
                return "<Post({})>".format(self.id)
```

- In `project/__init__.py`

```
from project.models improt db

app = Flask(__name__)
app.config.from_object(config.DevelopmentConfig)
with app.app_context():
        db.init_app()
```

- In `project/config.py` :

```
import os
BASE_DIR = os.path.abspath(os.path.dirname(__name__))

class BaseConfig(object):
        # ...
        SQLALCHEMY_DATABASE_URI = 'sqlite:///' + os.path.join(BASE_DIR, 'data.db'
        SQLALCHEMY_TRACK_MODIFICATIONS = False
```

- Creating the database and interacting with models and queries inside the shell:

```
>>> from project import app, db
>>> from project.models import Post
>>> app.app_context().push()
>>> db.create_all()
>>> p = Post('Hello!')
>>> db.session.add(p)
>>> db.session.commit()
>>> posts = Post.query.all()
```

- To operate through `$ flask shell` , include the following in `project/__init__.py` :

```
from project.models import db, User, Post

@app.shell_context_processor
def make_shell_context():
        return {'app': app, 'db': db, 'User': User, 'Post': Post}
```

- For more info on `Flask-SQLAlchemy` : official flask-sqlalchemy guide.
- For more info on `contexts` : official flask-sqlalchemy contexts guide

Using Database Migrations

- Add the following to `project/__init__.py` :

```
from flask_migrate import Migrate

app = Flask(__name__)
app.config.from_object)config.DevelopmentConfig)
migrate = Migrate(app, db)
```

- To initialize the `migrations` folder:

```
$ flask db init
```

- To migrate:

```
$ flask db migrate
```

- To upgrade:

```
$ flask db upgrade
```

Creating a Login Manager

- In `models.py` include:

```
from flask_login import LoginManager, UserMixin

login_manager = LoginManager()

@login_manager.user_loader
def load_user(user_id):
        return User.get(user_id)
```

- in `project/__init__.py` include:

```
from project.models import login_manager

with app.app_context():
        login_manager.init_app(app)
```

Connecting to a MySQL database

- Although `SQLite` is fine for a simple development server, the databases of production servers are often better suited to a more full-featured DBMS like `MySQL` .
- Install `MySQL` :

```
$ sudo apt-get install mysql-server
```

If not prompted to enter a password:

```
$ sudo mysql_secure_installation
```

- And then:

```
$ sudo mysql -u root -p
```

```
CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON database_name.* TO 'newuser'@'localhost';
```

Access the `MySQL` shell:

```
$ mysql -u root -p
```

Note: `root` *of MySQL is not the same as the* `root` of the OS.

- Show databases with:

```
SHOW DATABASES;
```

- Create a database:

```
CREATE DATABASE database_name;
```

- Establish connection with `SQLAlchemy`, install `pymysql`:

```
$ pip install pymysql
```

- And within the config object in `config.py` include:

```
SQLALCHEMY_DATABASE_URI  =  'mysql+pymysql://root:password@localhost/database';
```

- For more standard commands: the Digital Ocean MySQL tutorial is useful.