Web Application with Flask

Version: 7.0

Trainer:

Course Code: M394

Website: www.tertiarycourses.com.my

Email: sales@tertiarycourses.com.my

Tel: +603 7931 9658

About the Trainer

Let's Know Each Other...

Ground Rules

- Set your mobile phone to silent mode
- Participate actively in the class. No question is stupid.
- Mutual respect. Agree to disagree.
- One conversation at one time.
- Be punctual. Back from breaks on time.
- Exit the class silently if you need to step out for phone call, toilet break etc.
- 75% attendance is required

Course Outline

Topic 1 Get Started on Flask

- Use Flask as Web API Middleware
- Create a Simple Flask Web API
- Test the Flask Web API on Postman

Topic 2 Render Web Template

- Utilize Flask Templating Framework
- Integrate Data and Variables to Template
- Implement Control Structures and Functions
- Integrate Static Files

Topic 3 Create REST API

- Work with JSON Data
- HTTP Methods and Status Code
- Implement Variable Rules
- Test URL Rules

Course Outline

Topic 4: Deploy to Heroku

- Install Heroku CLI
- Install Git
- Deploying Flask app to Heroku

Access Google Classroom LMS

You can access more resources from Google classroom LMS

https://classroom.google.com/c/NTAwNzM0MjgzMjI2?cjc=pi6b3l6

Alternatively. You can goto https://classroom.google.com and enter the class code on the top right.

Google Class code: pi6b3l6

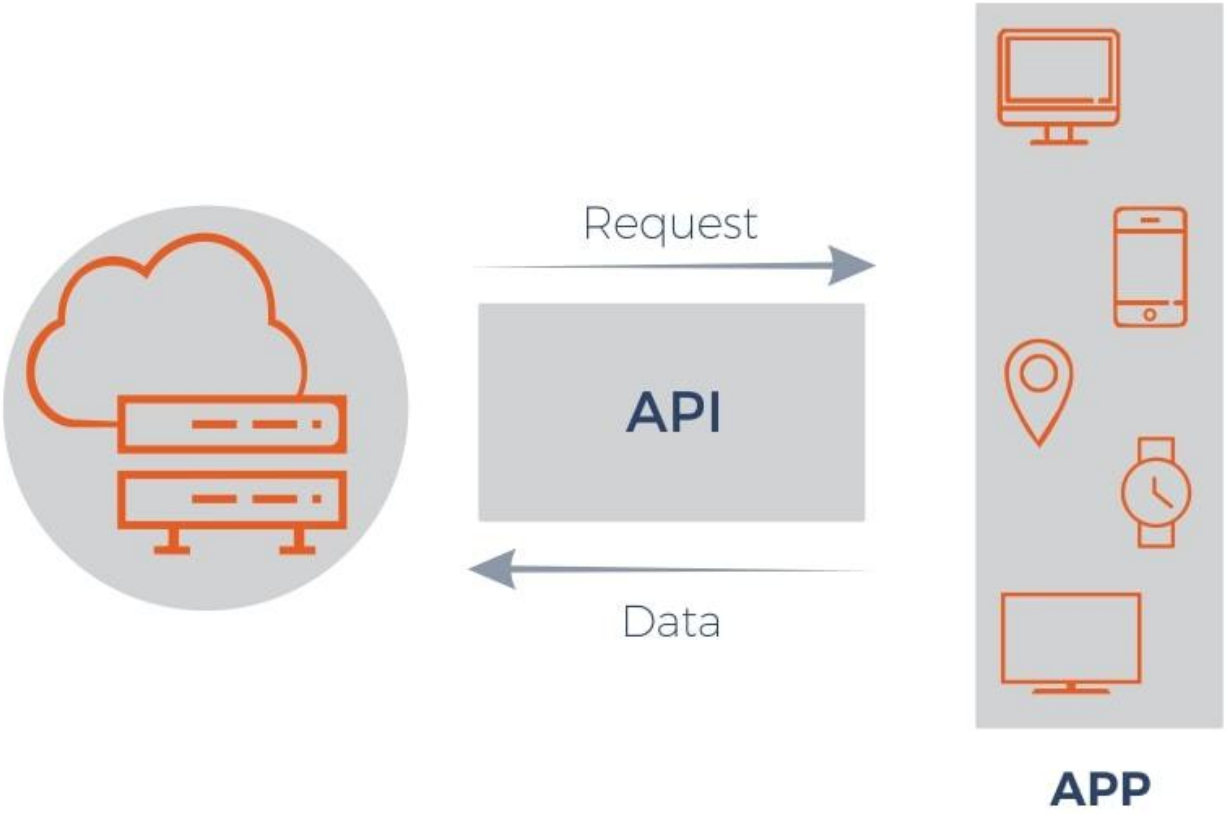
Topic 1
Get Started on Flask

What is API?

- An API (Application Programming Interface) is a set of features and rules that exist inside a software program (the application) enabling interaction with it through software - as opposed to a human user interface.
- The API can be seen as a simple contract (the interface) between the application offering it and other items, such as third-party software or hardware.
- In Web development, an API is generally a set of code features (e.g. methods, properties, events, and URLs) that a developer can use in their apps for interacting with components of a user's web browser, or other software/hardware on the user's computer, or third party websites and services.

Source: https://developer.mozilla.org/en-US/docs/Glossary/API

What is API?



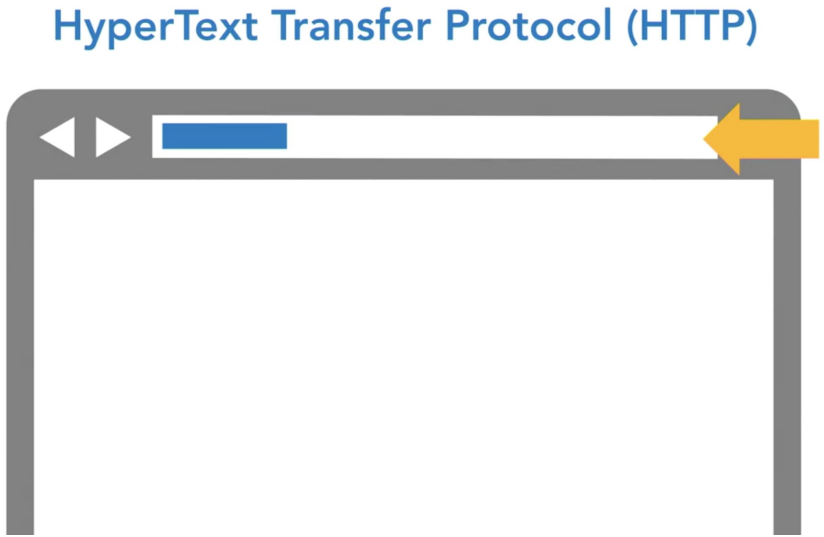
What is REST API?

- REST and API are acronyms for Representational State Transfer and Application Programming Interface.
- REST APIs are integral to web application development and are becoming mainstays of all web development
- The basic idea of REST is that a resource, e.g. a document, is transferred via well-recognized, language-agnostic, and reliably standardized client/server interactions.

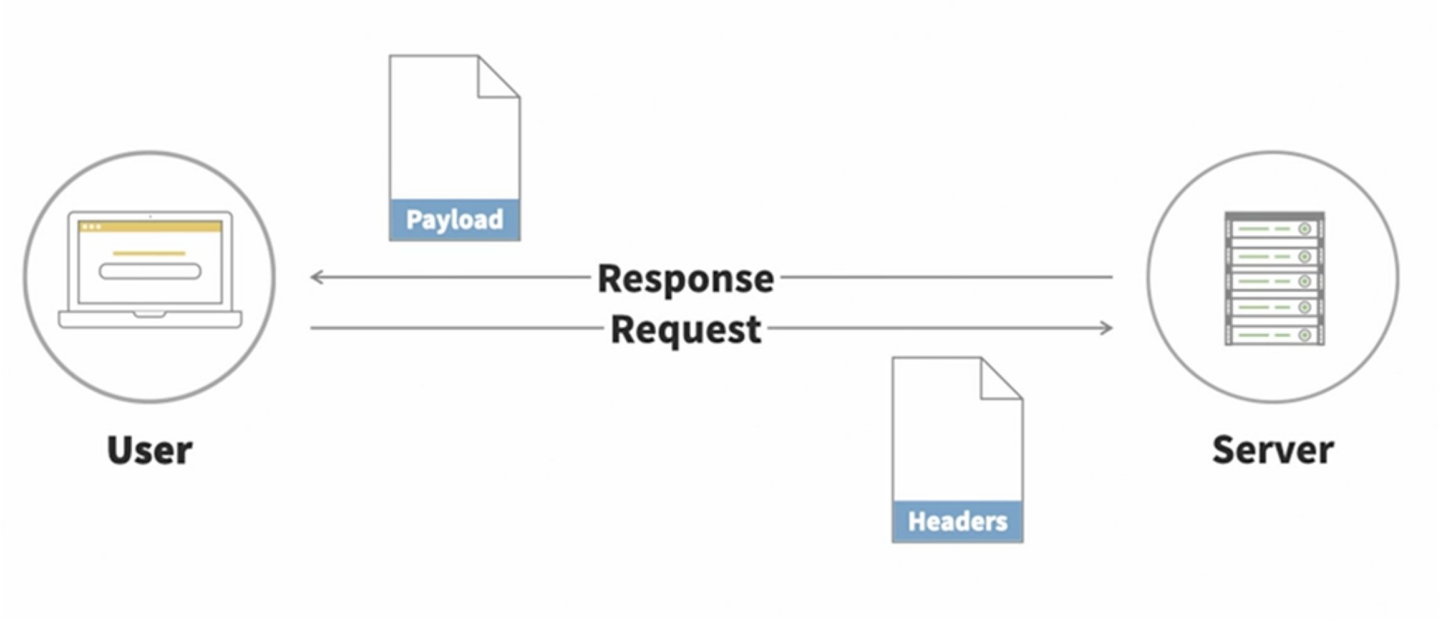
Source: https://developer.mozilla.org/en-US/docs/Glossary/REST

RESTFUL API

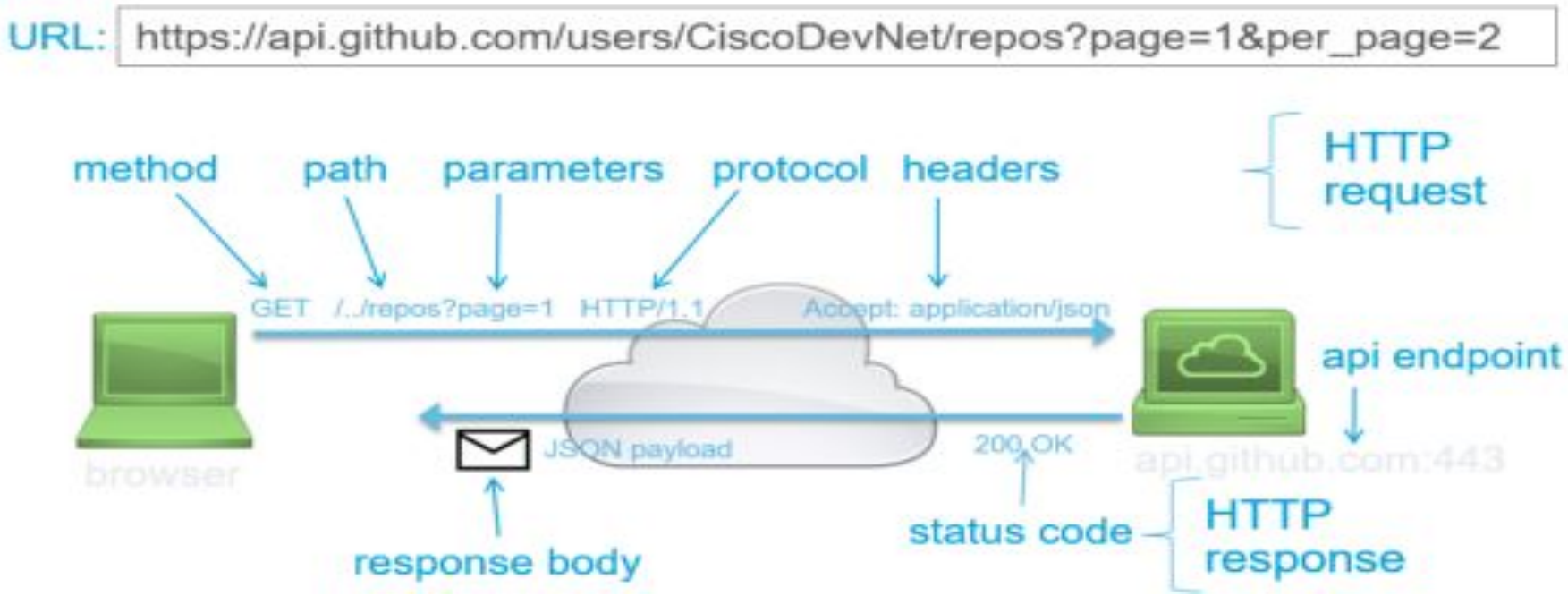
- If REST API send their requests through the HTTP, it is known as the RESTFUL API
- Beginners can assume a REST API means an HTTP service that can be called using standard web libraries and tools.



HTTP Request and Response

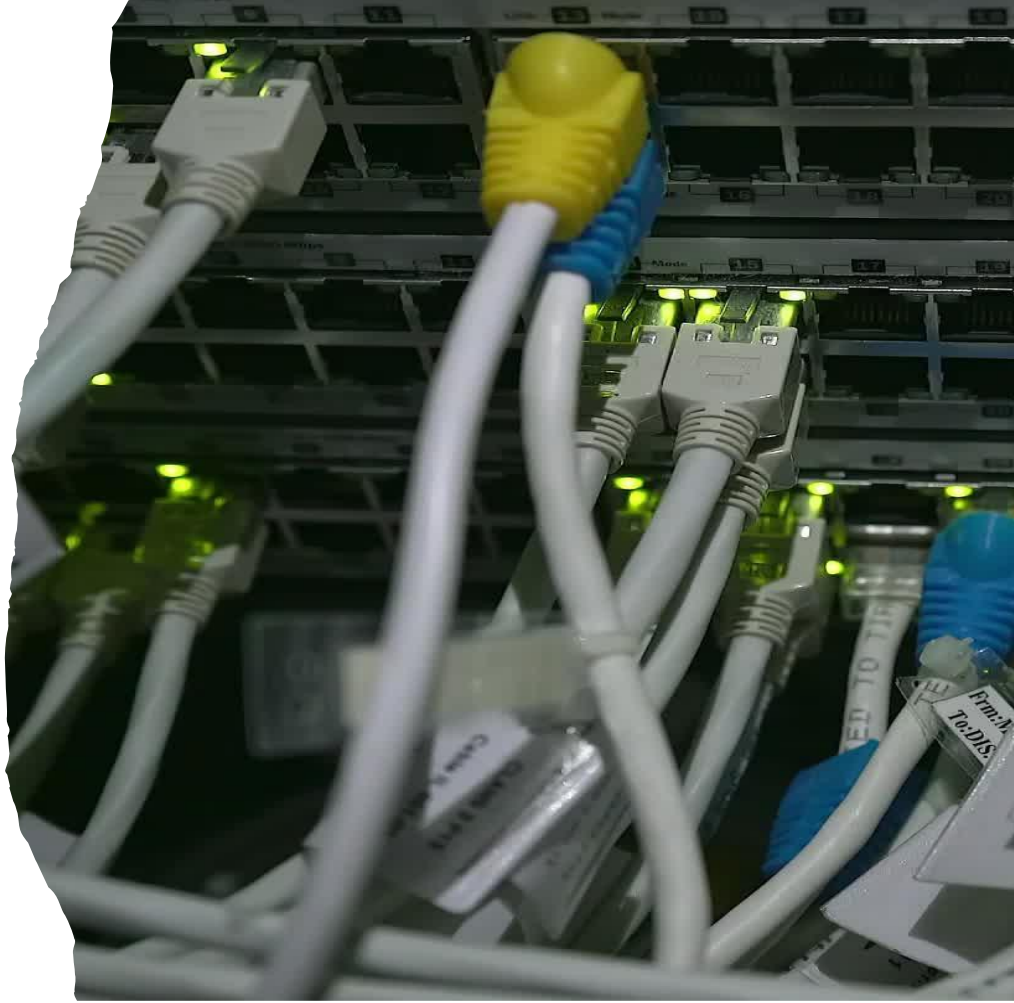


Anatomy of a REST API Query



Payload

Data sent between client and server

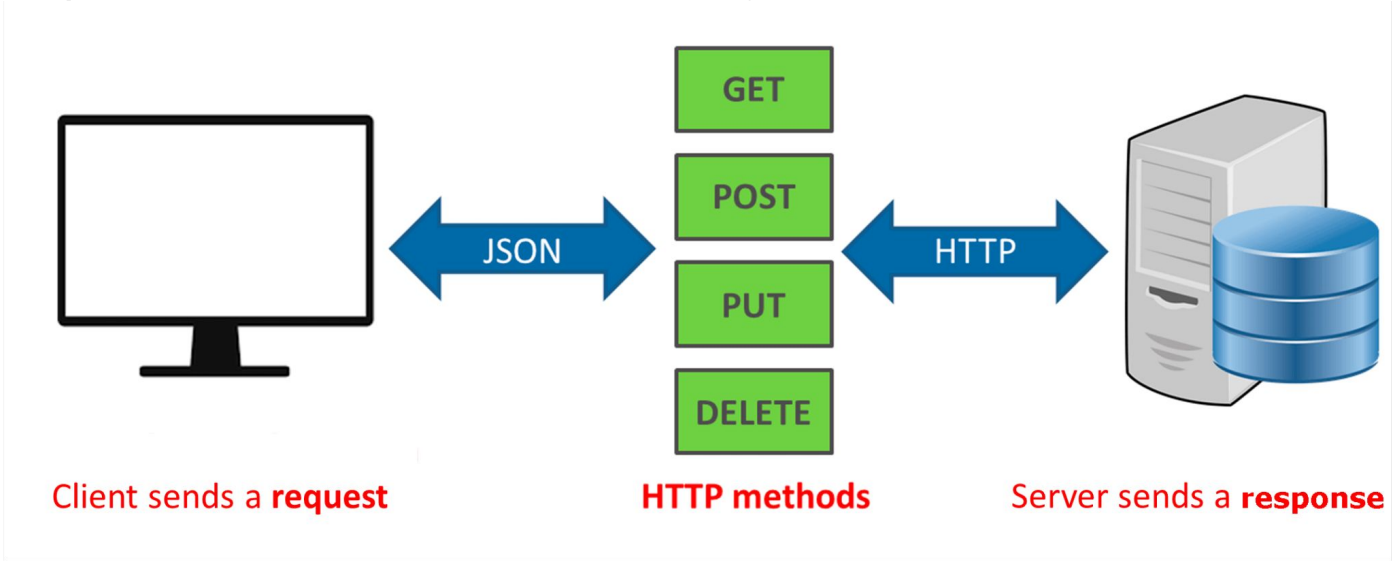


HTTP

- The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.
- HTTP works as a request-response protocol between a client and server.
- Example: A client (browser) sends an HTTP request to the server; then the server returns a response to the client.
- The response contains status information about the request and may also contain the requested content.

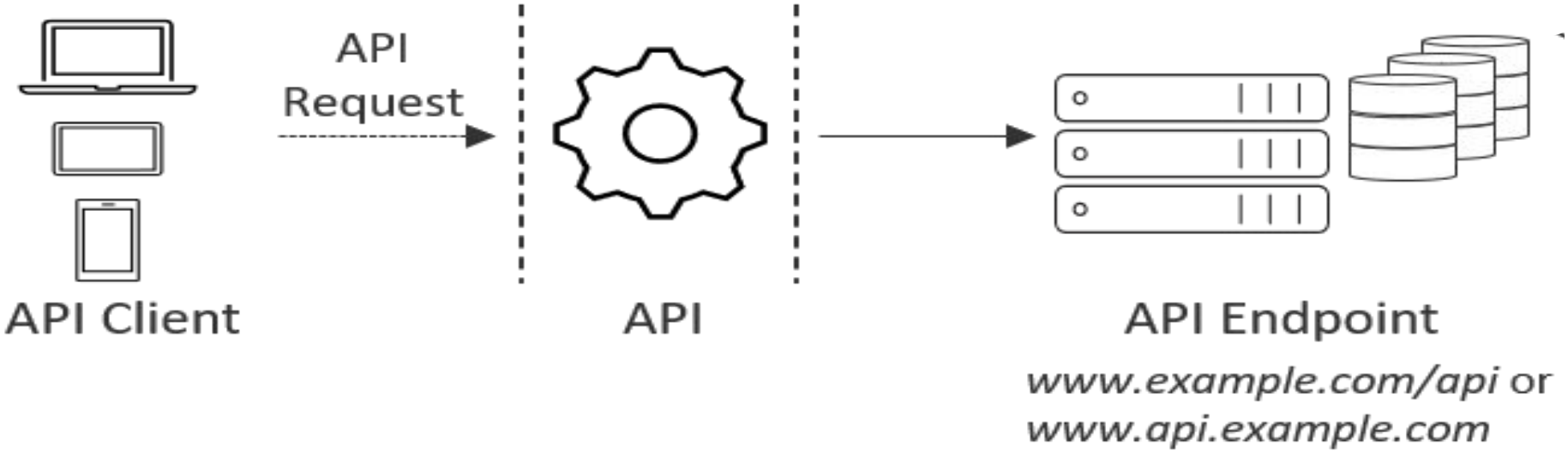
HTTP Methods

- The primary or most-commonly-used HTTP methods) are POST, GET, PUT, PATCH, and DELETE.
- These correspond to create, read, update, and delete (or CRUD) operations, respectively.



Endpoints

HTTPS endpoints are serverless functions that you call by sending requests from an HTTP client.



Examples of Paypal Endpoints

GET https://api-m.paypal.com/v1/catalogs/products

POST https://api-m.paypal.com/v1/catalogs/products

PATCH https://api-m.paypal.com/v1/catalogs/products/{product_id}

GET https://api-m.paypal.com/v1/catalogs/products/{product_id}

PUT https://api-m.paypal.com/v1/shipping/trackers/{id}

DELETE https://api-m.paypal.com/v2/invoicing/invoices/{invoice_id}

Source: https://developer.paypal.com/api/rest/

Example of RESTful API

```
import requests

url = "https://api2.binance.com/api/v3/ticker/24hr"

response = requests.request("GET", url)
print(response.json()[0])
print(response.json()[1])
```

What is Flask?

- Flask is a web application framework written in Python. It is developed by Armin Ronache
- You can use Flask as a middleware to create Web API, manage HTTP request management, and render template.
- http://flask.pocoo.org/

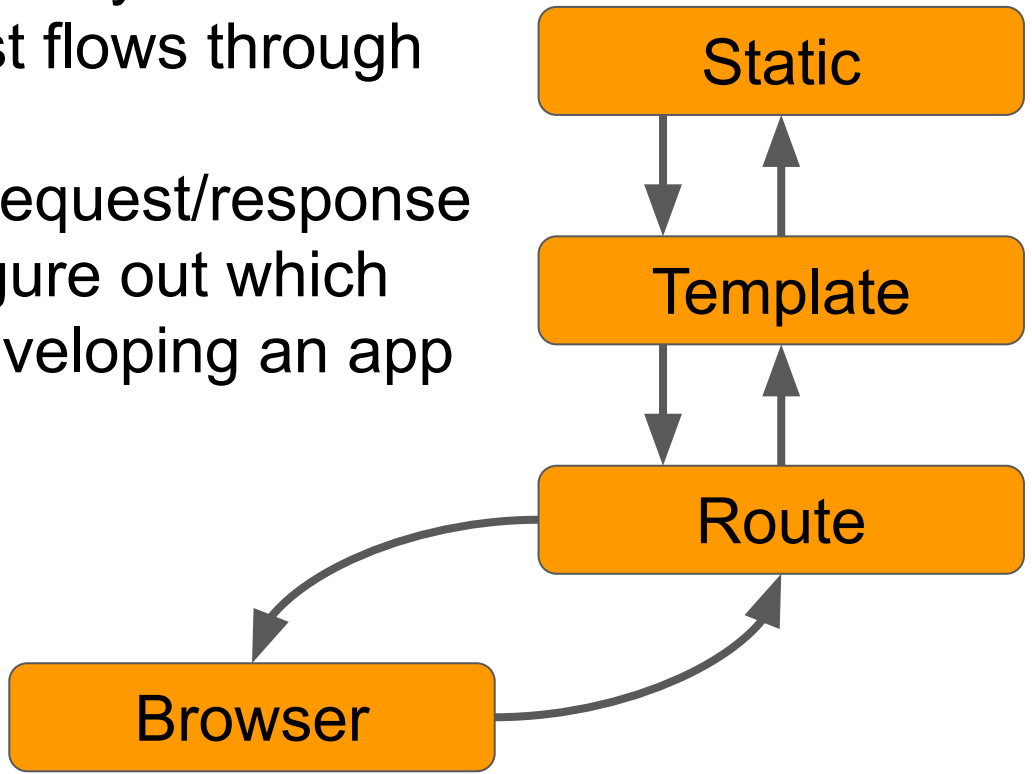
Install Flask

pip install flask (for Window)
or
pip3 install flask (for Mac)

For upgrade, pip install --upgrade flask

Flask Request Response Cycle

- The request/response cycle traces how a user's request flows through the app.
- Understanding the request/response cycle is helpful to figure out which files to edit when developing an app



Create Your First Flask Web API

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'


if __name__ == '__main__':
    app.run()
```

Execute Your Flask File Locally

If you filename is "app.py", you can run like normal python file
python app.py

or you can do a flask run
export FLASK_APP=app.py (Mac)
set FLASK_APP=app.py (Window)
flask run

Once your run the code, it will start a local server on the default port 5000 http://127.0.0.1:5000/

Create Endpoints In Flask

@app.route is a route decorate that binds a function to a URL (endpoints)

```
@app.route('/')
def hello_world():
    return 'Hello World!'
```

```
@app.route('/super_simple')
def super_simple():
    return 'Hello from the Planetary API.'
```

Check the endpoints http://127.0.0.1:5000/super_simple

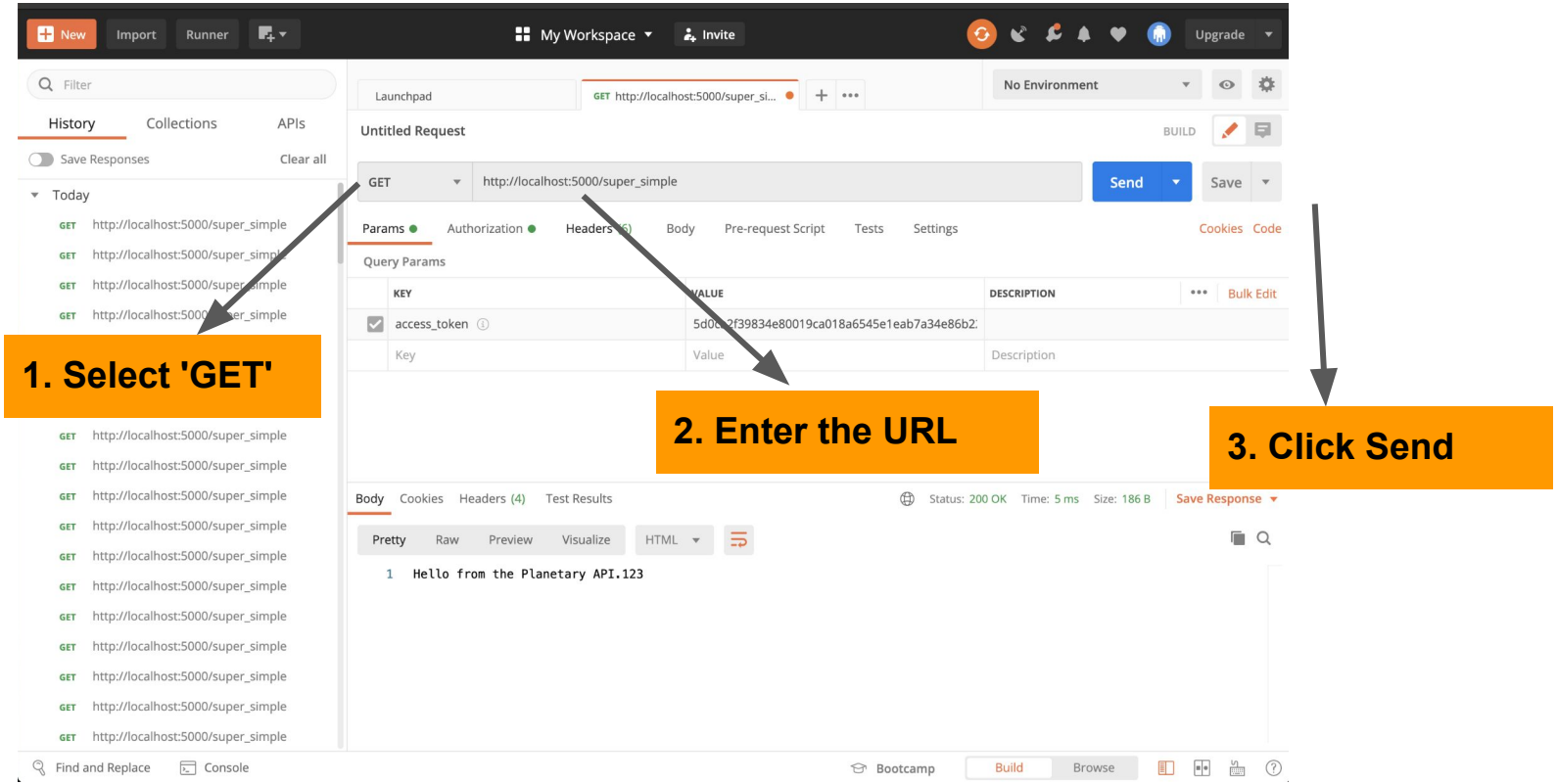
POSTMAN

- Postman is an API platform for building and using APIs.
- Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster

https://www.postman.com/

Test Out API on Postman

Use the GET http://localhost:5000/super_simple on POSTMAN



Activity: Create Web API in Flask

- Create an Web API endpoint "not_found" and return "Not Found"
- Test out the endpoint on Postman

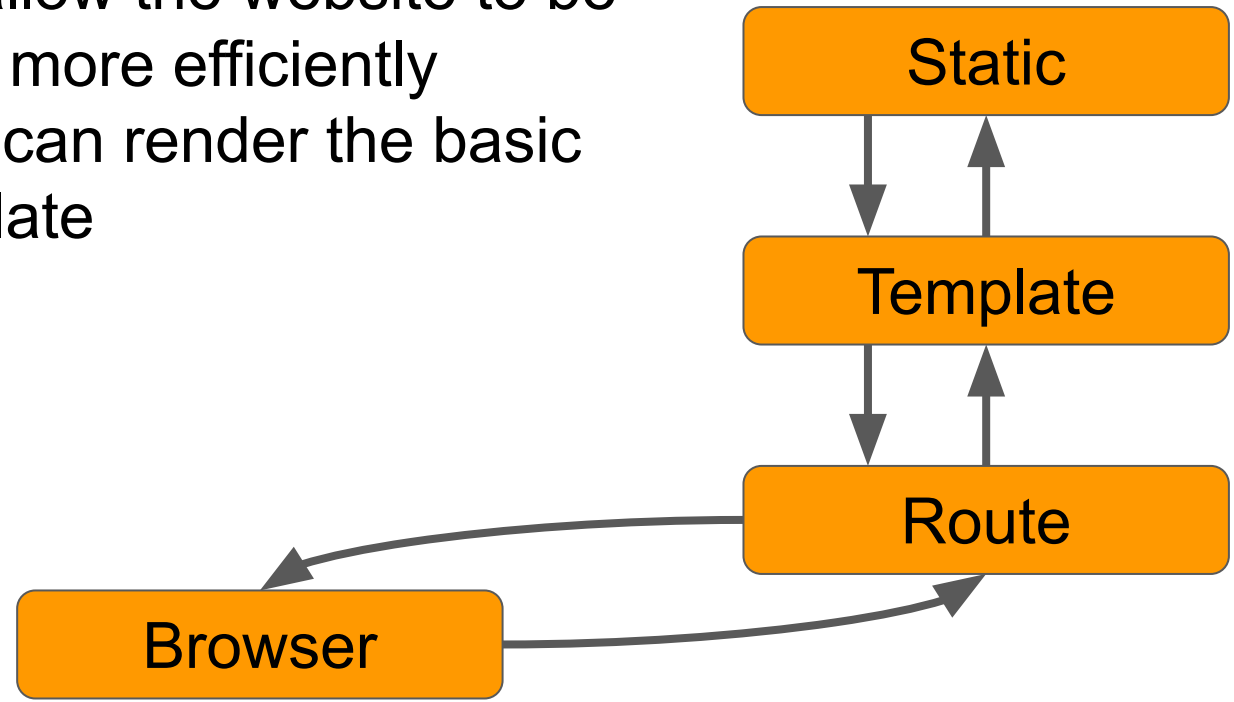
Topic 2
Render Web Template

Jinja Templating

Flask template is based on Jinja templating system.
http://jinja.pocoo.org/

Templating in Flask

- Templates allow the website to be constructed more efficiently
- Every page can render the basic layout template



Templates Directory

- Create a templates folder
- Add your HTML and CSS files to the folder

Index.html

```
<!DOCTYPE html>
<html>

  <head>
    <title>Flask</title>
  </head>

  <body>
        <h1>Hello Everyone</h1>
  </body>
```

Render Template

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route("/")
def index():
  return render_template("index.html")

@app.route("/aboutus")
def aboutus():
  return render_template("aboutus.html")

if __name__ == "__main__":
  app.run(debug=True)
```

Passing Variables in Template

Variables in HTML templates are enclosed by {{ …}}

Eg

```
<h1>{{title}}</h1>
<p>{{paragraph}}</p>
```

index2.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Learning Flask</title>
  </head>
  <body>
       <h1>{{title}}</h1>
     <p>{{paragraph}}</p>
  </body>
</html>
```

Render Template with Variables

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def index():
  return
render_template("index2.html",title="Flask",paragraph="Awesome"
)

if __name__ == "__main__":
  app.run(debug=True)
```

Pass in Variable from URL

Variable can also be passed from the URL

```
@app.route("/<title>")
def index(title):
    return render_template("index2.html", title=title,
paragraph="Awesome")
```

Extends

- The extends tag can be used to apply one template from another.
- It is efficient to apply the same template to multiple pages.

Eg

```
{% extends "layout.html" %}
```

Block

The content placeholder to the basic layout template

```
{% block content %}
{% endblock %}
```

The content can then be added to other template files Eg

```
{% block content %}
<p>Hello World</p>
{% endblock %}
```

Block Example

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def index():
    title = "Rendering"
    text = "Awesome day, learn a lot of stuff"
    return render_template("index3.html",title=title,text=text)

if __name__ == "__main__":
  app.run(debug=True)
```

index3.html

```
{% extends "layout.html" %}

{% block content %}
<h2>{{title}}</h2>
<p>{{text}}</p>
{% endblock %}
```

layout.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Learning Flask</title>
  </head>
  <body>
    <h1>Learning Flask</h1>
    {% block content %}
    {% endblock %}

  </body>
</html>
```

Activity: Block Content

Run 2_4b.appy, explain what you observe.

If-else

```
<title>Hello from Flask</title>
{% if name %}
  <h1>Hello {{ name }}!</h1>
{% else %}
  <h1>Hello, World!</h1>
{% endif %}
```

Note that no {{ }} require for the variable inside %

Activity: if-else

Code the following if-else:

If today = 1, show today is first day
If today = 30, show today is last day
else today is not first and last day

For Loop

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def index():
    fruit = ["Apple", "Banana", "Durian","Mango"]
    return render_template("index5.html", fruit=fruit)

if __name__ == "__main__":
   app.run(debug=True)
```

For Loop Template

```
{% extends "layout.html" %}

{% block content %}
<ul>
{% for a in fruit %}
<li>{{a}}</li>
{% endfor %}
</ul>
{% endblock %}
```

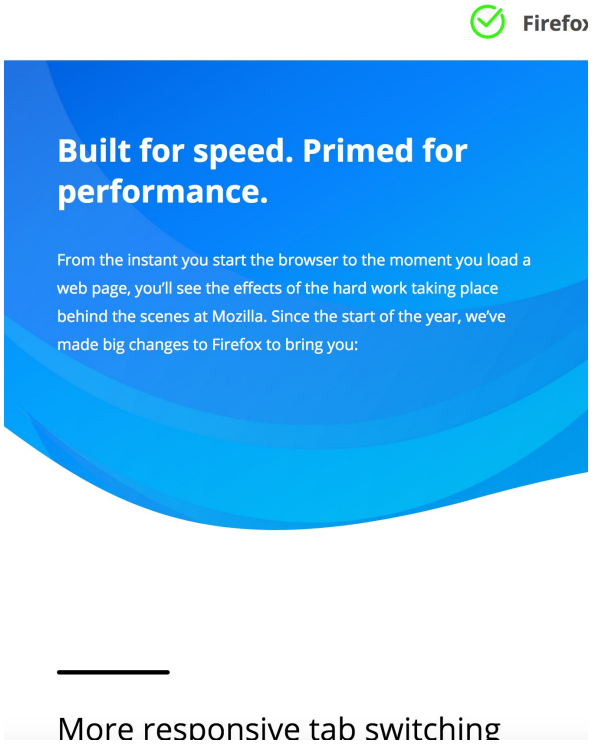
Activity: For Loop

Given
person = {"Ally":"apple","Alfred":"durian"}

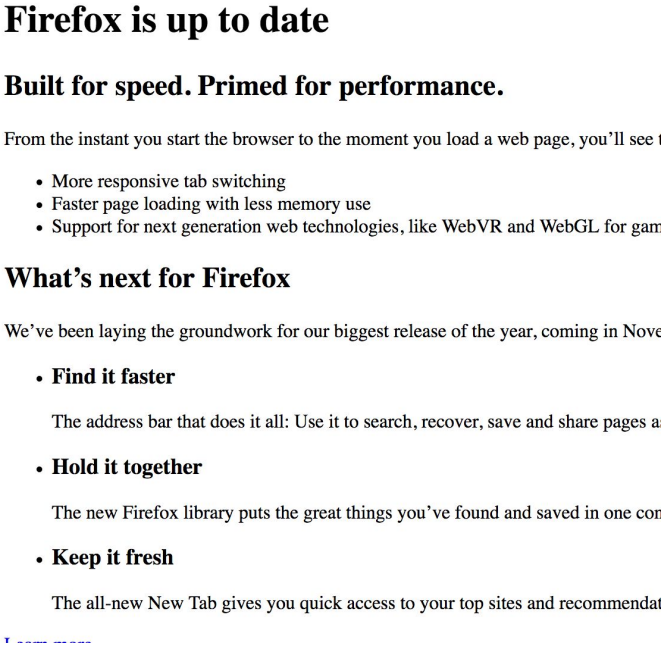
Code the following:

Ally love to eat apple
Alfred love to eat durian

CSS Styling



With CSS styling



Without CSS styling

Static Directory

- Create a static folder
- Add the CSS, Javascript and images files to the folder

Add CSS

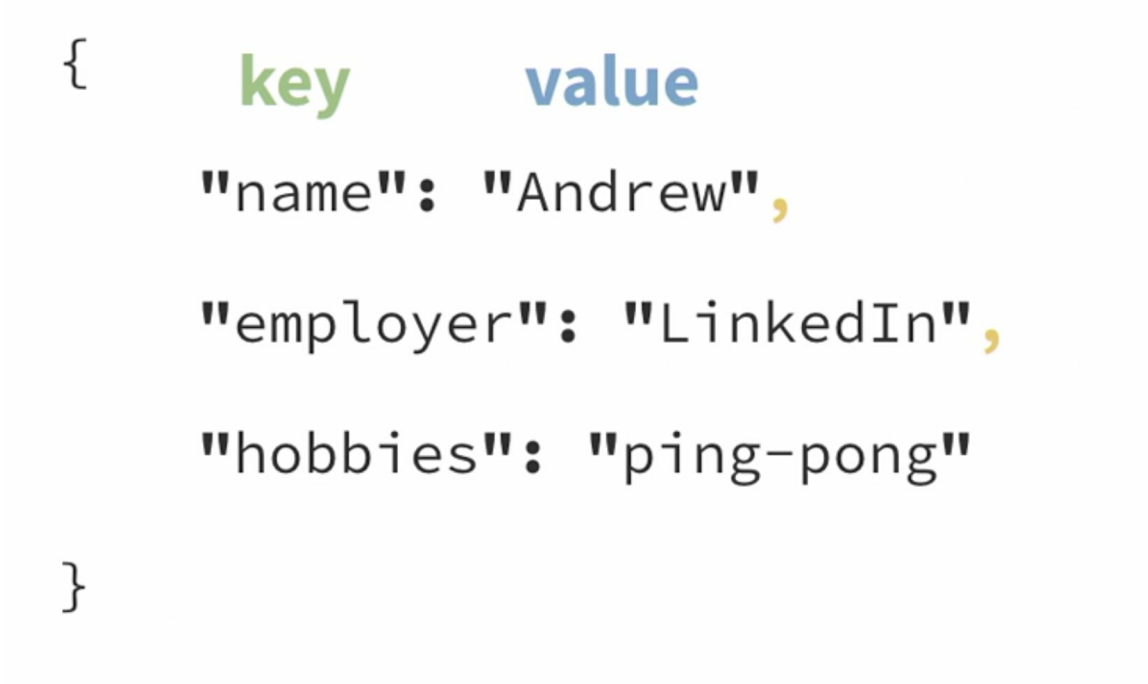
You can add the css file with the link tag

```
 <link href="{{ url_for('static', filename='css/style.css') }}" rel="stylesheet">
```

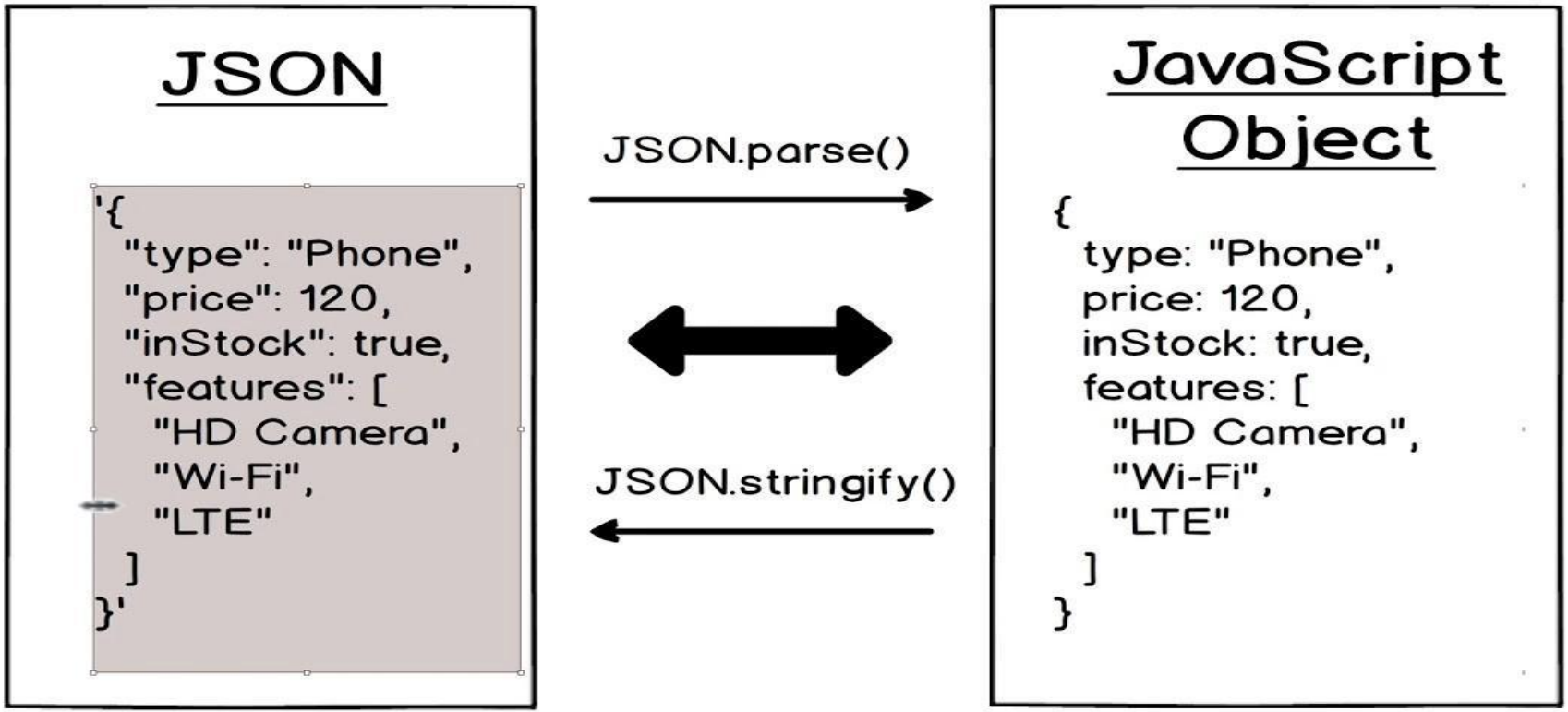
Topic 3
Create REST API

Payload in JSON Format

- API response payload is typically in JSON or XML format.
- JSON stands for Javascript Object Notation

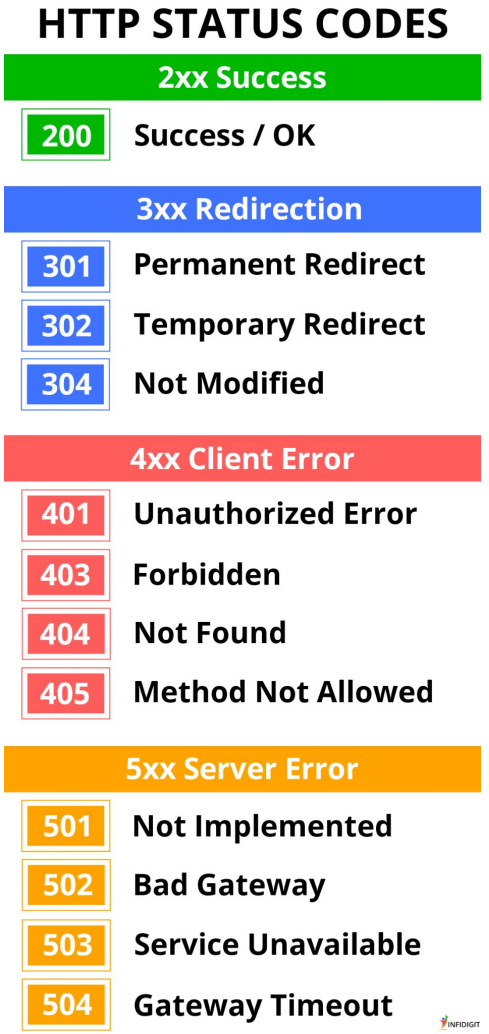


Transform JSON to Javascript Object



HTTP Status Code

- All web apps are based on a request-response mechanism
- Requests and responses have headers
- Headers contain useful but invisible metadata that is not obvious to end users
- The status code in the header will tell you whether you request was successful or not.



HTTP Response Codes

Code	Scope
100-199	Informational
200-299	Success
300-399	Redirect
400-499	Client Error
500-599	Server Error

Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

HTTP Response Codes: 2XX

2xx	Success
200	OK - Request was successful
201	Created - Resource was created
202	Action has started
204	No Content

Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

HTTP Response Codes: 3XX

3xx	Redirection
301	Moved permanently
302/303	Moved at this other URL
307	Temporarily redirect
308	Resume incomplete

Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

HTTP Response Codes: 4XX

4xx	Client Error
400	Bad request
401	Unauthorized
403	Forbidden
404	Not found
405	Method not allowed

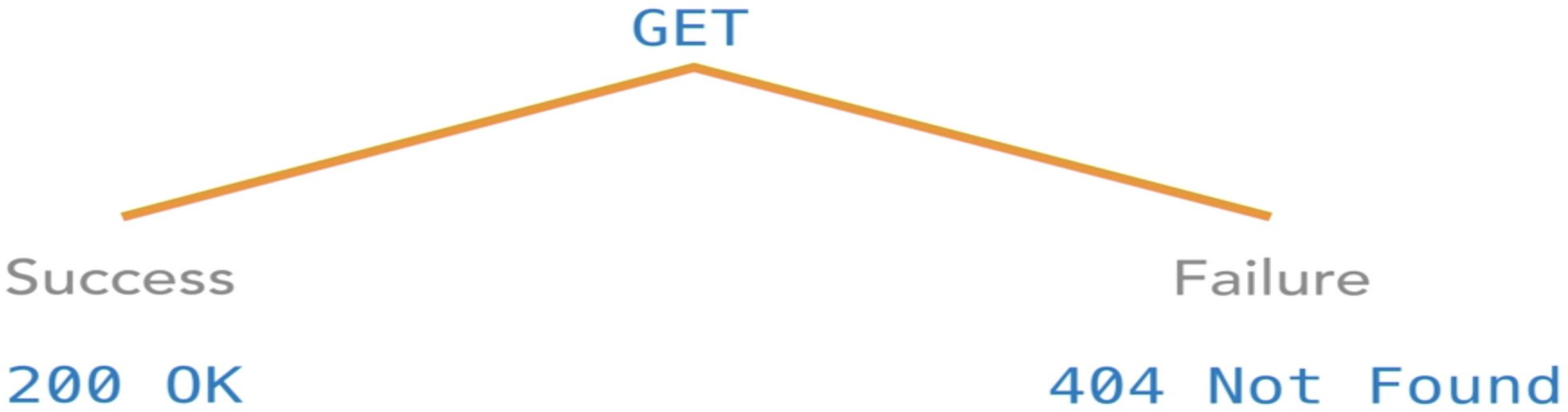
Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

HTTP Response Codes: 5XX

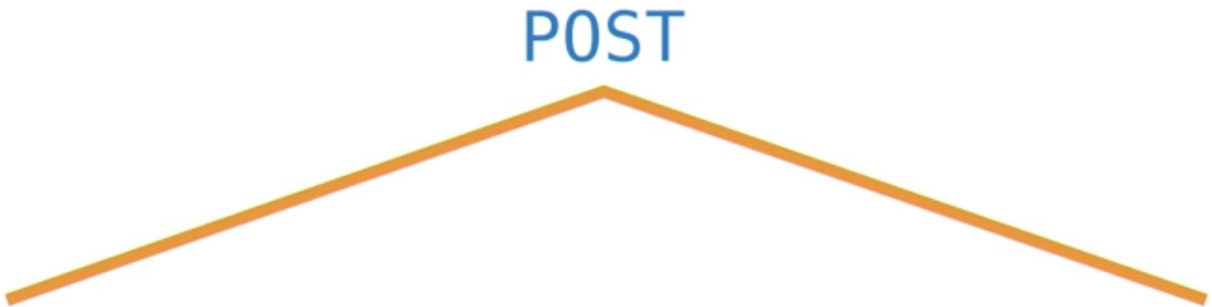
5xx	Server Error
500	Internal server error
502	Bad gateway
503	Service unavailable

Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Get Specific Resource if Available



Create a New Resource and Add it to a Resource



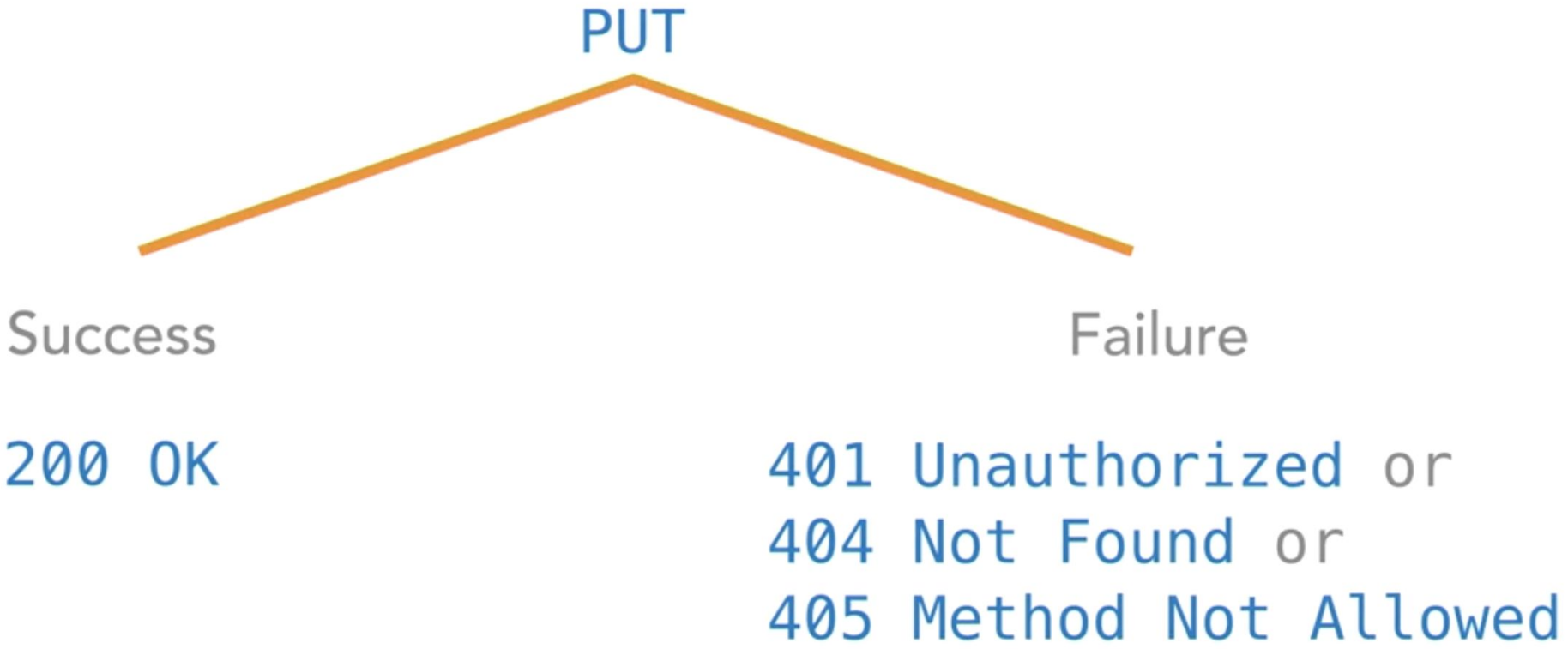
Success

201 Created

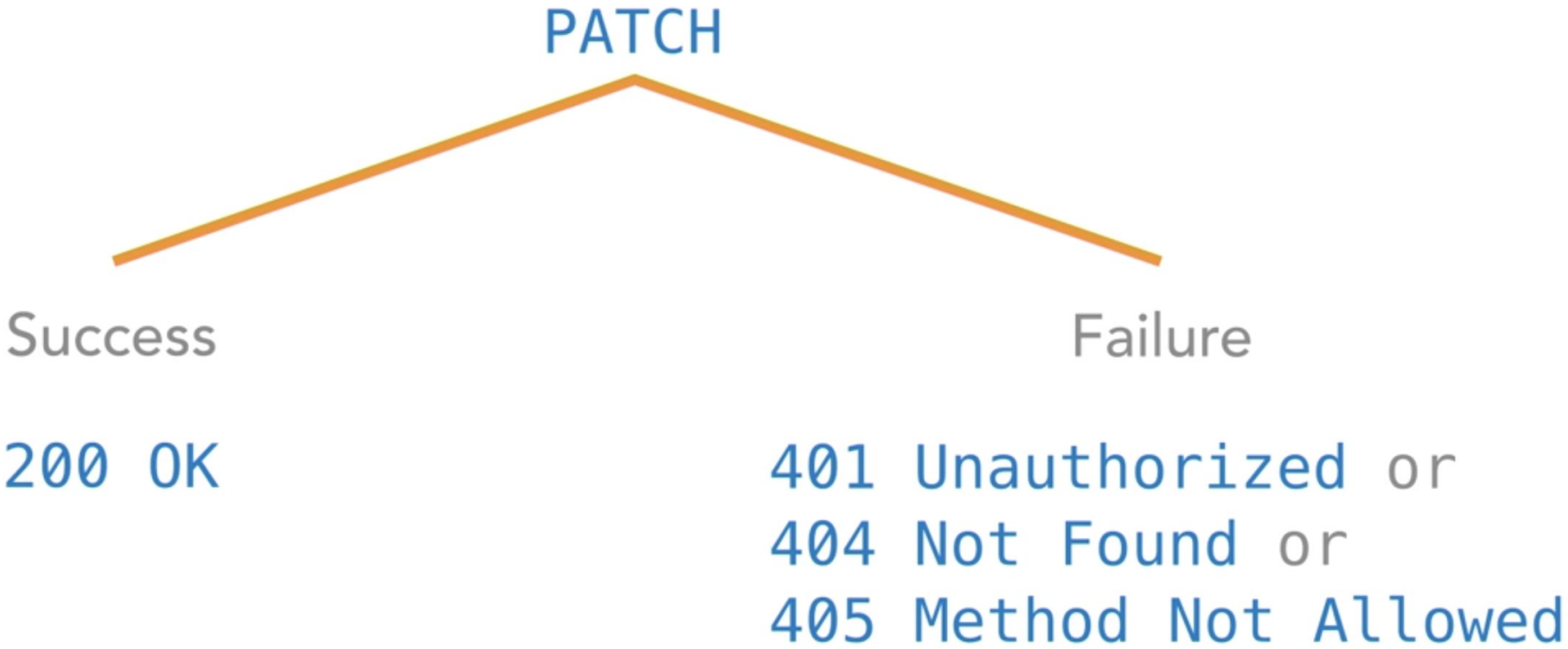
Failure

401 Unauthorized or
409 Conflict or
404 Not Found

Update an Existing Singleton Resource Based on ID



Modify an Existing Singleton Resource Based on ID



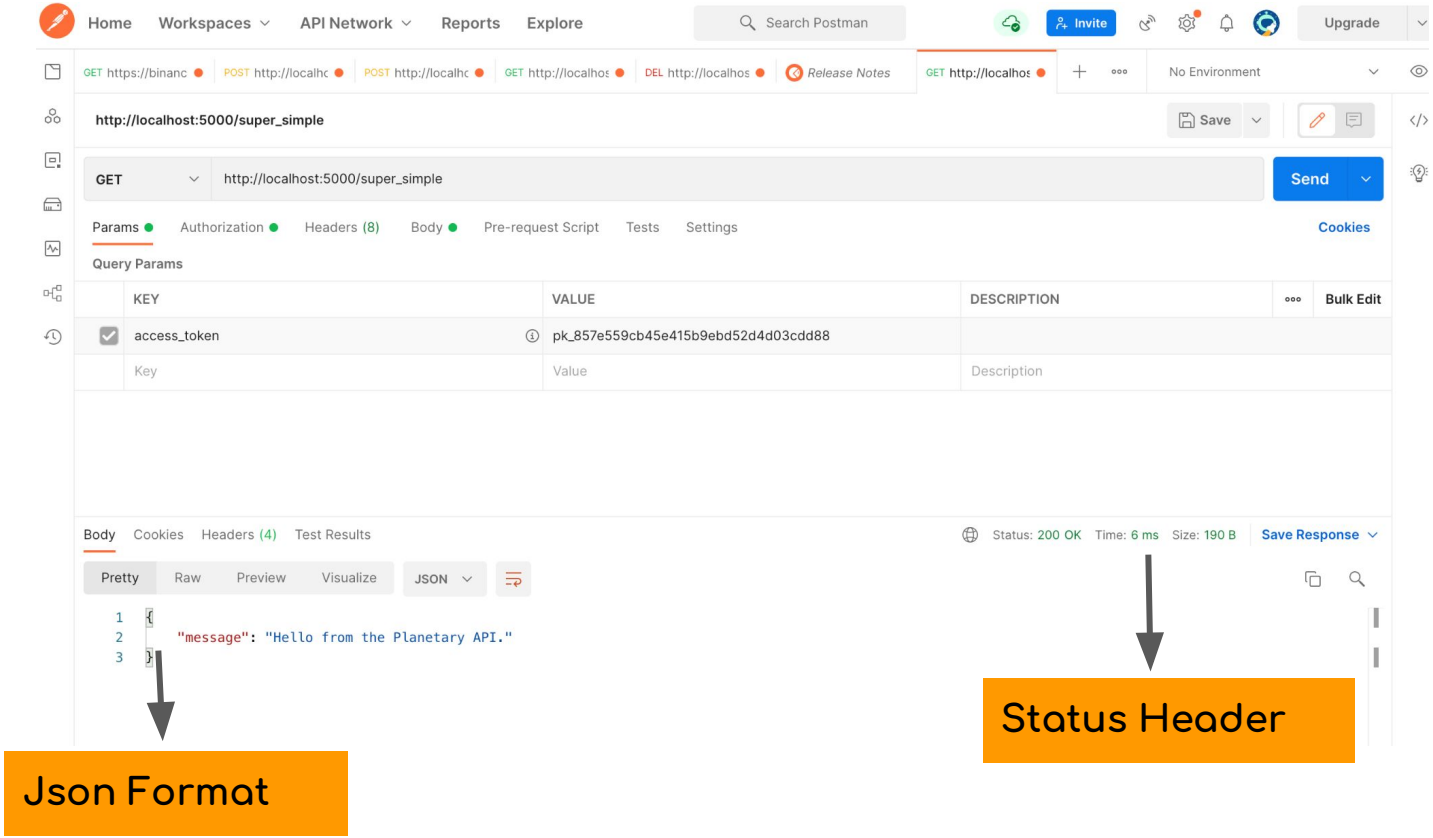
Delete an Existing Singleton Resource Based on ID

Jsonify and HTTP Status

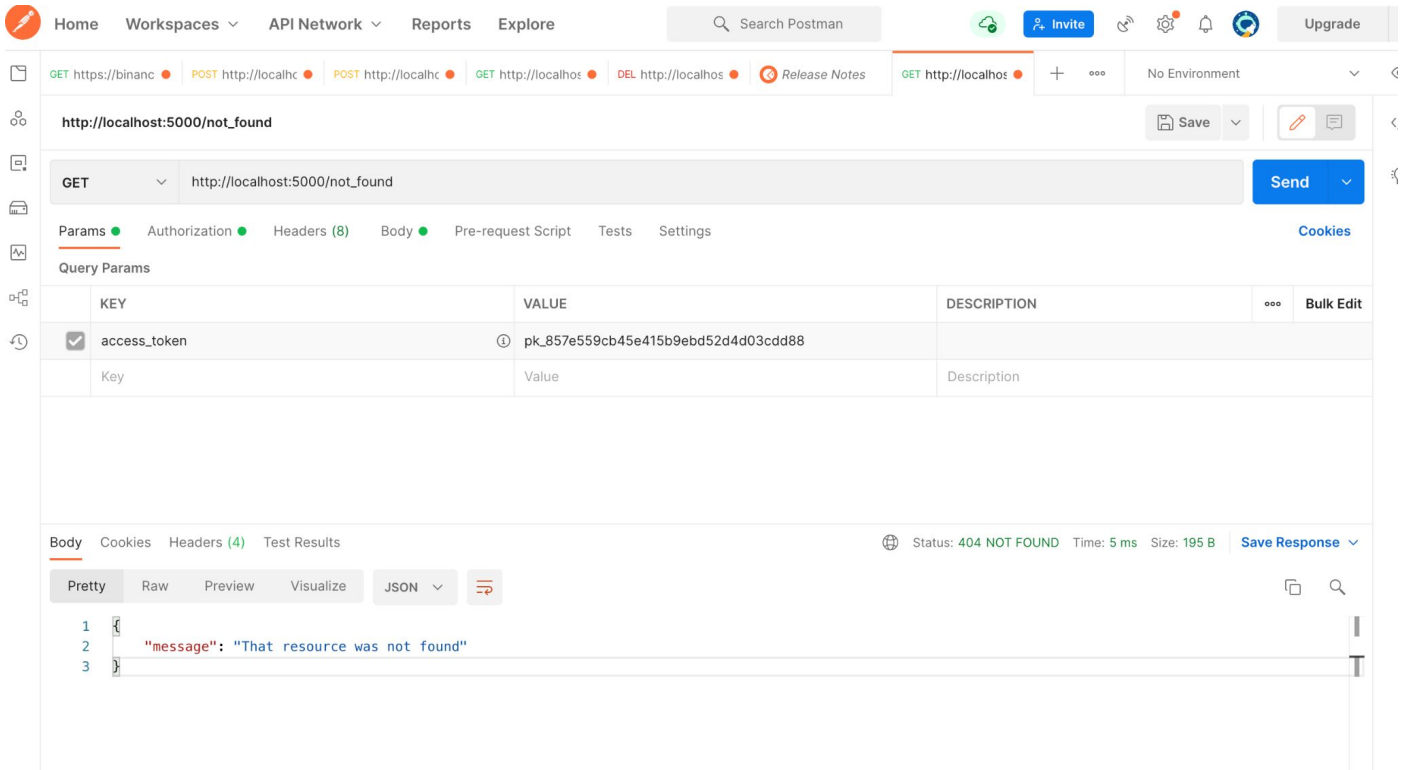
```
@app.route('/super_simple')
def super_simple():
    return jsonify(message='Hello from the Planetary API.'), 200


@app.route('/not_found')
def not_found():
    return jsonify(message='That resource was not found'), 404
```

JSON data with 200 Status Demo



JSON data with 404 Status Demo



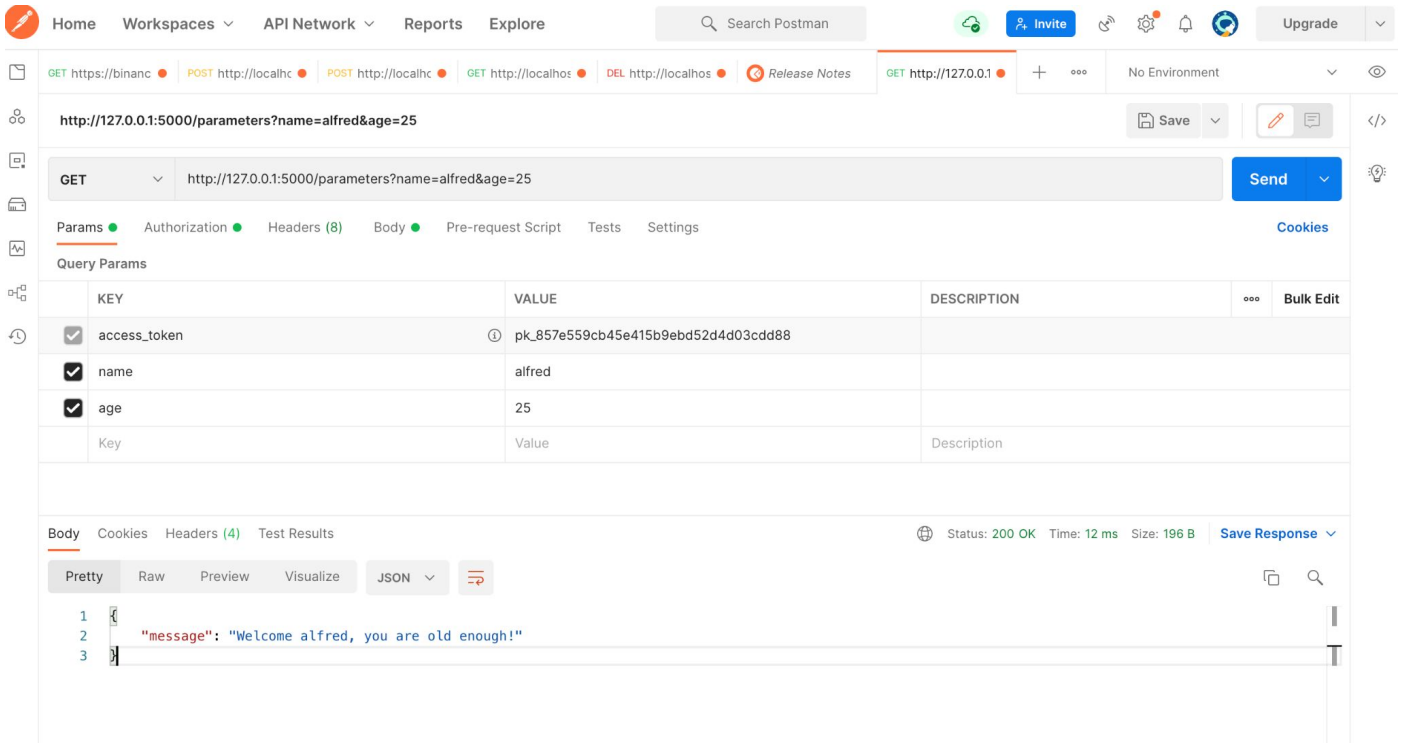
Endpoint with URL Parameter

- URL parameters are the portion of a URL that follows a question mark.
- They are comprised of a key and a value pair, separated by an equal sign.
- Multiple parameters can be added to a single page by using an ampersand

Endpoint with URL Parameter Demo

```
@app.route('/parameters')
def parameters():
    name = request.args.get('name')
    age = int(request.args.get('age'))
    if age < 18:
        return jsonify(message="Sorry " + name + ", you are not
old enough."), 401
    else:
        return jsonify(message="Welcome " + name + ", you are
old enough!")
```

Test Web API with URL Parameter



Activity: URL parameter

Create a Web API with URL parameter for blog post page with id such as

http://127.0.0.1:5000/blog/post?post_id=5

return
Post ID = 5

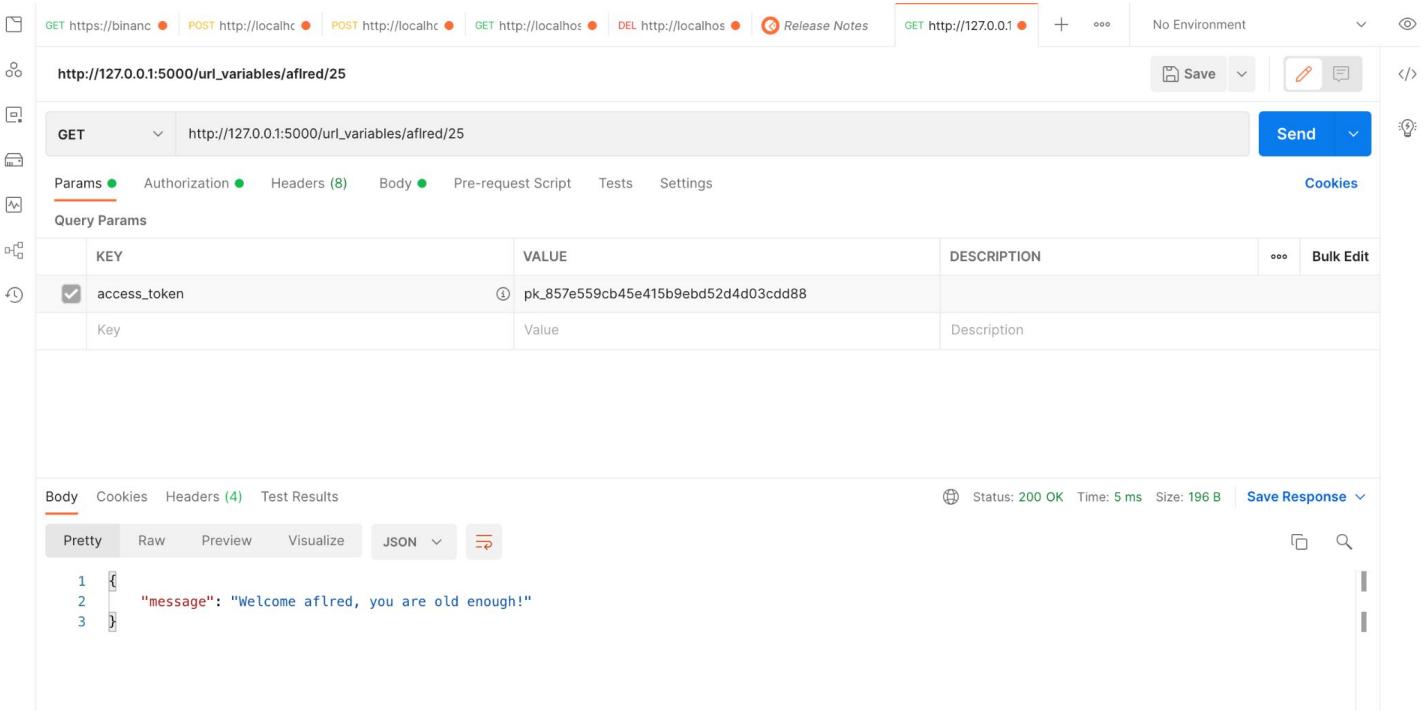
Endpoint Variable Rules

- You can add variable sections to a URL by marking sections with <variable_name>.
- Your function then receives the <variable_name> as a keyword argument.

Endpoint Variable Rule Demo

```
app.route('/url_variables/<string:name>/<int:age>')
def url_variables(name: str, age: int):
    if age < 18:
        return jsonify(message="Sorry " + name + ", you are not old enough."),
401
    else:
        return jsonify(message="Welcome " + name + ", you are old enough!"
```

Test Endpoint Variable Rule



Activity: Endpoint Variable Rule

Create a Web API endpoint with variable for blog post page with id such as

http://127.0.0.1:5000/blog/post/25

return
Post ID = 25

Topic 4
Deployment

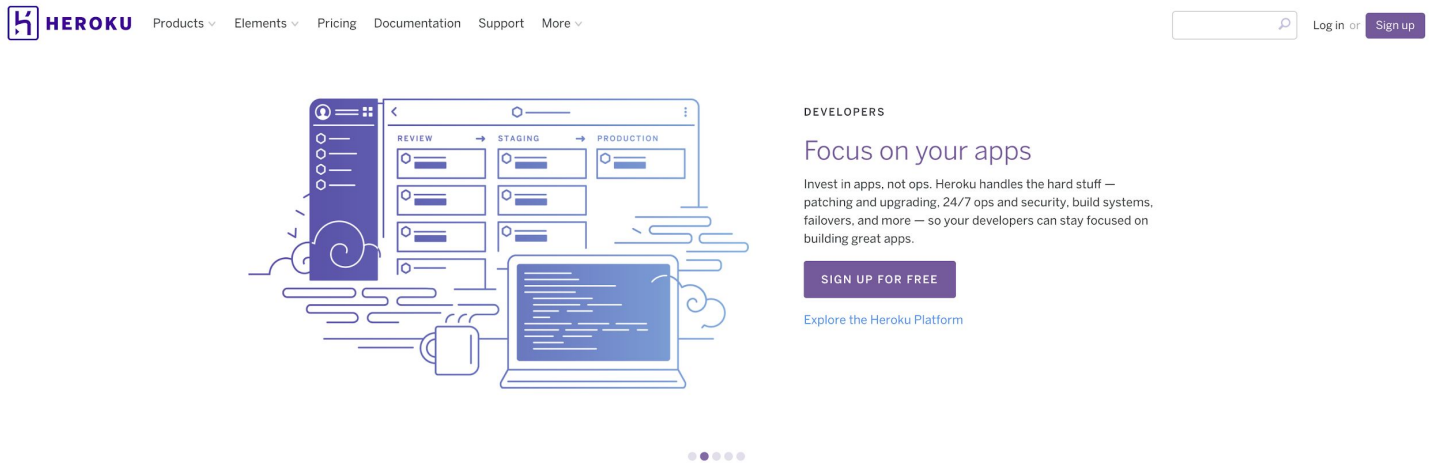
Deploying Flask App

- To switch over from a development environment to a full-fledged production environment, an application needs to be deployed on a real web server.
- For small application, the following hosted platforms, all of which offer free plan for small application.
  - Heroku
  - dotcloud
  - webfaction

Sign Up Heroku

Sign up a free account with Heroku

https://www.heroku.com/



Step 1: Install Heroku CLI

Download Heroku CLI (Command Line Interface) from the link below

https://devcenter.heroku.com/articles/heroku-cli#download-and-install

Step 2 : Install VirtualEnv

- pip3 install virtualenv
- virtualenv [flaskapp]
- source [flaskapp]/bin/activate

Step 3 : Install Flask, Gunicorn

- pip3 install flask
- pip3 install gunicorn

Step 4 : Create Folder

- mkdir [app]
- cd [app]
- Add your files

Step 5: Add Requirements

- pip freeze > requirements.txt
- nano Procfile
  web: gunicorn [filename]:app

filename without square brackets and .py extension

Step 6: Setup Git Repository

- Download and Install git https://git-scm.com/downloads
- git init
- git add -A
- git commit -m "init"

Step 7: Create Heroku App

- heroku create

Step 8: View Heroku App

- git push heroku master
- heroku open

Summary
Q&A

Feedback

Kindly submit your course feedback below
https://forms.gle/EYSNdkrecftKfwb27

Thank You!