

comp0034_flask_rest

COMP0034 Code as at the start of the REST API lecture

To see a completed version of the REST example created in the lecture refer to https://github.com/UCLComputerScience/comp0034_flask_rest_complete.git

Setup

1. Create a venv
2. Install the packages from requirements.txt

Exercise 1: Create a REST API for courses

1. Add a new Python package called `api` to `cscourses`
2. Create a new `routes.py` in the `api` directory and add a blueprint for the api

```
from flask import Blueprint

bp_api = Blueprint('api', __name__, url_prefix='/api')
```

3. Register the blueprint in `cscourses/__init__.py`

```
from cscourses.api.routes import bp_api
app.register_blueprint(bp_api)
```

4. The `Course` class in `models.py` has already been updated to add a property serialise the data.
5. Add routes for the following to `api/routes.py` :

HTTP Method	URI	Action
GET	http://[hostname]/api/courses	Retrieve the list of courses
GET	http://[hostname]/api/courses/[course_id]	Retrieve a course
POST	http://[hostname]/api/courses	Create a new course
PUT	http://[hostname]/api/courses/[course_id]	Update an existing course

```
from flask import Blueprint, jsonify, request, make_response
from flask_httpauth import HTTPBasicAuth
```

```
from cscourses import db
from cscourses.models import Course, User
```

```
bp_api = Blueprint('api', __name__, url_prefix='/api')
```

```
@bp_api.after_request
def add_header(response):
    response.headers['Content-Type'] = 'application/json'
    return response
```

```
@bp_api.errorhandler(404)
def not_found(error):
    error = {
        'status': 404,
        'message': 'Not Found: ' + request.url,
    }
    response = jsonify(error)
    return make_response(response, 404)
```

```

@bp_api.errorhandler(401)
def not_authorized():
    error = {
        'status': 401,
        'message': 'You must provide username and password to access this resource',
    }
    response = jsonify(error)
    return make_response(response, 404)

```

```

@http_auth.verify_password
def verify_password(username, password):
    user = User.query.filter_by(name=username).first()
    if not user or not user.check_password(password):
        return False
    return True

```

```

@bp_api.route('/courses', methods=['GET'])
@http_auth.login_required
def read_courses():
    courses = Course.query.all()
    json = jsonify(courses=[c.serialize for c in courses])
    return make_response(json, 200)

```

```

@bp_api.route('/courses/<int:course_id>', methods=['GET'])
@http_auth.login_required
def read_course(course_id):
    course = Course.query.filter_by(id=course_id).first_or_404()
    json = jsonify(course=course.serialize)
    return make_response(json, 200)

```

```

@bp_api.route('/courses', methods=['POST'])
@http_auth.login_required
def create_course():
    '''To create a new course all fields must be provided'''
    # request.args.get() will return None if the arg is not present in the request
    course_code = request.args.get('course_code', type=str)
    name = request.args.get('name', type=str)
    teacher_id = request.args.get('teacher_id', type=int)
    # The following line checks if any of the variables are None
    if None in (course_code, name, teacher_id):
        headers = {"Content-Type": "application/json"}
        json = jsonify({'message': 'Please provide: course_code, name, teacher_id'})
        return make_response(json, 400, headers)
    course = Course(course_code=course_code, name=name, teacher_id=teacher_id)
    db.session.add(course)
    db.session.commit()
    json = jsonify(Course=course.serialize)
    return make_response(json, 201)

```

```

@bp_api.route('/courses/<int:course_id>', methods=['PUT'])
@http_auth.login_required
def update_course(course_id):
    # Find the course by its ID
    course = db.session.query(Course).filter_by(id=course_id).first_or_404()
    # Create variables using the values posted in the request
    course_name = request.args.get('course_name')
    teacher_id = request.args.get('teacher_id')
    course_code = request.args.get('course_code')
    # If any of the variables have a provided value then update the database
    if course_name is not None:
        course.name = course_name
    if teacher_id is not None:
        course.teacher_id = teacher_id
    if course_code is not None:
        course.course_code = course_code
    db.session.commit()
    json = jsonify({'message': 'Updated Course with id {}'.format(course.id)})
    return make_response(json, 200)

```

```

@bp_api.route('/courses/<int:course_id>', methods=['DELETE'])
@http_auth.login_required
def delete_course(course_id):
    course = Course.query.filter_by(id=course_id).one()
    db.session.delete(course)
    db.session.commit()
    json = jsonify({'message': 'Removed Course with id {}'.format(course_id)})
    return make_response(json, 200)

```

6. Test the API routes using Postman

- i. Download and install postman from <https://www.getpostman.com/downloads/>
- ii. Test each of the following using the appropriate HTTP methods:
 - GET: <http://127.0.0.1:5000/api/courses>
 - GET: <http://localhost:5000/api/courses/1>
 - GET: <http://localhost:5000/api/courses/99> (returns 404)
 - POST: http://localhost:5000/api/courses?course_code=COMP989898&name=My new course&teacher_id=1 (/) /should return
 - POST: http://localhost:5000/api/courses?course_code=COMP989898 (should return error message)
 - PUT: http://localhost:5000/api/courses/14?teacher_id=2
 - DELETE: <http://localhost:5000/api/courses/14>

7. Add authentication

- i. Add a new route to create a user account

```

@bp_api.route('/users', methods=['POST'])
def create_user():
    username = request.args.get('username')
    password = request.args.get('password')
    headers = {"Content-Type": "application/json"}
    if username is None or password is None:
        json = jsonify({'message': 'Missing username or password'})
        return make_response(json, 400, headers)
    if User.query.filter_by(name=username).first() is not None:
        json = jsonify({'message': 'Duplicate username'})
        return make_response(json, 400, headers)
    user = User(name=username)
    user.set_password(password)
    db.session.add(user)
    db.session.commit()
    json = jsonify({'user_id': '{}'.format(user.id), 'name': '{}'.format(user.name) })
    return make_response(json, 201, headers)

```

- ii. Add Flask-HTTPAuth authentication to require users to provide username and password to access the API e.g.

```

from flask_httpauth import HTTPBasicAuth

bp_api = Blueprint('api', __name__, url_prefix='/api')

http_auth = HTTPBasicAuth()

```

- iii. Provide an implementation of verify_password to check the hashed password

```

@http_auth.verify_password
def verify_password(username, password):
    user = User.query.filter_by(name=username).first()
    if not user or not user.check_password(password):
        return False
    return True

```

- iv. Update the existing routes to require authentication to access them using `login_required` :

```
@bp_api.route('/courses', methods=['GET'])
@http_auth.login_required
def read_courses():
```

v. Test using Postman.

- Send a POST request to create a new user to <http://localhost:5000/api/users> and include a parameter for username and a parameter for password
- Send a GET request to <http://localhost:5000/api/courses>, on the Authorizarion tab select Type: Basic Auth and enter the username and password you created

Exercise 2: Create a page that uses the courses REST API

1. Add a new template `courses_api_client.html`. Note that the course data returned from the API results in a nested dict e.g.:

```
{'courses': [{'course_code': 'COMP0015', 'course_id': 1, 'name': 'Introduction to Programming', 'teacher_id': 6},
               {'course_code': 'COMP0034', 'course_id': 2, 'name': 'Software Engineering', 'teacher_id': 5}]
}
```

The for loop in the Jinja2 template needs to use the relevant syntax, e.g.

```
{% extends 'base.html' %}
{% block title %}Courses (from API){% endblock %}
{% block content %}
    {% if courses | length %}
        <table class="table">
            <thead class="thead-dark">
                <tr>
                    <th scope="col">Course code</th>
                    <th scope="col">Title</th>
                </tr>
            </thead>
            <tbody>
                {# Note: the following syntax is for a nested dictionary #}
                {% for course in courses['courses'] %}
                    <tr>
                        <td>{{ course.course_code }}</td>
                        <td>{{ course.name }}</td>
                    </tr>
                {% endfor %}
            </tbody>
        </table>
    {% else %}
        <p>No courses found</p>
    {% endif %}
{% endblock %}
```

2. Add a route to `main/routes.py` e.g. `/api_courses` that makes a GET request to the URL for the REST API `api/courses` route. You will need to import the requests package (already installed from `requirements.txt`).

```
import requests
```

```
@bp_main.route('/api_courses', methods=['GET'])
def api_courses():
    response = requests.get('http://localhost:5000/api/courses', auth=('sarah_api', 'sanders'))
    # response.json returns a dict, in this case it is a nested dictionary
    courses = response.json()
    return render_template('courses_api_client.html', courses=courses)
```

3. Test it!

- Stop/restart your Flask app

- Create a user account with the username and password using `api/users` e.g. <http://localhost:5000/api/users?username=sarah&password=sanders>
- Go to http://localhost:5000/api_courses in a browser. You should get a page displaying a table of courses.

Exercise 3: Create a page that uses data from the Hacker News API

The page will display the titles of the top 50 stories from the Hacker News API. Top stories URI is: <https://hacker-news.firebaseio.com/v0/topstories.json>. No authentication required.

1. Add a `news.html` template (already created).
2. Add a new route for news to `main/routes.py`

```
@bp_main.route('/news')
def news():
    # Make an API call, and store the response.
    url = 'https://hacker-news.firebaseio.com/v0/topstories.json'
    response = requests.get(url)
    # Process information about the first 10 news item in the top stories list.
    item_ids = response.json()
    stories = []
    for item_id in item_ids[:10]:
        # Make a separate API call for each item.
        url = ('https://hacker-news.firebaseio.com/v0/item/' + str(item_id) + '.json')
        response = requests.get(url)
        story_data = response.json()
        story = {'title': story_data['title'], 'url': story_data['url'], }
        stories.append(story)
    return render_template('news.html', stories=stories)
```

3. Test the page works (note it will take longer to load):