**Description: Create an application server that will implement a RESTful API.**

The API should allow a user to perform CRUD operations on an entity (eg a book)

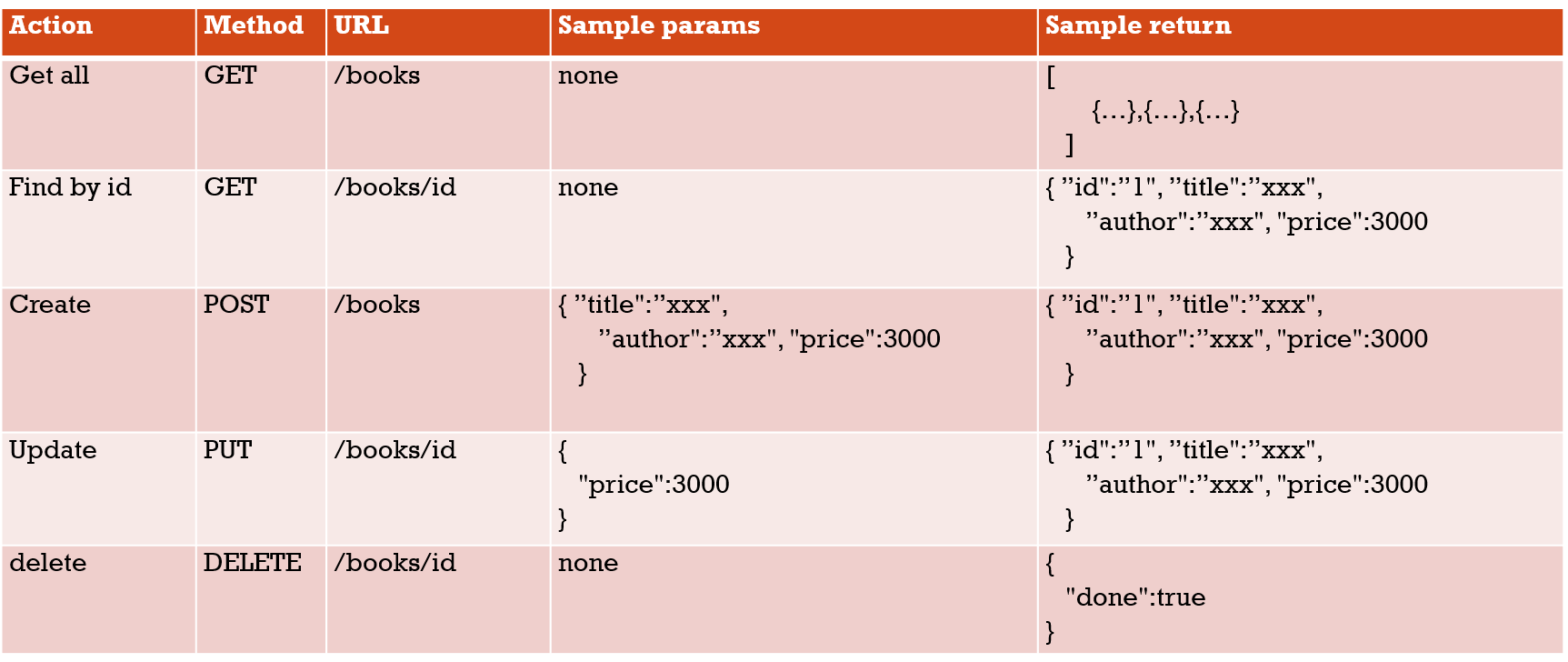
Test it using CURL

**Suggested steps:**

• Design the REST API (create a document)

1. Using the skills you learned in week 05 create a REST API for an entity of your choice

*This means “make a plan”… set out the functionality required and how the different urls will appear example:*



• Create a virtual environment to run a Flask server

*Now we start a-coding on the console for the moment!*

*Video Lecture 8.1 cover the virtual environment set up steps.*

1. Create a directory where this server is going to run

*Create a folder*

3. Create a Virtual environment

*λ python -m venv venv*

4. Run the virtual environment

*.\venv\Scripts\activate.bat*

5. Install Flask

*(venv) λ pip freeze*

*(venv) λ pip install flask*

6. Save the package list in a file called requirements.txt

*(venv) λ pip freeze*

*click==7.1.2*

*Flask==1.1.2*

*itsdangerous==1.1.0*

*Jinja2==2.11.2*

*MarkupSafe==1.1.1*

*Werkzeug==1.0.1*

*(venv) λ pip freeze > requirements.txt*

*(venv) λ ls*

*(venv) λ more requirements.txt*

7. Create a .gitignore file and put in venv/

• Create a basic Flask Server

8. Create a file called rest\_server.py

Sample minimum code

from flask import Flask, url\_for, request, redirect, abort

app = Flask(\_\_name\_\_, static\_url\_path='', static\_folder='staticpages')

@app.route('/')

def index():

   return  "is this thing on?"

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(debug=True)

9. Make a basic server in it and test it

*This is done in two steps*

*(venv) λ python rest\_server.py*

*Will start the server, a message appears on the console.*

*Second part – cut-and paste then open up the url on a browser*

*Also: test with curl*

*λ curl http://127.0.0.1:5000/*

*is this thing on?*

• Modify the Flask Server to deal with the required URL mappings (the functions can be just skeletons at this stage

10. Create a mapping and function for each of the possible API calls

*Refer to your original plan!*

* Implement getAll and test it

*Write code something like this. Save it and make sure it appears on the browser*

@app.route('/books')

def getAll():

   return  "this means the getAll() function was called"

*λ curl http://127.0.0.1:5000/books*

*this means the getAll() function was called*

*then replace the code with this*

@app.route('/books')

def getAll():

   #return  "this means the getAll() function was called"

   return jsonify(books)

*A curl as above will print out the book array to the screen.*

• Implement findById and test it

11. Create an array at the top of the file to store the data

12. Create a variable called nextId

13. Implement the getAll function by returning the array

14. Implement the findById function, by searching in the array for an object that has a matching id and return that object

@app.route('/books/<int:id>')

def findById(id):

    foundBooks = list(filter (lambda t : t["id"]== id, books))

#lambda function convert and filter the list of books, go through every object in the array

# id was passed in, books is the object

    #print(foundBooks)

    if len(foundBooks) == 0: #if ya find nothing with that id,  code 204 shows up on the server

        return jsonify({}) , 204

    return jsonify(foundBooks[0])

• Implement create and test it

*Basic create to start with looks like this.*

@app.route('/books', methods=['POST'])

def create():

   return "this means the create() function was called"

*λ CURL -X POST http://127.0.0.1:5000/books*

*this means the create() function was called*

15. Implement the create function, create an object with data in it, and store it in the array, make sure that it does not have an id that already exists (eg use the nextId variable)

*That is a fairly long piece of code*

16. Return that object

17. Test this (you can’t use the browser to test it, curl instead)

*λ curl -X POST -H "content-type:application/json" -d "{\"Title\": \"A Treatise \", \"Author\": \"some white guy\", \"Price\":15}" http://127.0.0.1:5000/books*

*YOU GOTTA HAVE* ***CONTENT TYPE*** *HEADER info in there too or you cannot create correctly on the console!!*

*curl http://127.0.0.1:5000/books*

• Implement update and test it

@app.route('/books/<int:id>', methods=['PUT'])

def update(id):

   return  "this means the update(id) function was called "+ str(id)

*λ CURL -X PUT http://127.0.0.1:5000/books/565*

*this means the update(id) function was called565*

18. Check that the user posted a JSON object (return error if they did not),

if not request.json: #abort the request if it is not in the correct json formt

        abort(400)

set the contents of the object to be the values from the JSON object that was uploaded.

  if 'Title' in request.json:

        currentBook['Title'] = request.json['Title']

    if 'Author' in request.json:

        currentBook['Author'] = request.json['Author']

    if 'Price' in request.json:

        currentBook['Price'] = request.json['Price']

    return jsonify(currentBook)

*λ curl -X PUT -d "{\"Title\":\"Overwrite Orginal Title\", \"Price\":9}" -H "content-type:application/json" http://127.0.0.1:5000/books/1*

• Implement delete and test it

20. Implement the Delete function a. Find the object you wish to delete and remove it from the array

@app.route('/books/<int:id>', methods=['DELETE'])

def delete(id):

   return  "this means the delete(id) function was called with id  "+ str(id)

*λ CURL -X DELETE http://127.0.0.1:5000/books/39*

*this means the delete(id) function was called39*

*After the more complexcode was added, then λ CURL -X DELETE http://127.0.0.1:5000/books/3*

*{*

*"done": true}*