Website

Mingming Li

First Created: Feb 20, 2023
Last Modified: March 17, 2023

Contents

List of Figures

List of Tables

Part I

Flask

Chapter 1

Installation

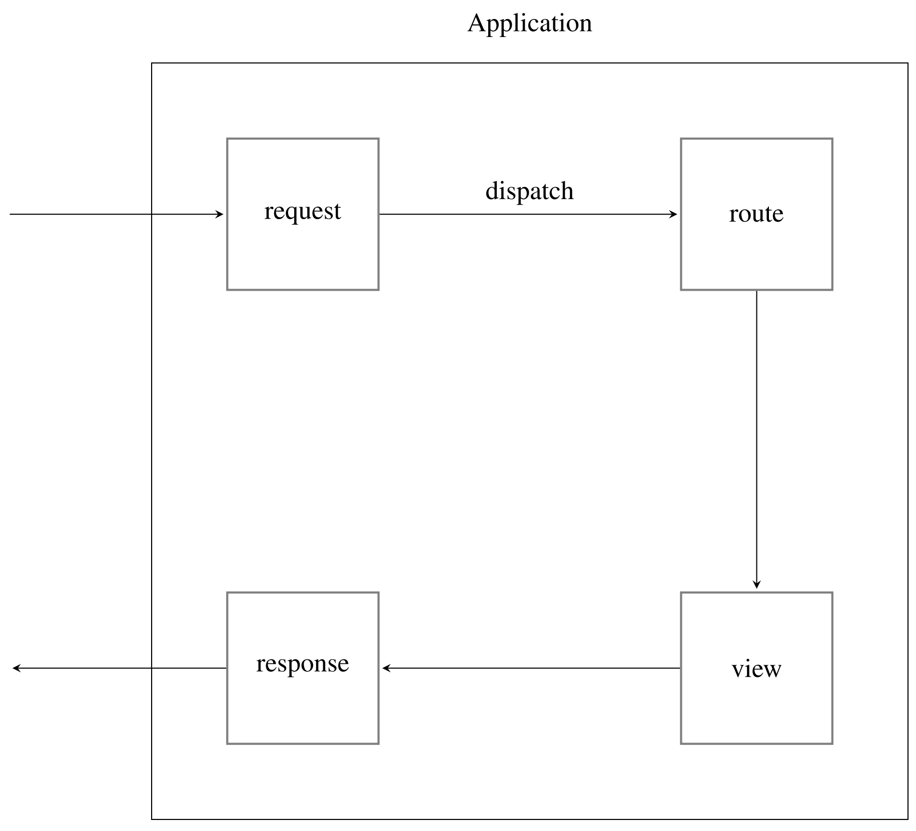
Flask is a lightweight WSGI web application framework. It provides a solid core with the basic services, while extensions provides the rest. Flask has three main dependencies. The routing, debugging, and Web Server Gateway Interface (WSGI) subsystems come from Werkzeug; the template support is provided by Jinja2; and the command-line integration comes from Click.

The easy way to install Flask is with the Python package manager **pip**.

```
pip install flask
```

Chapter 2

Application structure



2.1 Initialization

All Flask applications must create an application instance. The web server passes all requests it receives from clients to this object for handling, using a protocol called Web Server Gateway Interface (WSGI). The application instance is an object of class Flask, usually created as follows:

```
from flask import Flask
app = Flask(__name__)
```

2.2 Routes and view functions

The association between a URL and the function that handles it is called a route.

```
@app.route('/')
def index():
    return '<h1>Hello World!</h1>'
```

2.3 Responses

```
from flask import make_response
@app.route('/')
def index():
    response = make_response('<h1>This document carries a cookie!</h1>')
    response.set_cookie('answer', '42')
    return response
```

2.4 Development server

Flask applications include a development web server. To start the hello.py application:

```
flask run hello.py

# or
export FLASK_APP=hello.py
flask run

# to enable debug mode
flask run hello.py --debug
# or
export FLASK_DEBUG=1
flask run
```

2.5 Extensions

To use extensions:

1 Install the extension, for example

```
pip install flask-bootstrap
```

2 Initialize the extension.

```
from flask_bootstrap import Bootstrap
# ...
bootstrap = Bootstrap(app)
```

Here’s another method to initialize the extension:

```
bootstrap = Bootstrap()
bootstrap.init_app(app)
```

3 Use it in the template or in the code.

```
{% extends "bootstrap/base.html" %}
```

Chapter 3

Templates

A template is a file that contains the text of a response, with placeholder variables for the dynamic parts that will be known only in the context of a request. The process that replaces the variables with actual values and returns a final response string is called rendering. For the task of rendering templates, Flask uses a powerful template engine called Jinja2.

3.1 Rendering templates

Listing 3.1: templates/user.html

```
<h1>Hello, {{ name }}!</h1>
```

Listing 3.2: rendering a template

```
from flask import Flask, render_template

# ...

@app.route('/user/<name>')
def user(name):
    return render_template('user.html', name=name)
```

3.2 Variables

The `{{ name }}` construct used in the template shown in Listing 3.1 references a variable, a special placeholder that tells the template engine that the value that goes in that place should be obtained from data provided at the time the template is rendered.

3.3 Control structures

```
{% if condition %}
    ...
{% else %}
    ...
{% endif %}
{% for i in iterable %}
    ...
{% endfor %}



{% macro func(params) %}
    ...
{% endmacro %}


{% import 'macros.html' as macros %}


{% include 'common.html' %}



{% block block_name %}
    ...
{% endblock %}


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

In general, the code part is placed in the `{% %}`.

3.4 Links

Flask provides the `url_for()` helper function, which gener- ates URLs from the information stored in the application's URL map. This function can also be used to link static files.

```
{% block head %}
    {{ super() }}
    <link rel="shortcut icon" href="{{ url_for('static', filename='favicon.ico')
        }}" type="image/x-icon">
    <link rel="icon" href="{{ url_for('static', filename='favicon.ico') }}" type=
        "image/x-icon">
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='
        styles.css') }}">
    <script src="{{ url_for('static',filename='ckeditor/ckeditor.js') }}"></
        script>
{% endblock %}
```

Chapter 4

Extensions

4.1 Bootstrap

Bootstrap is an open-source web browser framework from Twitter that provides user interface components that help create clean and attractive web pages that are compatible with all modern web browsers used on desktop and mobile platforms.

Bootstrap is a client-side framework, so the server is not directly involved with it. All the server needs to do is provide HTML responses that reference Bootstrap's Cascading Style Sheets (CSS) and JavaScript files, and instantiate the desired user interface elements through HTML, CSS, and JavaScript code. this is in templates.

The extension is Flask-Bootstrap:

```
pip install flask-bootstrap
```

```
{% extends "bootstrap/base.html" %}
```

4.2 Moment

There is an excellent open source library written in JavaScript that renders dates and times in the browser called Moment.js. Flask-Moment is an extension for Flask applications that makes the integration of Moment.js into Jinja2 templates very easy.

```
pip install flask-moment
```

Listing 4.1: import the Moment.js library

```
{% block scripts %}
{{ super() }}
{{ moment.include_moment() }}
{% endblock %}
```

4.3 Web forms

The Flask-WTF extension makes working with web forms a much more pleasant experience. This extension is a Flask integration wrapper around the framework-agnostic WTForms package.

```
pip install flask-wtf
```

This extension does not need initialization, but need to have a secret key configured.

```
app = Flask(__name__)
app.config['SECRET_KEY'] = 'hard to guess string'
```

```
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField
from wtforms.validators import DataRequired

class NameForm(FlaskForm):
    name = StringField('What is your name?', validators=[DataRequired()])
    submit = SubmitField('Submit')
```

4.4 SQLAlchemy

Flask-SQLAlchemy is a Flask extension that simplifies the use of SQLAlchemy inside Flask applications. SQLAlchemy is a powerful relational database framework that supports several database backends. It offers a high-level ORM and low-level access to the database's native SQL functionality.

```
pip install flask-sqlalchemy
```

In Flask-SQLAlchemy, a database is specified as a URL. Table **??** lists the format of the URLs for the three most popular database engines.

Database engine	**URL**
MySQL	`mysql://username:password@hostname/database`
Postgres	`postgresql://username:password@hostname/database`
SQLite (Linux, macOS)	`sqlite:////absolute/path/to/database`

Tables 4.1: SQLAlchemy

Listing 4.2: database configuration

```
import os
from flask_sqlalchemy import SQLAlchemy

basedir = os.path.abspath(os.path.dirname(__file__))
app = Flask(__name__) app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///' + os.
    path.join(basedir, 'data.sqlite')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
```

Listing 4.3: Model definition

```
class Follow(db.Model):
    __tablename__ = 'follows'
    follower_id = db.Column(db.Integer, db.ForeignKey('users.id'), primary_key=
        True)
    followed_id = db.Column(db.Integer, db.ForeignKey('users.id'), primary_key=
        True)
    timestamp = db.Column(db.DateTime, default=datetime.utcnow)
```

You can operate the database through the model class. For example

```
user = User.query.filter_by(username=username).first()
```

4.5 Migrate

Flask-SQLAlchemy creates database tables from models only when they do not exist already, so the only way to make it update tables is by destroying the old tables first—but of course, this causes all the data in the database to be lost.

A better solution is to use a database migration framework. A database migration framework keeps track of changes to a database schema, allowing incre- mental changes to be applied.

```
pip install flask-migrate


from flask_migrate import Migrate
#...
migrate = Migrate(app, db)
```

To expose the database migration commands, Flask-Migrate adds a flask db com- mand with several sub-commands. When you work on a new project, you can add support for database migrations with the init subcommand:

```
flask db init
```

To make change to databases:

```
# make migrations
flask db migrate
# apply the migrations
flask db upgrade
```

4.6 Mail

Although the smtplib package from the Python standard library can be used to send email inside a Flask application, the Flask-Mail extension wraps smtplib and integrates it nicely with Flask.

```
pip install flask-mail
```

The extension connects to a Simple Mail Transfer Protocol (SMTP) server and passes emails to it for delivery. If no configuration is given, Flask-Mail connects to localhost at port 25 and sends email without authentication.

Key	Default	Description
MAIL_SERVER	localhost	Hostname or IP of the email server
MAIL_PORT	25	Port of the email server
MAIL_USE_TLS	False	Enable Transport Layer Security (TLS) security
MAIL_USE_SSL	False	Enable Secure Sockets Layer (SSL) security
MAIL_USERNAME	None	Mail account username
MAIL_PASSWORD	None	Mail account password

Tables 4.2: Mail configuration

```
app.config['MAIL_SERVER'] = 'smtp.googlemail.com'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USERNAME'] = os.environ.get('MAIL_USERNAME')
app.config['MAIL_PASSWORD'] = os.environ.get('MAIL_PASSWORD')
```

4.7 Login

When users log in to the application, their authenticated state has to be recorded in the user session, so that it is remembered as they navigate through different pages. Flask-Login is a small but extremely useful extension that specializes in managing this particular aspect of a user authentication system, without being tied to a specific authentication mechanism.

```
pip install flask-login
```

4.8 HTTPAuth

Because of the “statless” feature of the RESTful (the server is not allowed to “remember” anything about the client between requests), we can use HTTP authentication, user credentials are included in an Authorization header with all requests.

```
pip install flask-httpauth
```

Chapter 5

Flask Deploy

Here we used the Flask to develop the web site and used gunicorn and nginx to deploy it.

5.1 Gunicorn

5.1.1 Install Gunicorn

```
  pip install gunicorn
```

5.1.2 Creating the WSGI Entry Point

Create wsgi.py in you application. The content of the file is as follows:

```
from myblog import app

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

5.1.3 Start Gunicorn

```
  gunicorn --bind 0.0.0.0:8000 wsgi:app
```

5.1.4 Build a Gunicorn Service

Create the file /etc/systemd/system/myblog.service:

```
[Unit]
Sescription=Gunicorn instance to serve myblog
After=network.target

[Service]
User=root
Group=www-data
WorkingDirectory=/var/www/myblog
Environment="PATH=/var/www/myblog/venv/bin"
```

```
ExecStart=/var/www/myblog/venv/bin/gunicorn --workers 3 --bind 0.0.0.0:8000 wsgi:
    app

[Install]
WantedBy=multi-user.target
```

After the creation, you can start the service and enable it to start at the server starts up:

```
  systemclt start myblog
  systemctl enable myblog
```

5.2 Nginx

5.2.1 Install Nginx

```
  dnf install nginx
  # or
  apt install nginx
```

5.2.2 Config

Create the file /etc/nginx/sites-available/myblog:

```
server {
    listen 80;
    server_name mingmingli.site www.mingmingli.site;

    location / {
        include proxy_params;
        proxy_pass http://127.0.0.1:8000/;
    }
}
```

Then create a link:

```
  ln -s /etc/nginx/sites-available/myblog /etc/nginx/sites-enabled/
```

You can test the configuration is correct or not:

```
  nginx -t
```

5.2.3 Tell Flask it is Behind a Proxy

myblog.py

```
from werkzeug.middleware.proxy_fix import ProxyFix

app.wsgi_app = ProxyFix(
    app.wsgi_app, x_for=1, x_proto=1, x_host=1, x_prefix=1
)
```

5.3 SSL

https://certbot.eff.org/instructions

Bibliography

[1] *Flask Web Development*. O'Reilly Media, Inc, 2018.

[2] Aston Zhang et al. *Dive into Deep Learning*. 2021.

[3] Debra Cameron et al. *Learning GNU Emacs*. O'Reilly Media, Inc., 2004.

[4] Ian Goodfellow et al. *Deep Learning*. 2016.

[5] Yann LeCun et al. Gradientbased learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[6] D.E. Knuth. *The TEXbook*. Addison Wesley, 1986.

[7] D.E. Knuth. Typesetting concrete mathematics. *TUGboat*, 10(1):31–36, April 1989.

[8] Adrian Rosebrock. *Practical Python and OpenCV*. PyImageSearch, 2016.

[9] Adrian Rosebrock. *Deep Learning for Computer Vision with Python*. PyImageSearch, 2017.