

Creating the REST API routes in Flask

COMP0034 2023-24 Week 3 coding activities.

1. Preparation

This assumes you have already forked the coursework repository and cloned the resulting repository to your IDE.

- 1. Create and activate a virtual environment
- 2. Install the requirements `pip install -r requirements.txt`
- 3. Run the app `flask --app paralympics run --debug`
- 4. Open a browser and go to `http://127.0.0.1:5000`
- 5. Stop the app using `CTRL+C`
- 6. Check that you have an instance folder containing `paralympics.sqlite`

2. Introduction

Assume that the following routes were designed for the API.

HTTP method	URL	Body	Response
GET	regions	None	Returns a list of NOC region codes with region name and notes
GET	regions/<code>	None	Returns the region name and notes for a given code
PATCH	regions/<code>	Changed fields for the NOC record	Return all the details of the updated NOC record
POST	regions	Region code, region name and (optional) notes	Status code 201 if new NOC code was saved.
DELETE	regions/<code>	None	Removes an NOC code and if successful returns 202 (Accepted)
GET	events	None	Returns a list of events with all details

HTTP method	URL	Body	Response
GET	events/<event_id>	None	Returns all the details for a given event
POST	events	Event details	Status code 201 if new event was saved.
PATCH	events/<event_id>	Event details to be updated (specific fields to be passed)	Return all the details of the updated event
DELETE	events/<event_id>	None	Removes an event and if successful returns 202 (Accepted)

You will need to refer to the Flask documentation:

- [routing](#)
- [HTTP methods](#)

2. Serialize and deserialize the data

Serialization refers to the process of converting a Python object into a format that can be used to store or transmit the data and then recreate the object when needed using the reverse process of deserialization. Python objects need to be converted into a flat structure that contains only native Python datatypes.

There are different formats for the serialization of data, such as JSON, XML, and Python's pickle. JSON returns a human-readable string form, while Python's pickle library can return a byte array.

In the COMP0034 teaching materials pickle is used for serializing and deserializing machine learning models, and JSON for the REST API.

You could serialize your own classes, for example by adding a `to_json` method to a class, or by creating an instance of the class and `dump` ing. The limitation of this is it cannot be handle relationships between classes. The code would look something like this (not tested but should work!):

```
import json
from sqlalchemy.orm import Mapped, mapped_column
from paralympics import db

class User(db.Model):
    id: Mapped[int] = mapped_column(db.Integer, primary_key=True)
    email: Mapped[str] = mapped_column(db.String, unique=True, nullable=False)
```

```
password: Mapped[str] = mapped_column(db.String, unique=True, nullable=False)

def to_json(self):
    return json.dumps(self.__dict__)

# To use, create a User object by querying the database for the user with the id 1, serialise
user = db.session.execute(db.select(User).order_by(User.username)).scalars()
user_json = user.to_json()
print(user_json)
```

For complex types such as database models, use a serialization library to convert the data to valid JSON types. There are many serialization libraries, this activity uses:

- [Flask-Marshmallow](#)
- [marshmallow-sqlalchemy](#)

These packages are included in this week's requirements.txt. If you did not install the requirements.txt then `pip install flask-marshmallow marshmallow-sqlalchemy` will install them.

The steps for this part are:

- 2.1 Configure the app for Flask-Marshmallow
- 2.2 Create Marshmallow-SQLAlchemy schemas. The schema can then be used to `dump` and `load` ORM objects (i.e. objects from SQLAlchemy).

`dump` refers to creating JSON from an object. `load` refers to creating an object from JSON.

2.1 Configure the app for Flask-Marshmallow

Flask-Marshmallow will be used with Flask-SQLAlchemy. The [documentation](#) states that Flask-SQLAlchemy must be initialized before Flask-Marshmallow.

Add the code to create and initialise Marshmallow to the `__init__.py` file. Note that for brevity other code from the `__init__.py` file is not shown here:

```
from flask_marshmallow import Marshmallow

# Create a global SQLAlchemy object
db = SQLAlchemy()
# Create a global Flask-Marshmallow object
ma = Marshmallow()

def create_app():
    # Initialise Flask-SQLAlchemy
    db.init_app(app)
```

```
# Initialise Flask-Marshmallow
ma.init_app(app)
```

2.2 Create Marshmallow-SQLAlchemy schemas

Now define [Marshmallow SQLAlchemy schemas as per their documentation](#). These allow Marshmallow to essentially 'translate' the fields for a SQLAlchemy object and provide methods that allow you to convert the objects to JSON.

For this paralympics example the code is given for you below. There are two methods for creating schemas in this code.

The first example (for Region) you would use if you wish to only provide some, not all, the fields from a class in the data. This inherits `ma.SQLAlchemySchema` and you then need to state the fields that you wish to be included in the data.

The second example (for Event) provides all the fields. This inherits `ma.SQLAlchemyAutoSchema` and this automatically includes all the fields from your models class, so you do not have to repeat them.

`model = Region` states the name of the model class. You need to include this.

`sqla_session = db.session` tells Marshmallow the session to use to work with the database. You need to include this.

`load_instance = True` is optional and will deserialize to model instances.

`include_fk = True` is only needed if you want the foreign key field to be included in the data.

`include_relationships = True` is only needed if you have relationships between tables.

Create a file called `schemas.py` :

```
from paralympics.models import Event, Region
from paralympics import db, ma

# Flask-Marshmallow Schemas
# See https://marshmallow-sqlalchemy.readthedocs.io/en/latest/#generate-marshmallow-schemas

class RegionSchema(ma.SQLAlchemySchema):
    """Marshmallow schema defining the attributes for creating a new region."""

    class Meta:
        model = Region
        load_instance = True
        sqla_session = db.session
        include_relationships = True

    NOC = ma.auto_field()
```

```

region = ma.auto_field()
notes = ma.auto_field()

class EventSchema(ma.SQLAlchemyAutoSchema):
    """Marshmallow schema for the attributes of an event class. Inherits all the attributes fr

class Meta:
    model = Event
    include_fk = True
    load_instance = True
    sqla_session = db.session
    include_relationships = True


```

3. Create a REST API route

The steps for this are:

1. Import the Marshmallow SQLAlchemy schemas and create instances of them in the python file where you define the routes (`paralympics/paralympics.py`).
2. Define the Flask route.
3. Implement the function for the route to query the database, `dump` the result to JSON and return the JSON to the browser.

3.1 Import the Marshmallow SQLAlchemy schemas

There are two variants of each schema shown below, one provides a single result (e.g. one event), the other provides for multiple results (e.g. all events). 

Open the Python file with the routes, `paralympics\paralympics.py` and add:

```

from paralympics.schemas import RegionSchema, EventSchema

# Flask-Marshmallow Schemas
regions_schema = RegionSchema(many=True)
region_schema = RegionSchema()
events_schema = EventSchema(many=True)
event_schema = EventSchema()

```

3.2 Define a route

The first route is `/region` which gets a list of all the regions and returns them in JSON. This uses the [HTTP GET method](#).

To define a route with an HTTP method in Flask you can use either of the following:

```
# Use route and specify the HTTP method(s). If you do not specify the methods then it will def
@app.route('/something', methods=['GET', 'POST'])
def something():
    pass

# Use Flask shortcut methods for each HTTP method `.get`, `.post`, `.delete`, `.patch`, `.put`
@app.get('/something')
def something():
    pass
```

Add the following route:

```
@app.get("/regions")
def get_regions():
    """Returns a list of NOC regions and their details in JSON."""
```

3.3 Implement the route function

The route function, `get_regions()` should get a list of all the regions and returns these in an HTTP response in JSON format.

The Flask-SQLAlchemy query syntax for a 'SELECT' query is explained in the [Flask-SQLAlchemy 3.1 documentation](#)

Query the database to get the results, then use the schemas to convert the SQLAlchemy result objects to a JSON syntax.

```
@app.get("/regions")
def get_regions():
    """Returns a list of NOC region codes and their details in JSON."""
    # Select all the regions using Flask-SQLAlchemy
    all_regions = db.session.execute(db.select(Region)).scalars()
    # Get the data using Marshmallow schema (returns JSON)
    result = regions_schema.dump(all_regions)
    # Return the data
    return result
```

The code above will return the JSON data that is the result of the `schema.dump()`. You could also use the [flask.make_response\(\) function](#) to control what is returned.

Run the app and check that the route returns JSON.

- Run the app `flask --app paralympics run --debug`

- Go to <http://127.0.0.1:5000/region>
- Stop the app using CTRL+C

Now try and implement the `@app.get('/events')` route yourself.

4. Add a route to return one event

To return a single event you need to specify the event's id in the URL. This is done using a variable in the URL.

Variable routes in Flask can be defined as follows:

```
@app.get("/events/<event_id>")
def event_id(event_id):
    """ Returns the event with the given id JSON.

    :param event_id: The id of the event to return
    :param type event_id: int
    :returns: JSON
    """
    event = db.session.execute(
        db.select(Event).filter_by(event_id=event_id)
    ).scalar_one_or_none()
    return events_schema.dump(event)
```

Run the app and check that the route returns JSON for just one region.

- Run the app `flask --app paralympics run --debug`
- Go to <http://127.0.0.1:5000/region/2>

Now try and implement the `@app.get('/regions/<NOC>')` route yourself.

5. Add a route to create a new event

To create a new entry in a database, you submit an HTTP POST request in which you pass values for the new entry with the request. This is usually sent in the body of the request, though could be passed as parameters in the URL.

To access the body of the request, you can import Flask `request`

```
from Flask import request
```

```
@app.post('/events')
def add_event():
```

```
""" Adds a new event.
```

```
Gets the JSON data from the request body and uses this to deserialise JSON to an object us
event_schema.load()
```

```
:returns: JSON"""
```

```
ev_json = request.get_json()
event = event_schema.load(ev_json)
db.session.add(event)
db.session.commit()
return {"message": f"Event added with id= {event.id}"}
```

Now try and implement the `@app.post('/regions')` route yourself.

7. Add a route to delete an event

This uses the HTTP DELETE method and will require the id of the event to be deleted.

The code looks like this:

```
@app.delete('/events/<int:event_id>')
def delete_event(event_id):
    """ Deletes an event.

    Gets the event from the database and deletes it.

    :returns: JSON"""
    event = db.session.execute(
        db.select(Event).filter_by(event_id=event_id)
    ).scalar_one_or_none()
    db.session.delete(event)
    db.session.commit()
    return {"message": f"Event deleted with id= {event_id}"}
```

Now try and implement the `@app.delete('/regions/<noc_code>')` route yourself.

9. Add a route to update an event

To update there are two possible HTTP methods that could be used, PATCH or PUT.

PATCH allows partial updates, allowing the option to update a selection of the fields.

PUT - will replace the resource so all fields must be provided.

The following uses PATCH. As with the POST route, the data to be updated is passed in the body of the request.


```

@app.patch("/events/<event_id>")
def event_update(event_id):
    """Updates changed fields for the event.

    """
    # Find the event in the database
    existing_event = db.session.execute(
        db.select(Event).filter_by(event_id=event_id)
    ).scalar_one_or_none()
    # Get the updated details from the json sent in the HTTP patch request
    event_json = request.get_json()
    # Use Marshmallow to update the existing records with the changes from the json
    event_update = event_schema.load(event_json, instance=existing_event, partial=True)
    # Commit the changes to the database
    db.session.add(event_update)
    db.session.commit()
    # Return json showing the updated record
    updated_event = db.session.execute(
        db.select(Event).filter_by(event_id=event_id)
    ).scalar_one_or_none()
    result = event_schema.jsonify(updated_event)
    response = make_response(result, 200)
    response.headers["Content-Type"] = "application/json"
    return response

```

Checking the routes with Postman

You will only be able to submit a GET request using the browser.

To change the HTTP method to POST, UPDATE and DELETE, and to pass JSON in the body of the HTTP request, you either need Python code or a tool that supports this.

For VS Code only, there is at least one extension, [Thunder Client](#), that can test REST APIs.

[HTTPie](#) has a version that works from a terminal. You can install using `pip install httpie`.

A popular, and free, tool is Postman. The app has a lot of features and may you to sign up, though it is free.

- [Postman documentation](#)
- [Postman download](#)
- [Postman online \(requires signup\)](#)

Use postman or other to try the routes for the paralympic app.

The following include HTTPie commands that you can use in the IDE's Terminal. Make sure you install HTTPie it first!.

- GET for single region to URL <http://127.0.0.1:5000/regions/GBR>. HTTPie command: `http 127.0.0.1:5000/regions/GBR`
- GET for all regions to <http://127.0.0.1:5000/regions>. HTTPie command: `http 127.0.0.1:5000/regions`
- POST a new region to <http://127.0.0.1:5000/regions> For HTTPie: `http POST 127.0.0.1:5000/regions NOC=ZZZ region=ZedZedZed`

- For Postman, in the body select 'raw' and 'JSON' and enter the JSON below.

```
{
  "NOC": "ZZZ",
  "region": "ZedZedZed"
}
```

- PATCH to update the 'ZZZ' region adding notes. Use URL <http://127.0.0.1:5000/regions/ZZZ>. For HTTPie: `http PATCH 127.0.0.1:5000/regions/ZZZ notes="A new note."`

- For Postman, In the body select 'raw' and 'JSON' and enter

```
{
  "notes": "A new note."
}
```

- DELETE the "ZZZ" region using URL <http://127.0.0.1:5000/regions/ZZZ>. For HTTPie: `http DELETE 127.0.0.1:5000/regions/ZZZ`

Using the syntax covered above, try the following yourself:

- GET a single event: <http://127.0.0.1:5000/events/1>. For HTTPie: `http 127.0.0.1:5000/events/1`
- GET all events: <http://127.0.0.1:5000/events>
- POST a new event: <http://127.0.0.1:5000/events>
 - refer to <https://httpie.io/docs/cli/non-string-json-fields> for HTTPie non-string fields.

```
{
  "type": "Summer",
  "year": 2022,
  "country": "UK",
  "NOC": "GBR",
  "countries": 17,
  "disabilities_included": "Spinal injury",
  "end": "25-Sep-60",
  "events": "113",
  "participants_f": null,
  "host": "London",
  "male": null,
  "participants_m": 209,
```

```
"region": "ITA",  
"sports": "8",  
"start": "18-Sep-60"  
}
```

Take a note of the event id as you need it in the next request. Replace 28 with the id number that was returned.

- PATCH to update the new event: <http://127.0.0.1:5000/events/28> In the body select 'raw' and 'JSON' and enter

```
{  
  "countries": 21,  
  "end": "25-Sep-22",  
  "start": "18-Sep-22",  
  "year": 2022  
}
```

Further steps

The above should be just sufficient to allow you to create the REST API routes for your application.

Week 5 will consider:

- Handling errors
- Authentication

There are also Python and Flask libraries that are aimed at helping you to build REST APIs in Flask such as Flask-Restful that you can explore if you prefer.

Examples and tutorials

There are more links in the reading list.

- [RealPython REST API example](#)
- [REST API examples with Flask + SQLAlchemy](#)
- [Miguel Grinberg's Flask mega tutorial Chapter 23, APIs](#)