CODING NOMADS

18) Going RESTful: Build an API Lesson

Error Handling in Your Flask API

4 min to complete · By Brandon Gigous

Contents

Note: This lesson builds on a project that's been gradually worked on throughout this Python + Flask course.

Sometimes your app runs into problems, and your API is no different. Let's go over what you need to do for error handling in your Python + Flask API.

Error Handling with JSON

Your API will need to handle your clients' requests with JSON, a common way to package, send, and receive data in webapps. But currently, your error handler only responds to errors in HTML. To fix it, it needs to return JSON only when a request is made to the API, otherwise your app will return errors via HTML like it has until now. Flask's `jsonify()` function will help you do that. Jump into `error.py` of your `main` blueprint to fix the error handler:

```
from flask import jsonify

@main.app_errorhandler(404)
```

```
    def page_not_found(e):
        if request.accept_mimetypes.accept_json and \
                not request.accept_mimetypes.accept_html:
            response = jsonify({'error': 'not found'})
            response.status_code = 404
            return response
        error_msg="That page doesn't exist."
        return render_template('error.html', error_msg=error_msg), 404
```

"Whoa-whoa-whoa, what's all this "mime" stuff?" you might be asking. There's no need for makeup or imaginary props. The `accept_mimetypes` attribute of the `request` object is a shortcut to the contents of the `Accept` header in the request. This header tells the server what type of response the client expects. For your webapp, you only need to worry about JSON or HTML responses, and since your API is only going to deal with JSON, the code checks if `accept_json` is true. If true, a JSON response is constructed. The `jsonify()` function takes a dictionary and gives back a response as JSON.

Learn by Doing 

Task: Implement the same functionality for error handling in your 500 error handler.

Web Service Error Handling

Some errors through the API can be handled in your API code, including the 403 forbidden error. Head over to `api/error.py` to make a helper function:

```
def forbidden(message):
    response = jsonify({'error': 'forbidden', 'message': message})
    response.status_code = 403
    return response
```

That's right, "helper function." The API doesn't need the `@api.error_handler` decorator because your API can do it by itself when it encounters an error. It's only JSON, after all.

Learn by Doing 

Task: Create similar helper functions for errors:

- 400: bad request
- 401: unauthorized

Hint: they aren't very different from the one above. ;)

Hey, you did it! You've now completed the prepwork for your API. Now its time to get into the juicier details... onward!

Summary: Restful Web Services and Error Handling

- Your API will need to handle your clients' requests with JSON, a common way to package, send, and receive data in webapps. But currently, your error handler only responds to errors in HTML.
- To fix it, it needs to return JSON only when a request is made to the API, otherwise your app will return errors via HTML like it has until now.
- Flask's `jsonify()` function will help you do that.