Networking Academy

Python DTS PRoA 2022

Library Session 2: Flask

Muhammad Ogin Hasanuddin

KK Teknik Komputer
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Outline

Section 1: Setup & workflow

Creating a virtual environment

- venv (for Python 3) and virtualenv (for Python 2) allow you to manage separate package installations for different projects. They essentially allow you to create a "virtual" isolated Python installation and install packages into that virtual installation. When you switch projects, you can simply create a new virtual environment and not have to worry about breaking the packages installed in the other environments. It is always recommended to use a virtual environment while developing Python applications.
- To create a virtual environment, go to your project's directory and run venv. If you are using Python 2, replace venv with virtualenv in the below commands.

```
$ py -m venv flaskvenv
```

- The second argument is the location to create the virtual environment. Generally, you can just create this in your project and call it env.
- venv will create a virtual Python installation in the `flaskvenv` folder.

Note: You should exclude your virtual environment directory from your version control system using .gitignore or similar.

Activating a virtual environment

- Before you can start installing or using packages in your virtual environment you'll need to activate it. Activating a virtual environment will put the virtual environment-specific python and pip executables into your shell's PATH.

```
$ .\flaskvenv\Scripts\activate
```

- You can confirm you're in the virtual environment by checking the location of your Python interpreter:

```
$ where python
```

- It should be in the flaskvenv directory:

```
$ ...\flaskvenv\Scripts\python.exe
```

- As long as your virtual environment is activated pip will install packages into that specific environment and you'll be able to import and use packages in your Python application.

Leaving the virtual environment

- If you want to switch projects or otherwise leave your virtual environment, simply run:

```
$ deactivate
```

- If you want to re-enter the virtual environment just follow the same instructions before, for activating a virtual environment. There's no need to re-create the virtual environment.

Installing packages

- Now that you're in your virtual environment you can install packages. Let's install the Requests library from the Python Package Index (PyPI):

```
$ py -m pip install requests
```

Freezing dependencies

- Pip can export a list of all installed packages and their versions using the freeze command:

```
$ py -m pip freeze
```

- Which will output a list of package specifiers such as:

```
certifi==2021.10.8
charset-normalizer==2.0.12
idna==3.3
requests==2.27.1
urllib3==1.26.9
```

- This is useful for creating Requirements Files that can re-create the exact versions of all packages installed in an environment.

Using requirements files

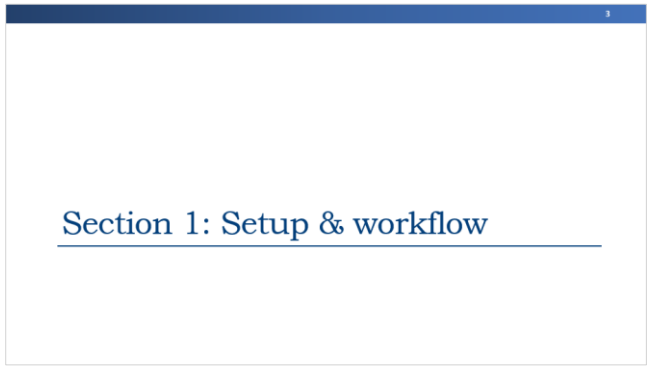
- Instead of installing packages individually, pip allows you to declare all dependencies in a Requirements File. For example you could create a requirements.txt file containing:

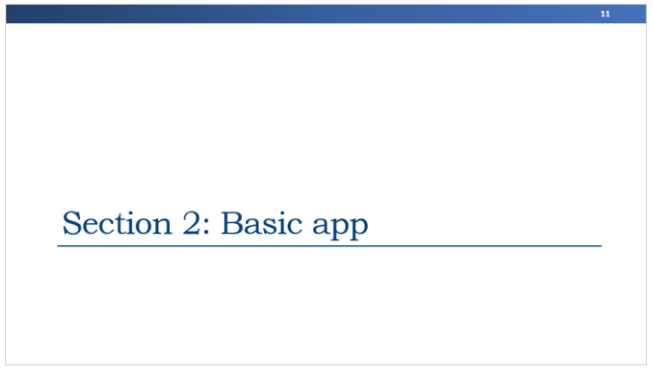
```
certifi==2021.10.8
charset-normalizer==2.0.12
idna==3.3
requests==2.27.1
urllib3==1.26.9
```

- And tell pip to install all of the packages in this file using the -r flag:

```
$ py -m pip install -r requirements.txt
```

Outline


3
Section 1: Setup & workflow


11
Section 2: Basic app


16
Section 3: Templates and static content


21
Section 4: SQLite database

Section 2: Basic app

A Minimal Application

- A minimal Flask application looks something like this:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

How it works?

- First we imported the Flask class. An instance of this class will be our WSGI application.
- Next we create an instance of this class. The first argument is the name of the application's module or package. __name__ is a convenient shortcut for this that is appropriate for most cases. This is needed so that Flask knows where to look for resources such as templates and static files.
- We then use the route() decorator to tell Flask what URL should trigger our function.
- The function returns the message we want to display in the user's browser. The default content type is HTML, so HTML in the string will be rendered by the browser.

Run application

- To run the application, use the flask command or python -m flask. Before you can do that you need to tell your terminal the application to work with by exporting the FLASK_APP environment variable:

```
> set FLASK_APP=hello
> flask run
```

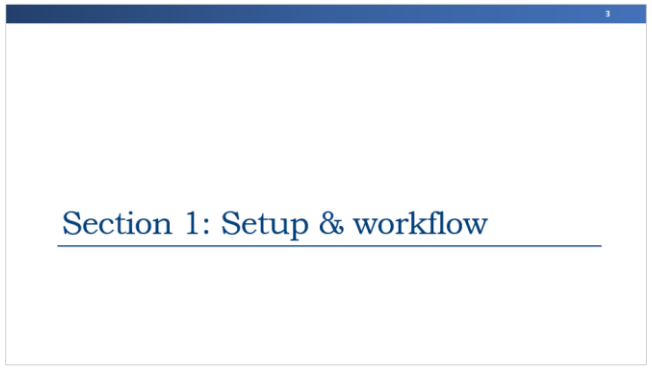
- Another way to run the application by adding:

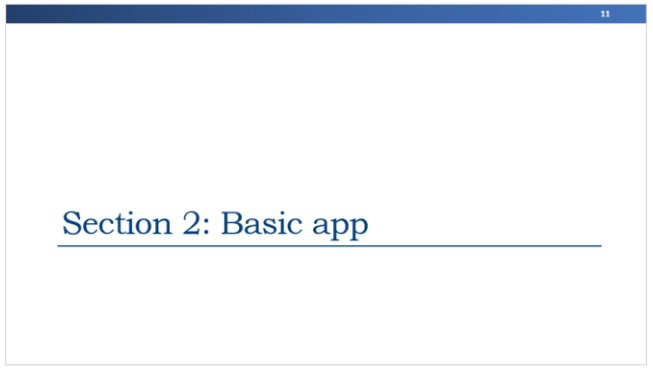
```
if __name__ == "__main__":
    app.run(debug=True)
```

- And call:

```
> python app_name
```

Outline


3
Section 1: Setup & workflow


11
Section 2: Basic app


16
Section 3: Templates and static content


21
Section 4: SQLite database

Section 3: Templates and static content

Create folder inside project folder

- Create folder for static content
  - `> mkdir static`
- Create folder for templates
  - `> mkdir templates`

Rendering Templates

- Generating HTML from within Python is not fun, and actually pretty cumbersome because you have to do the HTML escaping on your own to keep the application secure. Because of that Flask configures the Jinja2 template engine for you automatically.
- To render a template you can use the render_template() method. All you have to do is provide the name of the template and the variables you want to pass to the template engine as keyword arguments. Here's a simple example of how to render a template:

```
from flask import render_template

@app.route('/hello/')
def hello():
    return render_template('hello.html')
```

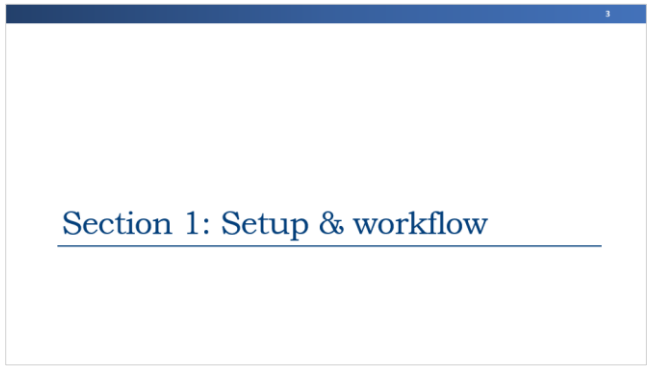
Static Files

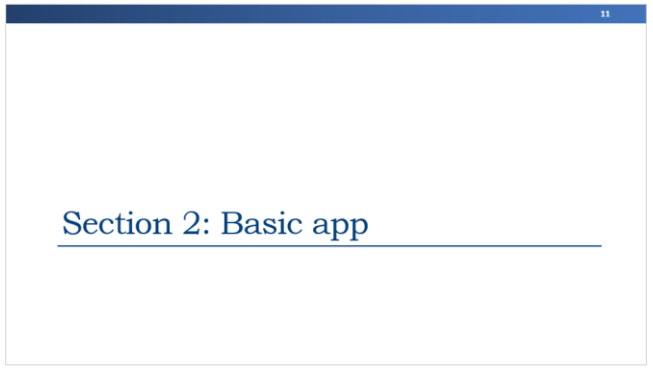
- Dynamic web applications also need static files. That's usually where the CSS and JavaScript files are coming from. Ideally your web server is configured to serve them for you, but during development Flask can do that as well. Just create a folder called static in your package or next to your module and it will be available at /static on the application.
- To generate URLs for static files, use the special 'static' endpoint name:

```
url_for('static', filename='style.css')
```

- The file has to be stored on the filesystem as static/style.css.

Outline


3
Section 1: Setup & workflow


11
Section 2: Basic app


16
Section 3: Templates and static content


21
Section 4: SQLite database

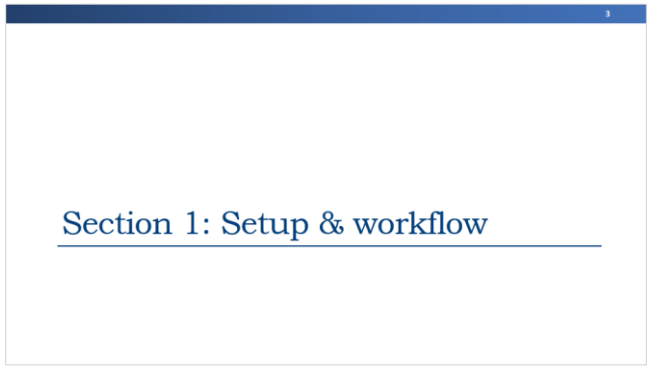
Section 4: SQLite database

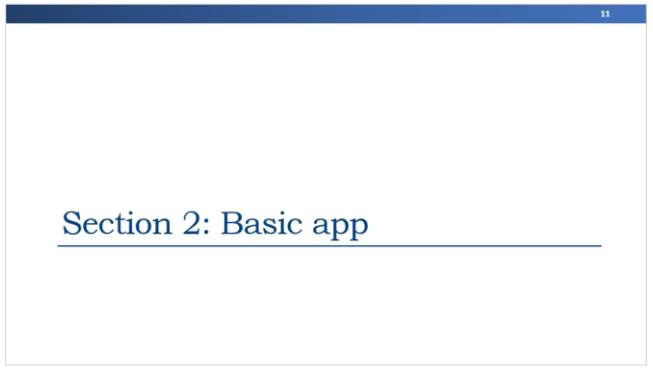
Minimal app

- For the common case of having one Flask application all you have to do is to create your Flask application, load the configuration of choice and then create the SQLAlchemy object by passing it the application.

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'sqlite:////tmp/test.db'
db = SQLAlchemy(app)


class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True,
nullable=False)
    email = db.Column(db.String(120), unique=True,
nullable=False)

    def __repr__(self):
        return '<User %r>' % self.username
```

Outline


3
Section 1: Setup & workflow


11
Section 2: Basic app


16
Section 3: Templates and static content


21
Section 4: SQLite database

Terimakasih!