Building a REST API with Flask

Doug Lay, CTO, DinnerTime Inc.

Dave Hopkins, Sr. Programmer, Welch Medical Library, JHU

March 26, 2018

What is an API?

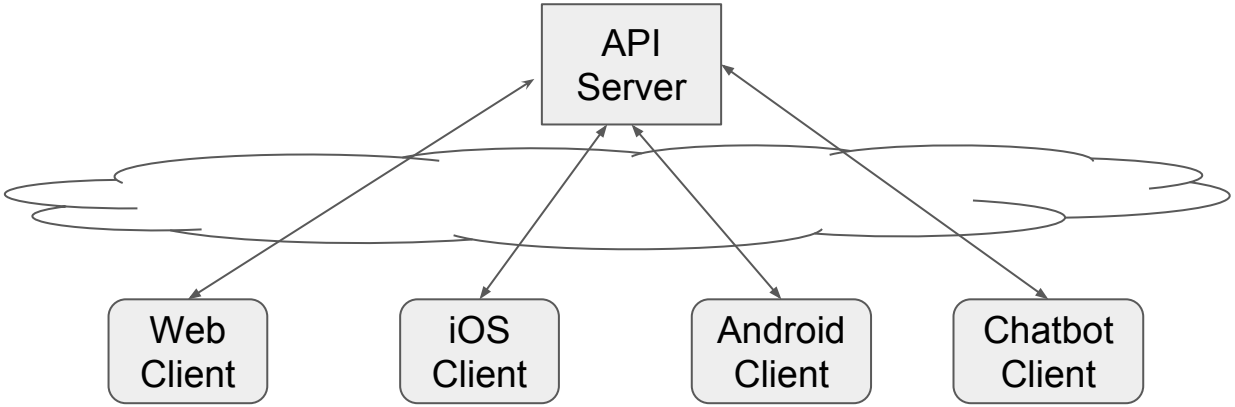
Application Programming Interface

Wikipedia: In general terms,[an API] is a set of clearly defined methods of communication between various software components.

Several types of API:

- Libraries and Frameworks
- Operating Systems
- Remote APIs
  - **Web APIs**

Remote APIs enable distributed computing



To work in a distributed, rapidly-changing world, a remote API needs to be independent of platform and programming language.

History of Remote APIs

1980s+ - Remote Procedure Calls - language/OS specific

1990s+ - Distributed Objects (CORBA, DCOM, EJB) - more language/OS independent, very complex

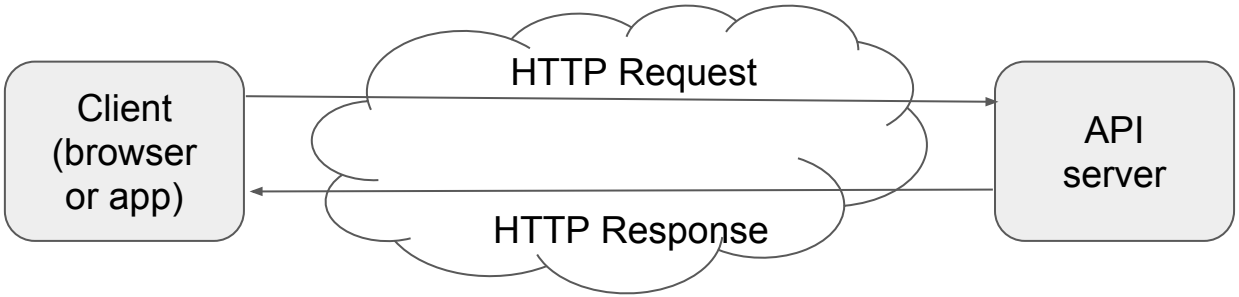
2000s+ - Web-based APIs - simpler, completely language/OS independent

- XML-RPC
- SOAP
- Web Services
- **REST**

REST won out over the others in most instances, probably because it was simpler

2010s+ - (Possible) improvements on REST (GraphQL, Falcor)

REST uses the Hypertext Transfer Protocol (HTTP)



Each individual request/response cycle is completely independent. All ‘state’ information must be sent with each request

REST stands for Representational State Transfer. Almost no one knows what that means.

Structure of a HTTP request

- URL
  - Protocol
  - Domain/Port
  - Path
  - Querystring
- Method (GET/POST in Web browsers, PUT and DELETE also used for REST)
- Headers (Content-type, Authentication, Cache-control, Cookie, etc)
- Data (not included in GET or DELETE requests)

Structure of a HTTP response

- Status Code
  - 2xx: success
  - 3xx: redirection
  - 4xx: client error (usually Not Found or Access Denied)
  - 5xx: server error
- Headers (same as for request with some additional headers like Location)
- Content
  - HTML, CSS, image, Javascript, JSON

REST uses URLs to provide access to resources

Collection URL: provides access to a collection of resources

https://my.api/api/v1/widgets/

Item URL: provides access to an item within the collection

https://my.api/api/v1/widgets/10

HTTP methods specify actions on resources

Method	**Type of URL**	**Action**
GET	Collection	Show collection
GET	Item	Show item
POST	Collection	Add item to collection
PUT	Item	Update item in collection
DELETE	Item	Delete item from collection

REST APIs usually transfer data in JSON format

JSON (https://www.json.org) is a very simple format for representing structured data.

```
{
  "id"    : 1,
  "name" : "Doug",
  "is_staff" : true,
  "classes" : ["Databases","APIs"]
}
```

(XML is sometimes available as an alternative)

Flask

Microframework for building (server-side) Web applications in Python “one drop at a time”

http://flask.pocoo.org/

Core of Flask is very minimal, but easy to extend

Popular choice for building REST APIs, especially simple APIs that may need more functionality over time.

Okay, let’s get coding

Download or clone from GitHub URL:

https://github.com/davehops/bucketlist

Follow installation instructions in the README.

3 options:

- OSX/Linux
- Windows with minTTY or other Cygwin shell
- Windows with standard command prompt

Enabling URLs for our API

We need:

- A URL for our collection
- A URL *pattern* for our items

Flask allows us to link actions to URLs and patterns using methods with *decorators*:

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

```
@app.route('/bucketlists/<int:id>', methods=['GET', 'PUT', 'DELETE'])
def bucketlist_manipulation(id, **kwargs):
```

POST to add new resources

POST /bucketlists

- Send information needed to create a new resource in JSON format, in the data section of the HTTP request.
- Return status code of 201 if resource is successfully created in the data store.
- Return entire resource in JSON format, in the content section of the HTTP response.

Let’s write code to do this inside our bucketlists() method, first checking to see that the HTTP request method is POST.

```
if request.method == "POST":
    name = str(request.data.get('name', ''))
    if name:
        bucketlist = Bucketlist(name=name)
        bucketlist.save()
        response = jsonify({
            'id': bucketlist.id,
            'name': bucketlist.name,
            'date_created': bucketlist.date_created,
            'date_modified': bucketlist.date_modified
        })
        response.status_code = 201
        return response
```

GET to view the collection

GET /bucketlists

- Return all resources in the collection, in JSON format (array of JSON objects)
- Return status code of 200 on success.

Let’s add code to our bucketlists() method to do this, checking first to see that the request method is GET.

```
else:
    # GET
    bucketlists = Bucketlist.get_all()
    results = []

    for bucketlist in bucketlists:
        obj = {
            'id': bucketlist.id,
            'name': bucketlist.name,
            'date_created': bucketlist.date_created,
            'date_modified': bucketlist.date_modified
        }
        results.append(obj)
    response = jsonify(results)
    response.status_code = 200
    return response
```

Let’s test

1. Run built-in unit tests
   a. python test_bucketlist.py
2. Test using a REST client
   a. Download and install RESTlet Chrome extension (Google for it)
   b. Fire up Flask using *flask run*
   c. Within RESTlet, POST to http://localhost:8000/bucketlists. Pass in a name attribute in the data section of the request, in JSON format
3. GET requests can be tested in a regular Web browser
   a. With Flask still running, browse to http://localhost:8000/bucketlists

Let’s populate our handler for Item URLs

/bucketlists/<*id*>

- First, we make sure that the id in the URL matches the id of a resource in our collection.
- Return a 404 (Not Found) error if the id does not match.

```
@app.route('/bucketlists/<int:id>', methods=['GET', 'PUT', 'DELETE'])
def bucketlist_manipulation(id, **kwargs):
    bucketlist = Bucketlist.query.filter_by(id=id).first()
    if not bucketlist:
        abort(404)
```

PUT on an item URL

PUT /bucketlists/<*id*>

- Send information needed to update the resource in JSON format, in the data section of the HTTP request.
- Update the resource in the data store
- Return status code of 200 on success
- Return entire resource in JSON format, in the content section of the HTTP response.

Let’s write code to do this inside our bucketlist_manipulation() method, first checking to see that the HTTP request method is PUT.

```
if request.method == 'PUT':
    name = str(request.data.get('name', ''))
    bucketlist.name = name
    bucketlist.save()
    response = jsonify({
        'id': bucketlist.id,
        'name': bucketlist.name,
        'date_created': bucketlist.date_created,
        'date_modified': bucketlist.date_modified
    })
    response.status_code = 200
    return response
```

DELETE on an item URL

DELETE /bucketlists/<*id*>

- Delete the resource from the data store
- Return status code of 200 on success
- Return status message in content section of response, indicating resource was deleted.

Let’s add some code to our bucketlist_manipulation() method.

```
elif request.method == 'DELETE':
    msg = "bucketlist {} deleted successfully".format(bucketlist.id)
    bucketlist.delete()
    response = jsonify({
        'message': msg
    })
    response.status_code = 200
    return response
```

GET on an item URL

GET /bucketlists/<*id*>

Return the entire resource specified by the id, in JSON format, in the response content area.

Let’s add some code to our bucketlist_manipulation() method.

```
else:
    # GET
    response = jsonify({
        'id': bucketlist.id,
        'name': bucketlist.name,
        'date_created': bucketlist.date_created,
        'date_modified': bucketlist.date_modified
    })
    response.status_code = 200
    return response
```

Let’s test again

What’s missing? Lots.

Authentication / Authorization

Filtering

Sorting

Pagination

Error Handling

Documentation

Client-side code