Python Flask Cheat Sheet

by amicheletti via cheatography.com/39488/cs/12263/

Routing

`route()` decorator is used to bind a `function` to a `URL`

Example:

```
@app.route('/')
```

By default a route only answers to GET requests, but you can provide the `methods` argument.

```
@app.route('/login', methods=['GET', 'POST'])
```

flask-restful

With Flask-Restful you can create RESTful API with your Flask app

Create an Flask App

```
app = Flask(__name__)
```

Then create the Api object passing the App object

```
api = Api(app)
```

Then you can create `Resource`s and add them to the API

```
class NewsFinder(Resource): pass
```

```
api.add_resouce(NewsFinder, '/', '/news')
```

You can implement each HTTP verb with functions named like the verb, but in lowercase.

Example:

```
def get(self): pass
```

```
def put(self, id): pass
```

To parse arguments passed by the url use

```
parser = reqparse.RequestParser()
```

You can pass `parse_args(strict=True)` to throw an error if arguments that were not defined by you has been passed

Add the arguments with `parser.add_arguments('limit',`

```
type=int, help='Help Text', required=True)
```

You can specify the location to look for this argument with

```
add_argument('User-Agent', location='headers')
```

Example locations: form, args, headers, session, cookies, files

Then inside the function you can `args = parser.parse_args()` to get the parsed args. This variable `args` will become a dictionary with the values, ccess via `args['limit']`

Imports

```
from flask_restful import Api, Resource, reqparse
```

Flask_jwt

```
from flask import Flask
from flask_restful import Api, Resource
from flask_jwt import JWT, jwt_required,
current_identity
app = Flask(__name__)
app.config['SECRET_KEY'] = 'my-secret'
api = Api(app, prefix='/api/v1')
USER_DATA = {
"amicheletti": "coracaopeludo"
}
class User(object):
def __init__(self, id):
self.id = id
def __str__(self):
return "User (id={})".format(self.id)
def verify(username, password):
if not (username and password):
return False
if (USER_DATA.get(username) == password):
return User(id=123)
def identity(payload):
user_id = payload['identity']
return { "uid": user_id }
jwt = JWT(app, verify, identity)
class UltimateQuestion(Resource):
@jwt_required()
def get(self):
return { "meaningoflife" : 42, "who_asked" :
dict(current_identity) }
api.add_resource(UltimateQuestion, '/', '/life')
if __name__ == "__main__":
app.run(debug=True)
```

You must have an `authentication_handler()` which takes 2 arguments and a `identity_handler()` which takes 1 argument

Authentication handler must return an Object that has an id attribute

Identity handler return what is going to be send to `'identity'` key of the JSON

To get the token, curl POST to the `/auth` like this:

```
curl -H "Content-type: application/json" -X POST -d
'{"username":"amicheletti","password":"coracaopeludo"}
' http://127.0.0.1:5000/auth`
```

URL Building

When routing some function to a URL, you can use function `url_for()` to generate the URL to that function.
Example, if you have something like `@app.route('/user/<username>') def profile(username): pass` you use `url_for('profile', username="Andre")` to get the URL for that route.
That way you can avoid having to change the hardcoded URL everywhere in the code.

File Uploads

To handle file uploads with Flask, the HTML form must be set with `enctype="multipart/form-data"`
Then you can use it from a dictionary in`requests.files`
Example:

```
f = request.files['the_file']
f.save('/var/www/uploads/uploaded_file.txt')
```

Redirects and Errors

`redirect('url')` Pass a URL to this function to redirect a user
`abort(401)` This will abort the request early with an error code
To customize the error page use `@app.errorhandler(404)`, but don't forget to pass the error code. Example:

```
return render_template('page_not_found.html'), 404
```

virtualenv

`virtualenv my_project`	Create environment named my_project
`-p /usr/bin/python3.5`	Pass this argument to define Python to be used
`source my_project/bin/activate`	Start using the environment
`deactivate`	To leave your environment
`pip freeze > requirements.txt`	Freeze your requirements to a file
`pip install -r requirements.txt`	Install using the requirements file

Blueprint

Blueprints are objects similar to the Flask application object, but are not an actual application. They can record operations and endpoints routing and deliver resources, and then they are registered to the application (can be registered multiple times) under a specific URL.
Create a blueprint:

```
feed_blueprint = Blueprint('feed', __name__)
```

Use blueprint like an Flask app object:

```
@feed_blueprint.route('\')
```

Register the blueprint to the real application

```
app.register_blueprint(feed_blueprint,
url_prefix='/feed')
```

Blueprint root folder

```
feed_blueprint.root_path
```

To build url for Blueprints, put the name used in the object creation before the function name:

```
url_for('feed.index')
```

Also you can use the error handler just like the Flask object

```
@feed_blueprint.errorhandler(404)
```

JWT

JWT stands for JSON Web Token, that are used to securely transmit JSON information between two parties or authenticate
They consist in three parts: Header, Payload and Signature. These three parts are JSON object which are then `Base64URL` encoded and included to
the token `header.payload.signature`

- Header

In Header, you generally have two information:
the type of the token and the algorithm used

```
{
"alg" : "HS256",
"typ" : "JWT"
}
```

- Payload

In Payload you have "claims" about an Entity (the user for example) and other metadata.
Example:

```
{
"id": "1234567890",
"name": "John Doe",
```

JWT (cont)

```
"admin": true
}
```

There are Reserved Claims (predefined), Public Claims (defined by users at IANA JSON Web Token Registry) and Private Claims (custom claims agreed by both parties)

- Signature

To generate the signature, take the encoded header and payload, a secret and encode all that with the algorithm used.

Example: `HMACSHA256( base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)`

- Usage

Now when the user wants to access a protected route or resource, the user agent must send the JWT typically in the *Authorization header*, using the *Bearer* schema, like this:

`Authorization: Bearer <token>`

Variable Rules

`<username>`	default for `<string:>`
`<string:>`	accepts any text without slash
`<int:>`	accepts integers
`<float:>`	floating point values
`<path:>`	like `<string:>` but accept slashes
`<any:>`	matches one of the items provided
`<uuid:>`	accepts UUID strings

Add variable parts to a URL. You can also specify a converter to the variable.

Request Object

The request object is available when routing passing `method` argument.

`request.method` is the HTTP method (POST, GET...)

`request.fòrm` Use this to access the form data passed

`request.args.get('key', '')` Use this to access parameters passed by url `?key=value`

```
from flask import request
```

Logging

```
app.logger.debug('A value for debugging')
app.logger.warning('A warning occurred (%d apples)',
42)
app.logger.error('An error occurred')
```