Backend Web App Development

MSBA AN6100 Programming Essentials

Coordinator

- Mr Koh Choon Chye
- cckoh@ntu.edu.sg

Course Content

- Client-Server Model
- Building REST API
- Flask

This seminar we need to use the spyder in anaconda

Web application methodology

- **Waterfall**
- **Agile**
- **Scrum**
- **Extreme Programming**
- **Lean**

Many more ….

With a variety of methodologies for web development applicable to a different set of projects, developers have lots of options to build software that works flawlessly. Every approach has its own peculiarities and there is no silver bullet as the development process of each particular project might depend on several factors

https://scand.com/company/blog/methodology-for-web-development/

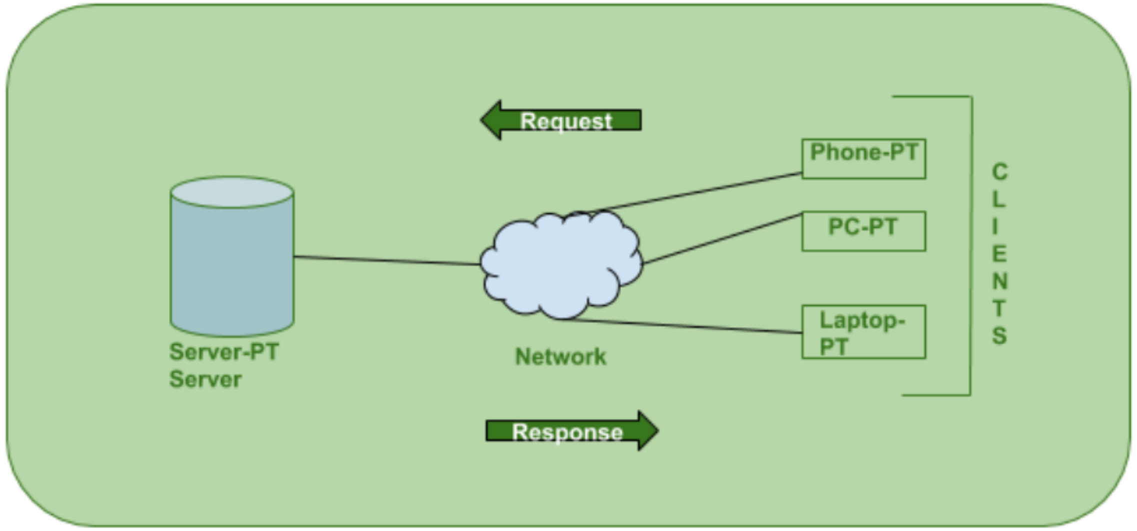
Client-Server Model

- The Client-server model is a distributed application structure that partitions task or workload between the providers of a resource or service, called servers, and service requesters called clients. In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and deliver the data packets requested back to the client. Clients do not share any of their resources. Examples of Client-Server Model are Email, World Wide Web, etc.

https://www.geeksforgeeks.org/client-server-model/

How the Client-Server Model works ?

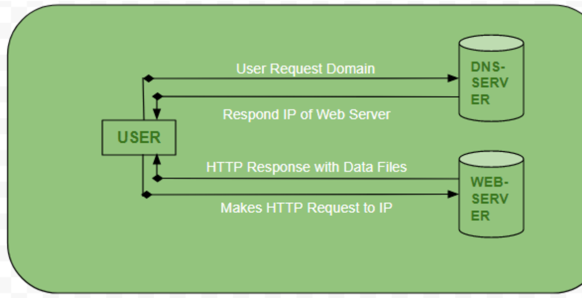
- **Client:** A computer receiving information or using a particular service from the service providers (**Servers**).
- **Servers:** A remote computer which provides information (data) or access to particular services.



https://www.geeksforgeeks.org/client-server-model/

How the browser interacts with the servers ?

- User enters the **URL**(Uniform Resource Locator) of the website or file. The Browser then requests the **DNS**(DOMAIN NAME SYSTEM) Server.
- **DNS Server** lookup for the address of the **WEB Server**.
- **DNS Server** responds with the **IP address** of the **WEB Server**.
- Browser sends over an **HTTP/HTTPS** request to **WEB Server's IP** (provided by **DNS server**).
- Server sends over the necessary files of the website.
- Browser then renders the files and the website is displayed. This rendering is done with the help of **DOM** (Document Object Model) interpreter, **CSS** interpreter and **JS Engine** collectively known as the **JIT** or (Just in Time) Compilers.



https://www.geeksforgeeks.org/client-server-model/

Web Framework

- an architecture containing tools, libraries, and functionalities suitable to build and maintain massive web projects using a fast and efficient approach.
- designed to streamline programs and promote code reuse
- In python, Django and Flask
- Python Flask Framework is a lightweight micro-framework based on Werkzeug, Jinja2. Its core functionality is small yet typically extensible to cover an array of small and large applications
- Although Django is considered Full-stack, as more and more developers focusing on microservices and micro-frontends to reduce the loading time of a webpage, Flask seems to be more welcome.

Overview of Python Flask Framework

- Web apps are developed to generate content based on retrieved data that changes based on a user's interaction with the site. The server is responsible for querying, retrieving, and updating data. This makes web applications to be slower and more complicated to deploy than static websites for simple applications.
- There are two primary coding environments for the whole web app ecosystem.
  - **Client-side Scripting**
    - The code executed on the user's browser visible to anyone who has access to the system, generating the first results.
  - **Server-side Scripting**
    - This type of code is run on the backend on a web server. To enable developers to design, build, maintain, host web apps over the internet, a web framework necessary.

Why use Flask for rest API?

- Flask is able to achieve generally faster performance than Django, generally due to the fact that Flask is much more minimal in design.
- Flask also has NoSQL support.
- Flask also has Flask-RESTful, an extension for Flask that provides additional support for building REST APIs.
- Flask-Restful is a lightweight abstraction that works with the existing Object-relation mapping (ORM)/libraries.
- Flask-RESTful encourages best practices with minimal setup. And also take less time to master.

Initialization

- flask applications must create an application instance. The web server passes all the requests it receives from clients to objects for handling using a protocol for WSG from flask import Flask

```
from flask import Flask

app = Flask(__name__)

if __name__ == "__main__":
    app.run(host='localhost', port=5000, debug=False)
```

- app = Flask (__name__) (An application instance is an object of class Flask.)

lect_demo1.pyc

Server Startup

- The application instance has a ‘run’ method that launches flask’s integrated development webserver –

```
from flask import Flask

app = Flask(__name__)

if __name__ == "__main__":
    app.run(host='localhost', port=5000, debug=False)
```

- Once the script starts, it waits for requests and services in a loop.
- Local-Host – Run a python script in a virtual environment. Flask starts the server listening on 127.0.0.1 and port 5000 by default. To accept connection from any remote address, use host = ‘0.0.0.0.’
- Note that some machine may already use port 5000 for other apps, please try port 5001 or 8000 or 8080

lect_demo1.py

Routes and View Functions in Flask Framework Instance

- Clients send requests to the webserver, in turn, sends them to the Flask application instance.
- The instance needs to know what code needs to run for each URL requested and map URLs to Python functions.
- The association between a URL and the function that handles it is called a route. The most convenient way to define a route in a Flask application is through the (app.route).
- Decorator exposed by the application instance, which registers the 'decorated function,' decorators are python feature that modifies the behavior of a function.

```
@app.route('/')
def index():
   return "I am now in web development!"
```

```
@app.route('/<name>', methods=['GET'])
def home(name):
            return jsonify({"message": f"Hello! {name}"})
```

- The index is a view function, and the response can even be a string format HTML.

lect_demo2.py

Routes and route variables in Flask

- Web applications typically use components of a route as variables that are passed to the route function. Flask lets you do this by way of a special route definition syntax.
- In the example belong, where we have a route in the format /lang/ followed by a name, the name is extracted and passed along to the function as the variable username.

```
@app.route("/lang/<name>")
def greet(name):
        return f"You are learning {name}"
```

- Visit this route with /lang/Python, and you'll see "You are learning Python" in the browser.

[Get started with Flask 2.0 - ProQuest (ntu.edu.sg)]

Route methods in Flask

- Route decorators can also specify the methods used to access the route. You can create multiple functions to handle a single route with different methods, like this:

```
@app.route('/post', methods=['GET'])
def post_message_route_get():
    return show_post_message_form()

@app.route('/post', methods=['POST'])
def post_message_route_post():
    return post_message_to_site()
```

[Get started with Flask 2.0 - ProQuest (ntu.edu.sg)]

Alternative to **Route methods in Flask**

we need to import the global request object to access the method property

```
from flask import request


@app.route('/post', methods=['GET', 'POST'])
def post_message_route():
    if request.method == 'POST':
        return post_message_to_site()
    else:
        return show_post_message_form()
```

```
#short cuts, either way will do
@app.get('/post')
def post_message_route_get():
    return show_post_message_form()

@app.post('/post')
def post_message_route_post():
    return post_message_to_site()
```

Get started with Flask 2.0 - ProQuest (ntu.edu.sg)

Error handlers in Flask

- To create a route that handles a particular class of server error, use the errorhandler decorator:

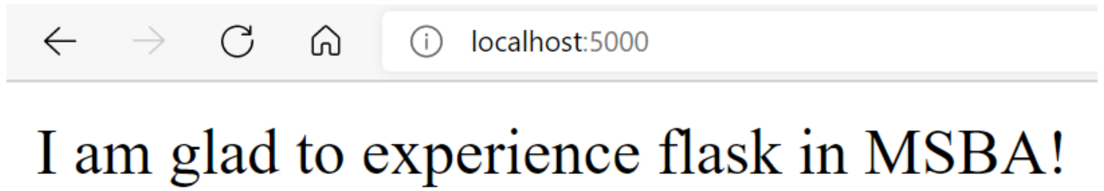
```
@app.errorhandler(404)
def page_not_found(error):
    return f"error: {error}“
```

- For this app, whenever a 404 error is generated, the result returned to the client will be generated by the page_not_found function. error is the exception generated by the application, so you can extract more details from it if needed and pass them back to the client.

Get started with Flask 2.0 - ProQuest (ntu.edu.sg)

Workshop1:

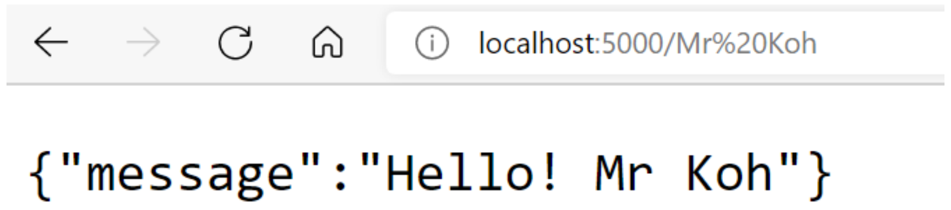
- Prepare a flask application that will start server at port 5000
- Able to view a message at local:5000:


localhost:5000

I am glad to experience flask in MSBA!

- Able to view the greeting when enter

  Localhost:5000/Mr Koh


localhost:5000/Mr%20Koh

```
{"message":"Hello! Mr Koh"}
```

Workshop 1 recommended solution

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

@app.route('/')
def index():
   return "I am glad to experience flask in MSBA!"

@app.route('/<name>', methods=['GET'])
def home(name):
		return jsonify({"message": f"Hello! {name}"})


if __name__ == "__main__":
   app.run(host='localhost', port=5000, debug=False)
```

Workshop1_route.py

Server-side Programming :

1) Querying the database
2) Operations over databases
3) Access/Write a file on server.
4) Interact with other servers.
5) Structure web applications.
6) Process user input. For example if user input is a text in search box, run a search algorithm on data stored on server and send the results.

The Programming languages for server-side programming are :
1) PHP
2) C++
3) Java and JSP
4) Python
5) Ruby on Rails
And many more and more to come …

https://www.geeksforgeeks.org/client-server-model/

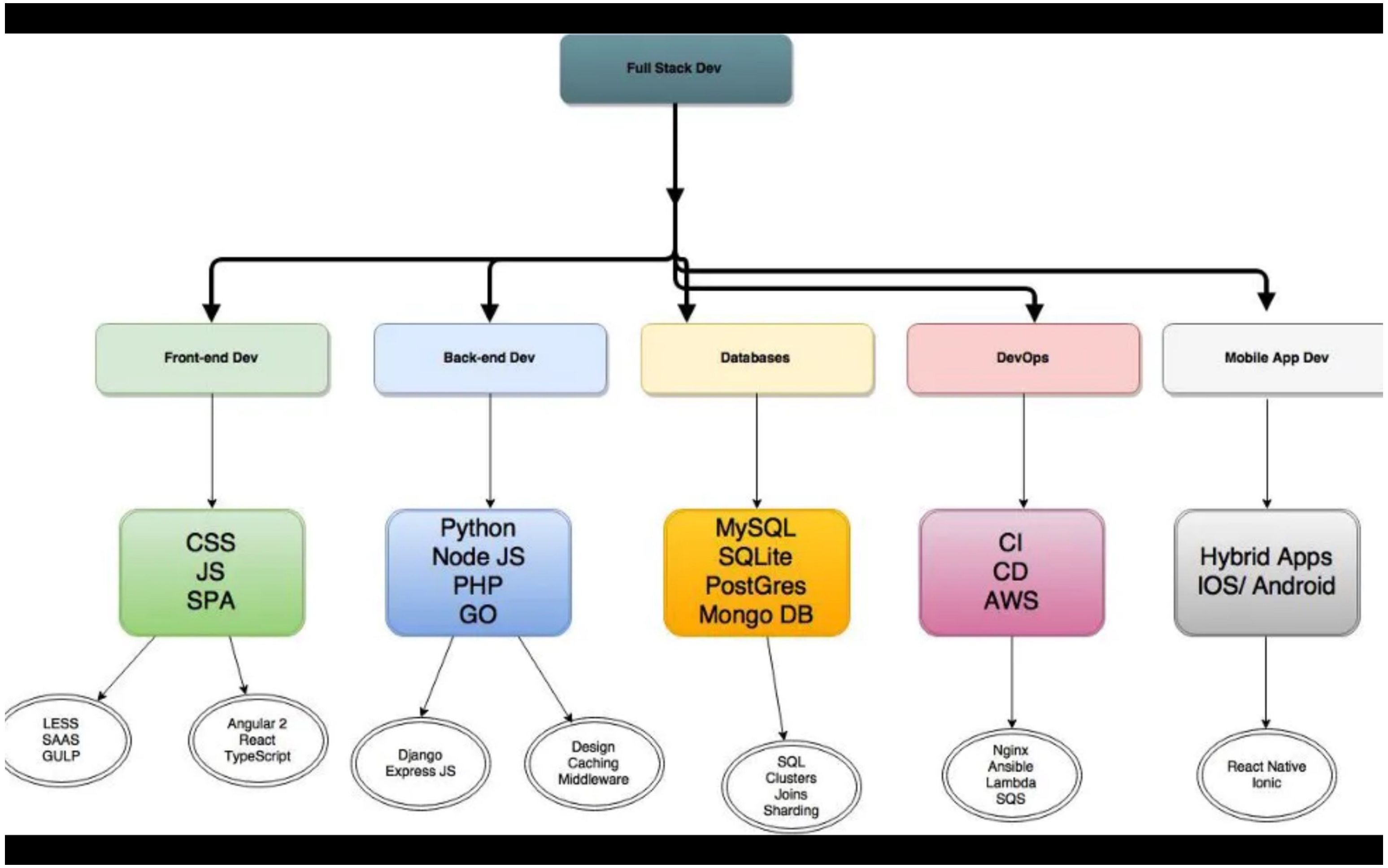
Client-side Programming :

- 1) Interact with temporary storage
  2) Make interactive web pages
  3) Interact with local storage
  4) Sending request for data to server
  5) Send request to server
  6) work as an interface between server and user

The Programming languages for client-side programming are :
1) Javascript
2) VBScript
3) HTML
4) CSS
5) AJAX
And many more and more to come …

https://www.geeksforgeeks.org/client-server-model/

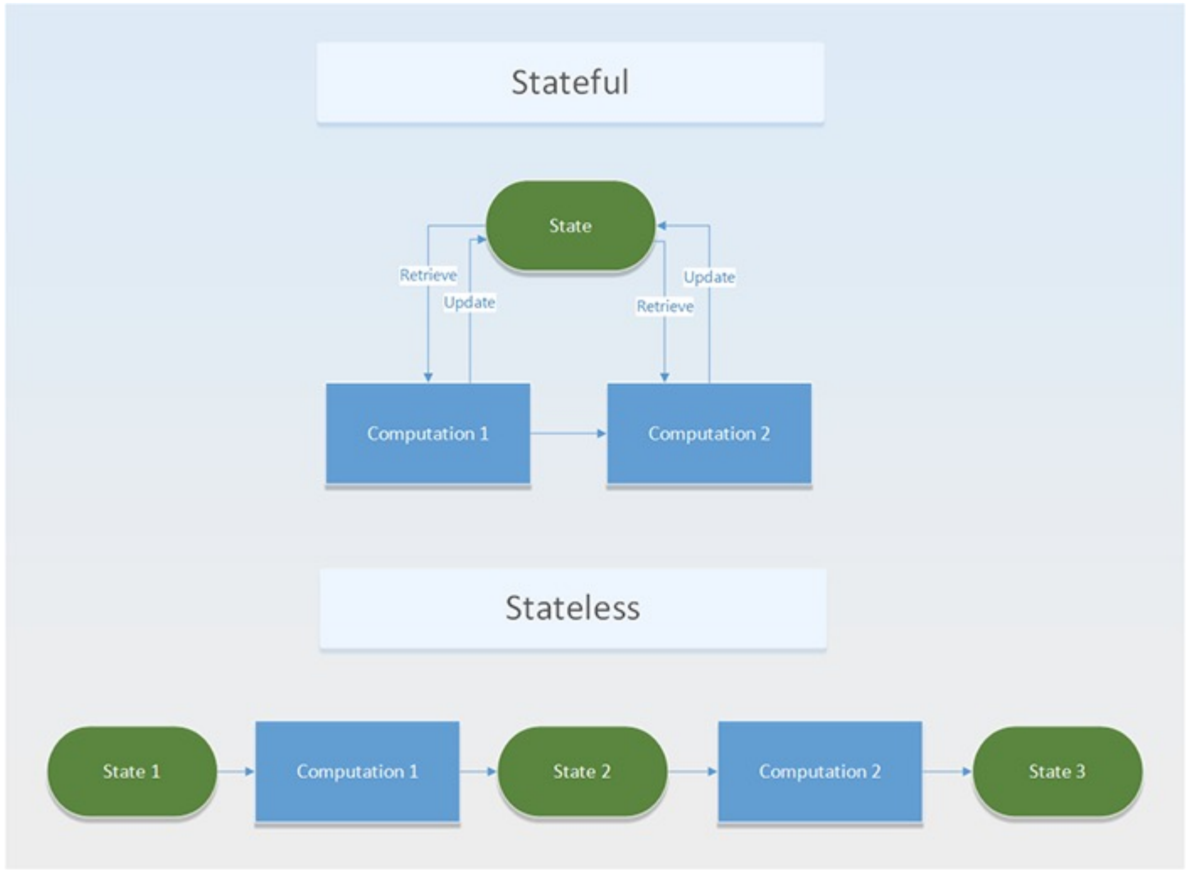
Full Stack Web Development



Picture Credit : https://hackernoon.com/6-essential-tips-on-how-to-become-a-full-stack-developer-1d10965aaead

Stateless Over Stateful Applications

- A **Stateful** application saves data about each client session and uses that data the next time the client makes a request.
- A **Stateless** app is an application program that does not save client data generated in one session for use in the next session with that client.



https://medium.com/@rachna3singhal/stateless-over-stateful-applications-73cbe025f07

Why Stateless Applications?

- A stateful application created a connection with the persistent storage to read and store the data. How long should that connection be open?
- When dealing with sessions a lot of transactions happen at every request which needs to be maintained
- How do we ensure that the connection is there or if it is disconnected?
- How to maintain user's data in cases of a network or storage failure?
- In a Stateful application you are essentially saying that the client is dumb to store any information.

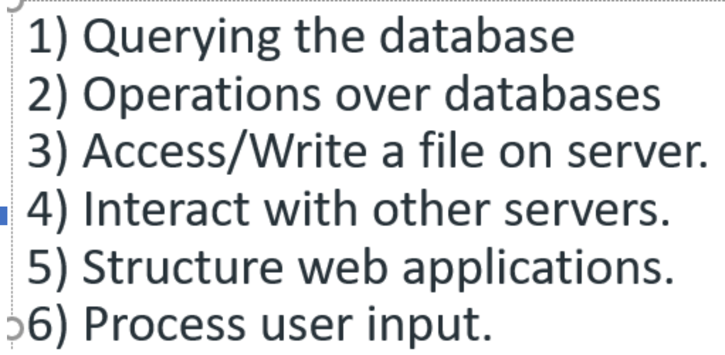
https://medium.com/@rachna3singhal/stateless-over-stateful-applications-73cbe025f07

Advantages of Statelessness

- You can simply scale a stateless application by deploying it on multiple servers. Scales beautifully horizontally
- Stateless apps can be cached easily and hence can be faster
- Less storage
- No need to bind the client to the server as in each request the client provides all its information necessary for the server to act upon it

Most software which are Micro-services are already Stateless applications using REST API design. Statelessness of application is the foundation on which most web and other concepts such as RESTful design are reliant on. A good understanding and advantage of Stateless over Stateful is very important in providing to the ever so demanding needs of the users nowadays.

https://medium.com/@rachna3singhal/stateless-over-stateful-applications-73cbe025f07



1) Querying the database
2) Operations over databases
3) Access/Write a file on server.
4) Interact with other servers.
5) Structure web applications.
6) Process user input.

Recall API

- **API** (Application Program Interface) is an interface that allows communication between multiple intermediaries meaning that one can access any type of data using any technology. For instance, you can access an API using Javascript which could be built using Python.
- Application Programming Interface (API) is simply an *interface,* used to help two different systems to interact with one another. There are two key ideas behind this:
    1. Don’t expose your application implementation;
    2. Easier to access your application’s data.

API - Application Programming Interface

- This interface allows people to further build upon another application's functionality and data. One might understand them as building blocks you can use to make almost anything as they can be found in everything from Spotify to Yahoo Finance.
- The API frameworks allow developers to perform tasks that aren't all that different from everyday events. For instance, think of giving an order to a server, that server putting your order in, and then bringing back the order when it's ready. This step-by-step process returns the desired outcome: a tasty meal (in this case). A web-based example might be someone signing up to a new e-Commerce site by using their Facebook account.
- Essentially, APIs help sites to communicate on the web and understand information (regardless of programming languages) in order to facilitate processes. HTTP protocol requests allow for sending data and receiving data. The only caveat is that each API requires continuous testing to ensure consistent performance.

https://www.parasoft.com/blog/web-api-vs-web-services-microservices-basics-differences/

Types of Web API

- Composite APIs. These merge service and data APIs. The series of tasks operate synchronously due to execution and NOT due to task requests. These APIs can expedite the execution process, as well as improve web interface listening performance.

- Partner APIs. These require licenses or special rights for access as they are not generally available to public developers.

- Open APIs. In contrast, Open or “Public” APIs bear no access restrictions and can be accessed by the public.

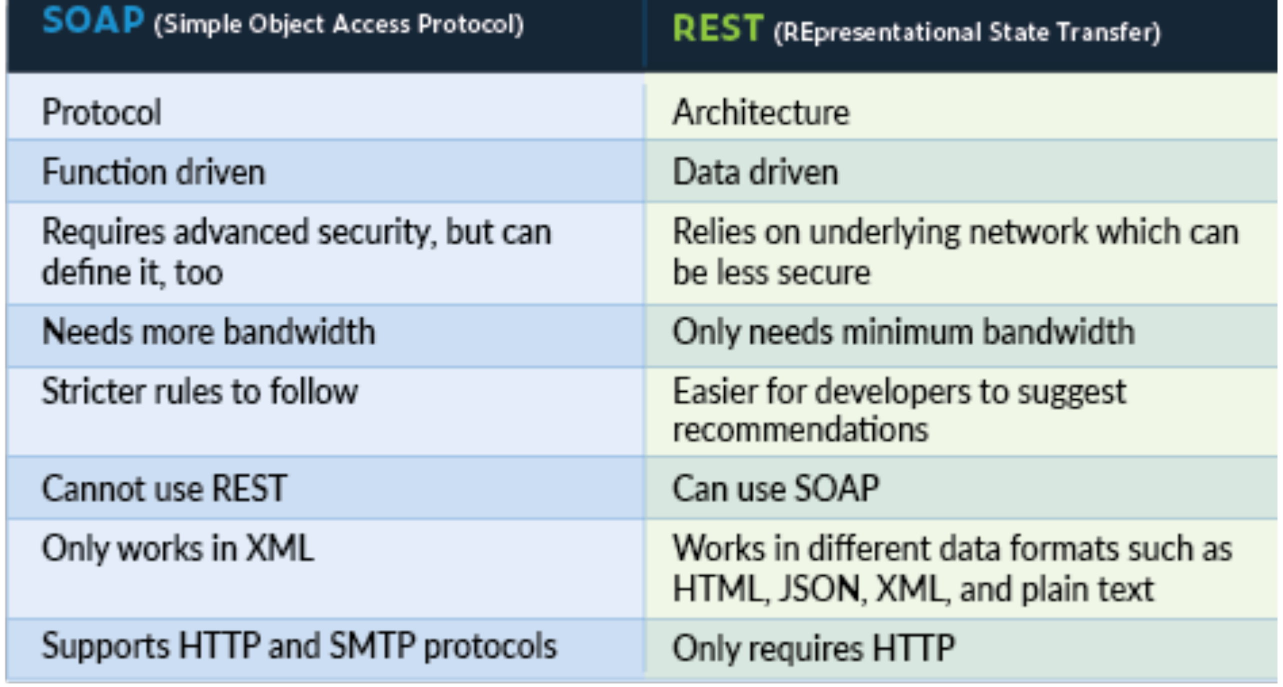
- Internal APIs. As the name suggests, these operate as “Private” APIs within internal systems. They can be used among internal teams in a single company for service or product improvement.

Some APIs also require keys for authentication before allowing the mixture of information.

Web Service

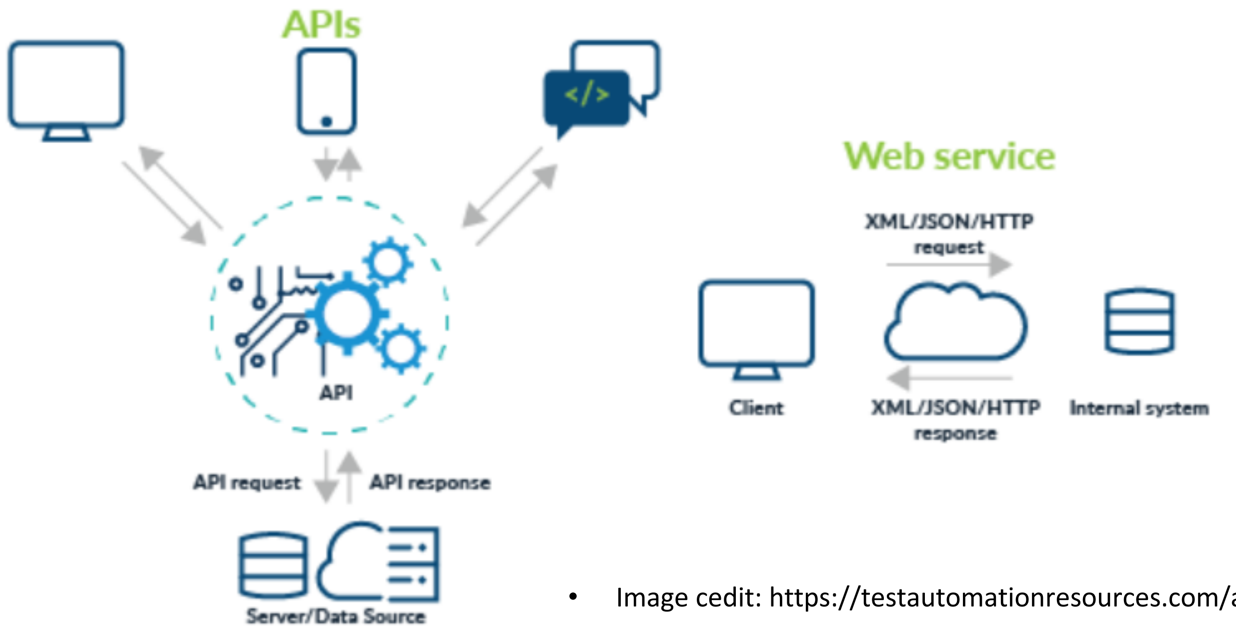
- A web service, in contrast to an API, functions more like a resource that's available using the internet. The network-based resource can be applied to specific tasks, but they require a network to function. This means that all web services are APIs, but only some APIs are web services.

- A web service works by supporting interoperable machine-to-machine communication using a network. As such, web services tend to be connected with SOA or Service Oriented Architecture. This allows for different features to be separated then made available as various services within a network.



SOAP (Simple Object Access Protocol)	REST (REpresentational State Transfer)
Protocol	Architecture
Function driven	Data driven
Requires advanced security, but can define it, too	Relies on underlying network which can be less secure
Needs more bandwidth	Only needs minimum bandwidth
Stricter rules to follow	Easier for developers to suggest recommendations
Cannot use REST	Can use SOAP
Only works in XML	Works in different data formats such as HTML, JSON, XML, and plain text
Supports HTTP and SMTP protocols	Only requires HTTP

Table credit : https://www.parasoft.com/blog/web-api-vs-web-services-microservices-basics-differences/



APIs vs Web Services

- Image cedit: https://testautomationresources.com/api-testing/differences-web-services-api/

Microservice

- Microservices are architectural styles typically used in modern web apps that require more fragmented functionality. That means that each service is a modular, unique process that can be deployed independently. The lightweight architecture still makes use of SOA and can be especially advantageous for larger companies.
- Separate teams can work on various items without encountering difficulties. But this necessitates communication among the different parts which is where APIs come in. However, web services and microservices are not quite the same either.
- It's best to consider a microservice as an autonomous application designed for a single, specific service as part of a larger application architecture. In contrast, a web service acts as a strategy to facilitate service availability across applications by using a web interface.

Microservices, APIs, and web services can all be used separately or in tandem to help your business. The choice between them is likely to depend on the specific protocols, messaging formats, or communication styles you need to support.

https://www.parasoft.com/blog/web-api-vs-web-services-microservices-basics-differences/

Web Reference:

- https://blog.dreamfactory.com/restful-api-and-microservices-the-differences-and-how-they-work-together/

Business Case for APIs

- Integrate with third party APIs
- Build APIs for internal use
- Build APIs and expose APIs for external use

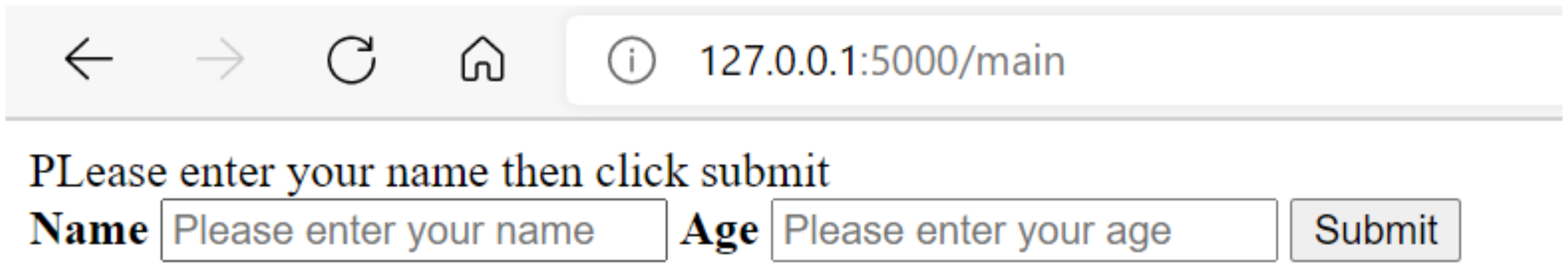
What is an API Endpoint?

- Simply put, an endpoint is one end of a communication channel. When an API interacts with another system, the touchpoints of this communication are considered endpoints. For APIs, an endpoint can include a URL of a server or service. Each endpoint is the location from which APIs can access the resources they need to carry out their function.

- APIs work using ‘requests’ and ‘responses.’ When an API requests information from a web application or web server, it will receive a response. The place that APIs send requests and where the resource lives, is called an endpoint.

Workshop 2

- Prepare a endpoint /main so that upon called, appear in the browser as follows:



127.0.0.1:5000/main

PLease enter your name then click submit

Name Please enter your name **Age** Please enter your age Submit

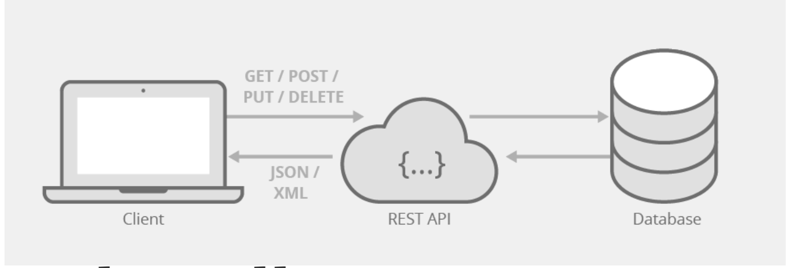
copy & past from main.txt for the html codes

Workshop 2 recommended solution

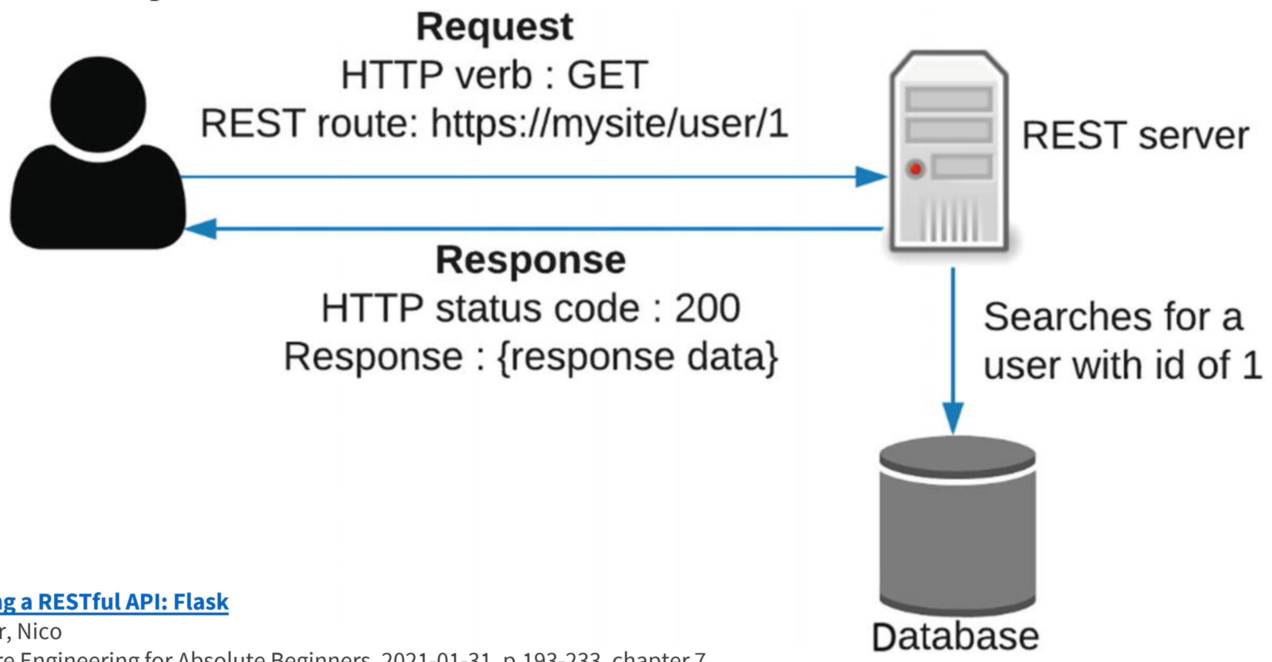
```
@app.route('/main')
def main():
   return """
<html>
<div id="rightdiv">
   PLease enter your name then click submit
   <form >
         <label for="name">
           <strong>Name</strong>
         </label>
         <input type="text" id="name" placeholder="Please enter your name"
name="name" required>
         <label for="age">
           <strong>Age</strong>
         </label>
         <input type="text" id="age" placeholder="Please enter your age"
name="age" required>
         <input type="submit" value="Submit">
   </form>
</div>
</html>
"""
```

Workshop2_route.py

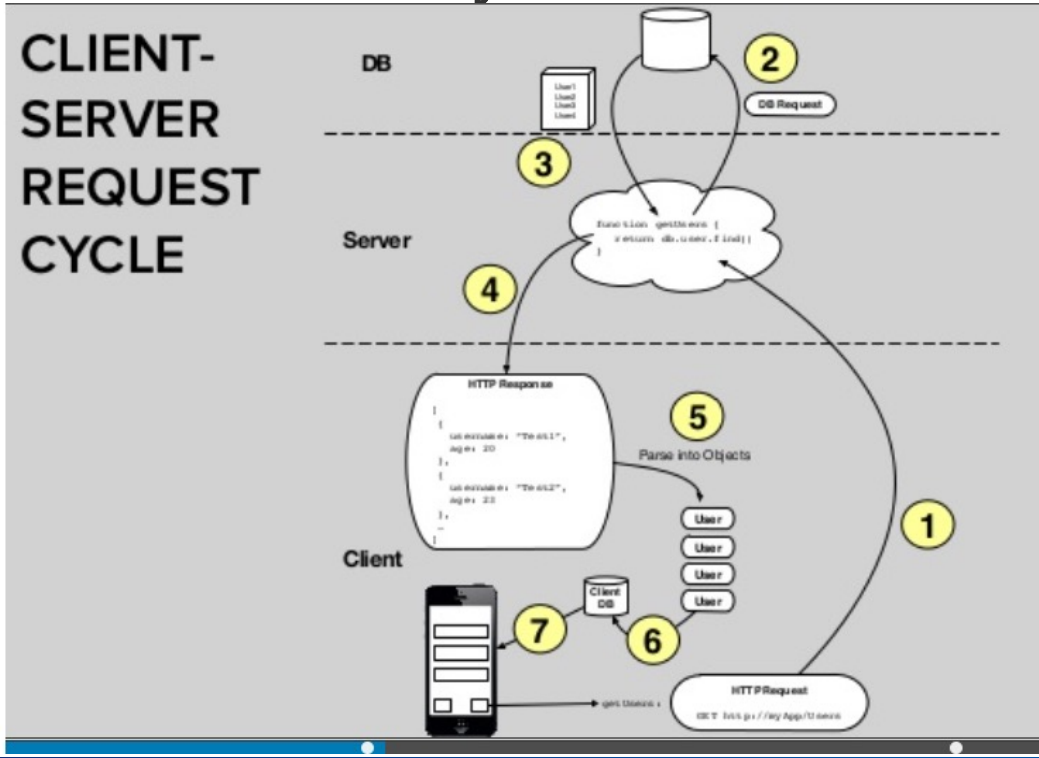
What Is REST?



- REST, or RESTful, is a popular architectural pattern that allows a system to serve data to an external source, like a front end, in a reliable and predictable way. It allows different systems, written in completely different technologies, to communicate with your system



Request
HTTP verb : GET
REST route: https://mysite/user/1
REST server
Response
HTTP status code : 200
Response : {response data}
Searches for a user with id of 1
Database



[Creating a RESTful API: Flask](#)
Loubser, Nico
Software Engineering for Absolute Beginners, 2021-01-31, p.193-233, chapter 7

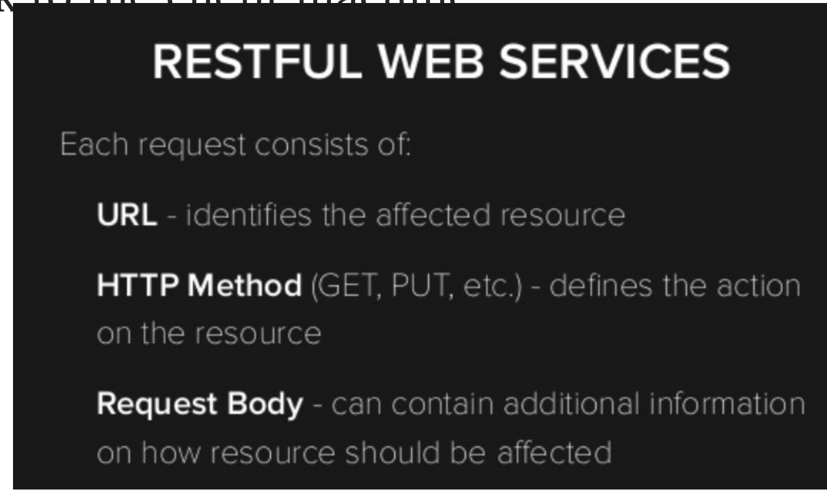
Representational State Transfer (REST)

- REST is an HTTP, or preferably an HTTPS, request very similar a client machine places a request to obtain information from a server, but with very specific rules that allow for dependable and understandable data transfer.

- The server is not only limited to a database but can also be another machine that reads and writes data from a flat file or any other data store.

- The server machine then transfers the current state of the resource back to the client machine.

- It is an architectural style defined by key principles:
  1. **Uniform Interface**
     There must be a standardized way of accessing your API routes.
  2. **Stateless**
     Every client request has nothing to do with previous or subsequent requests.
  3. **Client-Server**
     There must be a client a server and they act individually.
  4. **Cacheable & Layered System**
     Responsible for networking efficiency and consistent behavior regardless of intermediaries.



RESTFUL WEB SERVICES

Each request consists of:

URL - identifies the affected resource

HTTP Method (GET, PUT, etc.) - defines the action on the resource

Request Body - can contain additional information on how resource should be affected

HTTP Requests

- APIs primarily transmit data to the client via HTTP. This protocol has its own methods and elements:
  - **GET:** ONLY retrieves information for the requested resource of the given URL.
  - **POST:** Send data to the server to create a new resource.
  - **PUT:** Replaces all of the representation of the target resource with the requested data.
  - **PATCH:** Partially modifies the representation of the target resource with the requested data.
  - **DELETE:** Removes all of the representations of the resource specified by the URL.
  - **OPTIONS:** Sends the communication options for the requested resource

HTTP Methods in Flask

- **Request**
  - To process incoming data in Flask, you need to use the request object, including mime-type, IP address, and data. HEAD: Un-encrypted data sent to server w/o response.
- **GET**
  - Sends data to the server requesting a response body.
- **POST**
  - Read form inputs and register a user, send HTML data to the server are methods handled by the route.
- Flask attaches methods to each route so that different view functions can handle different request methods to the same URL.

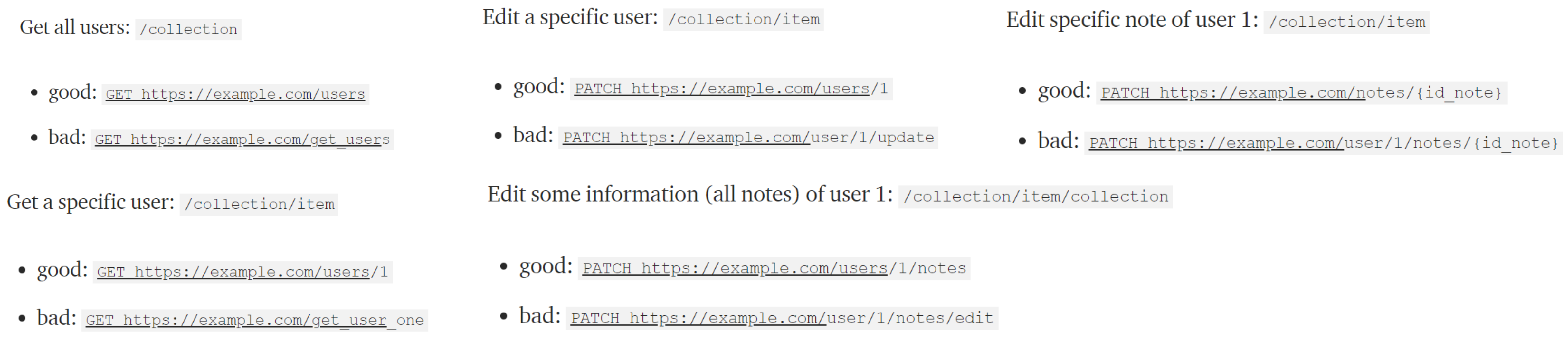
CRUD

Hence, REST APIs are not limited to CRUD functions

- refers to four basic operations performed on database servers
  - **Create**
    When the client sends a POST request to the API, the create function is responsible to handle it.
  - **Read**
    When the client sends a GET request to the API, the read function is responsible to handle it.
  - **Update**
    When the client sends a PUT or PATCH request to the API, the update function is responsible to handle it.
  - **Delete**
    When the client sends a DELETE request to the API, the delete** function is responsible to handle it.

Endpoints Conventions

- There are some principles in how to define your endpoints. In other words:
  1. Should be intuitive
  2. Organized by resource: Use nouns instead of verbs
  3. Consistent scheme: Specify plural nouns for collections
  4. Not too lengthy: collection/item/collection



Get all users: `/collection`

- good: `GET https://example.com/users`
- bad: `GET https://example.com/get_users`

Get a specific user: `/collection/item`

- good: `GET https://example.com/users/1`
- bad: `GET https://example.com/get_user_one`

Edit a specific user: `/collection/item`

- good: `PATCH https://example.com/users/1`
- bad: `PATCH https://example.com/user/1/update`

Edit some information (all notes) of user 1: `/collection/item/collection`

- good: `PATCH https://example.com/users/1/notes`
- bad: `PATCH https://example.com/user/1/notes/edit`

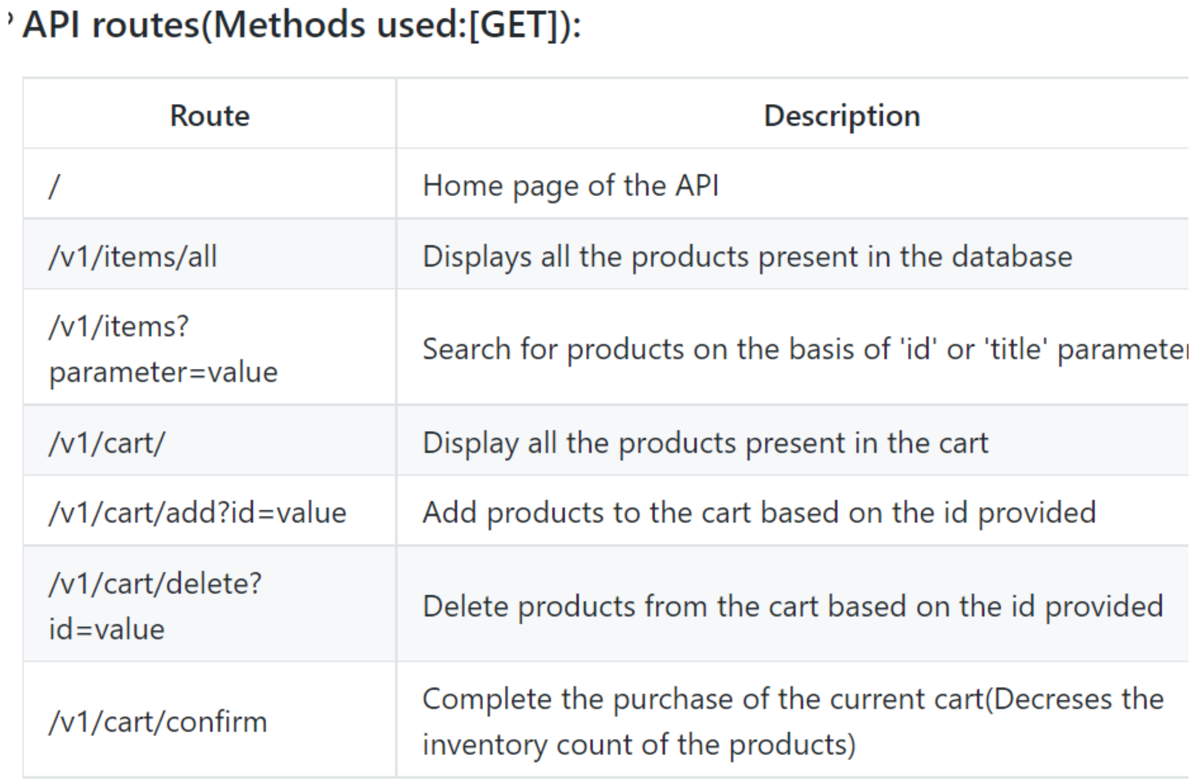
Edit specific note of user 1: `/collection/item`

- good: `PATCH https://example.com/notes/{id_note}`
- bad: `PATCH https://example.com/user/1/notes/{id_note}`

Use of JSON in Client-Server API Interaction

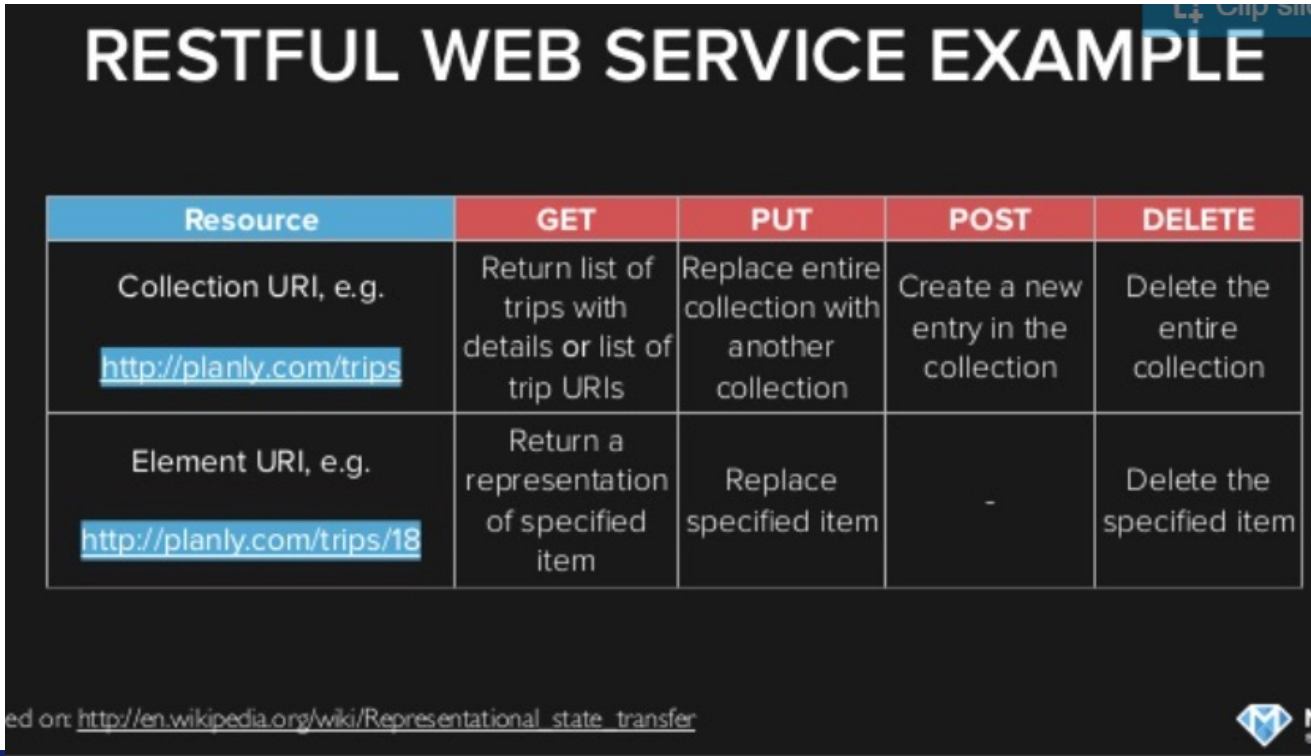
- **APIs use JSON format** for **accepting** and **returning requests**.
- That is, the client sends the request data to the server as a JSON text. Similarly, the server processes the data and returns back the response again as a JSON text.
- Therefore, the whole process of a Web Application based on REST API is as follows:
  - The user sends the request data to the server as JSON.
  - The server first converts the JSON into a python-readable format.
  - Then the server processes the request and creates the response data again as a JSON
  - Then the webpage Template converts the JSON response to a user-readable format and displays it on the webpage.
- Therefore, real exchange of information between the client-side (FRONT-END) and the server (BACK-END) in API happens using **JSON text.**

How APIs are documented (normally)



API routes(Methods used:[GET]):

Route	Description
/	Home page of the API
/v1/items/all	Displays all the products present in the database
/v1/items?parameter=value	Search for products on the basis of 'id' or 'title' paramete
/v1/cart/	Display all the products present in the cart
/v1/cart/add?id=value	Add products to the cart based on the id provided
/v1/cart/delete?id=value	Delete products from the cart based on the id provided
/v1/cart/confirm	Complete the purchase of the current cart(Decreses the inventory count of the products)



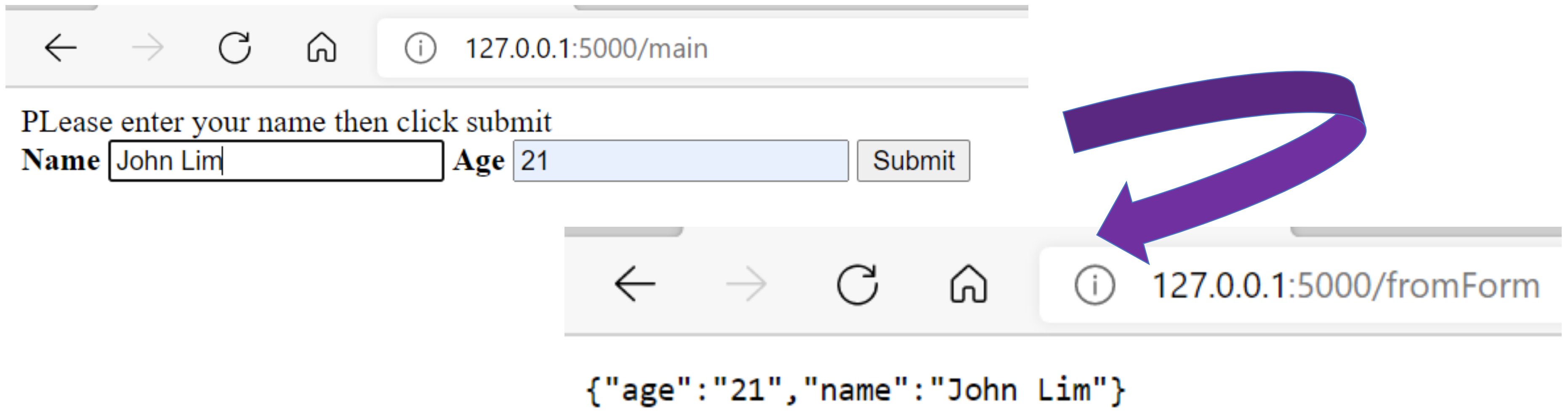
RESTFUL WEB SERVICE EXAMPLE

Resource	GET	PUT	POST	DELETE
Collection URI, e.g. http://planly.com/trips	Return list of trips with details or list of trip URIs	Replace entire collection with another collection	Create a new entry in the collection	Delete the entire collection
Element URI, e.g. http://planly.com/trips/18	Return a representation of specified item	Replace specified item	-	Delete the specified item

ed on: http://en.wikipedia.org/wiki/Representational_state_transfer

Workshop 3

- Create a end point fromForm so that when the user click the submit after entering name and age to the form, the browser will show a dictionary as shown.
- Modify the html codes to allow the POST event


127.0.0.1:5000/main

PLease enter your name then click submit

Name John Lim **Age** 21 Submit

127.0.0.1:5000/fromForm

```
{"age":"21","name":"John Lim"}
```

Workshop 3 recommended solution [1/2]

```
from flask import Flask, request,  jsonify
app = Flask(__name__)

@app.route('/')
def index():
   return "I am glad to experience flask in MSBA!"

@app.route('/main')
def main():
   return """
<html>
<div id="rightdiv">
  PLease enter your name then click submit
   <form action="/fromForm" method="post" >
       <label for="name">
         <strong>Name</strong>
       </label>
       <input type="text" id="name" placeholder="Please enter your name" name="name" required>
       <label for="age">
         <strong>Age</strong>
       </label>
       <input type="text" id="age" placeholder="Please enter your age" name="age" required>
       <input type="submit" value="Submit">
  </form>
</div>
</html>
"""
```

Workshop3_route.py

```
@app.route('/fromForm', methods=['POST'])
def fromForm():
    user = request.form['name'] #data['name']
    age =  request.form['age']  #data['age']
    print("server getting",user, age)
    return {'user' :user, 'age':age}
```

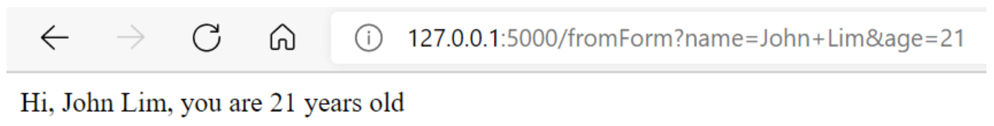
Workshop 3 recommended solution [2/2]

Workshop3_route.py

Workshop 4 : Allow form get

- Change the html codes for the form from method post to method get, re-run the program and observe what happens? Why is it so? How to resolve it?

```
<form action="/fromForm" method=“get“ >
```


127.0.0.1:5000/fromForm?name=John+Lim&age=21

Hi, John Lim, you are 21 years old

Workshop 4 Proposed Solution

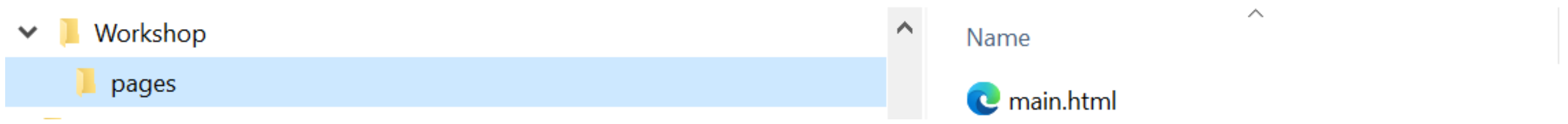
```
@app.route('/fromForm', methods=['POST','GET'])
def get_forminfo():
    print(request.method)
    print(request.data)
    if request.method =='POST':
        user = request.form['name'] #data['name']
        age =  request.form['age']  #data['age']
        #print("server getting",user, age)
        return {'user' :user, 'age':age}
    else:
        return f"Hi, {request.args.get('name')}, you are {request.args.get('age')} years old"
```

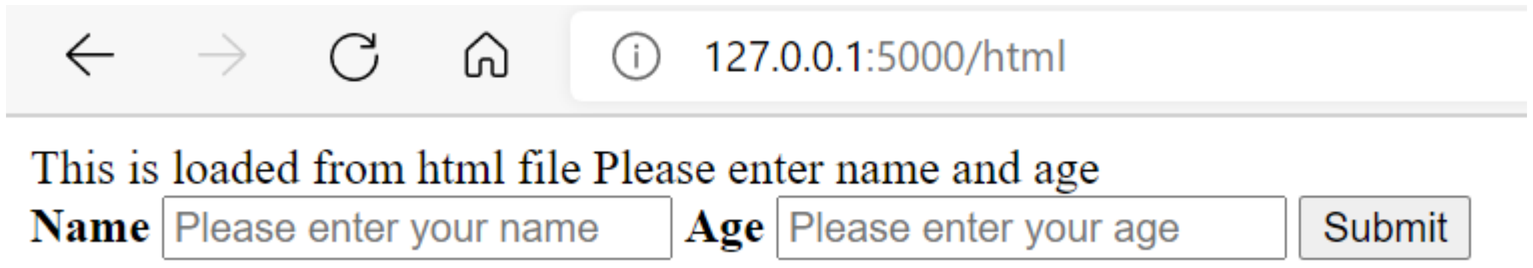
Workshop4_route.py

Workshop 5

Add an end point “html” that will load the same form, but from a prepared html file(stored in pages folder) with the post action.

Try on the html click and observe the response for the form POST action





This is loaded from html file Please enter name and age

Name [Please enter your name] Age [Please enter your age] Submit

```
from flask import Flask, request, render_template, jsonify
app = Flask(__name__, template_folder= "pages")

@app.route('/')
def index():
    return "I am glad to experience flask in MSBA!"

@app.route('/html')
def html():
    return render_template('main.html’)

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

Workshop5route.py

Returning responses in Flask

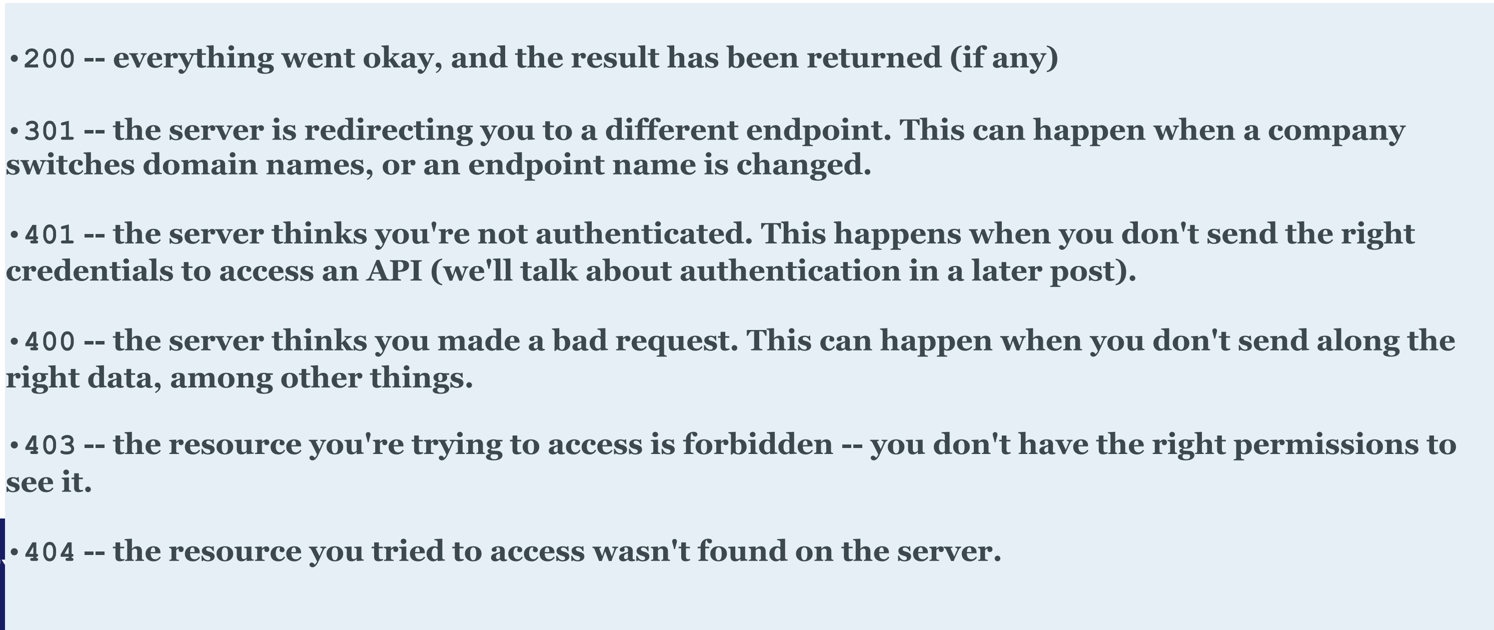
- When a route function returns data, Flask makes a best guess to interpret what has been returned:
  - * Response objects are returned as is. Creating a response object gives you fine-grained control over what you return to the client, but for most use cases you can use one of the items below.
  - * Strings, including the output of Jinja2 templates (more on this next), are converted into Response objects, with a 200 OK status code and a MIME type of text/html.
  - * Dictionaries are converted into JSON.
  - * Tuples can be any of the following:
  - * (response, status code [int])
  - * (response, headers [list/dict])
  - * (response, status code [int], headers [list/dict])
  - Generally, it’s best to return whatever makes clearest the route function’s job. For instance, a 404 error handler can return a 2-tuple — the error message, and the 404 error code. This keeps the route function uncluttered.

[Get started with Flask 2.0 - ProQuest (ntu.edu.sg)]

Response

- Flask invokes a view function. It has to return a response value to the client.

- HTTP requires it to be more than a string response, a status code.
  - Informational – 1xx
  - Successful – 2xx
  - Redirection – 3xx
  - Client Error – 4xx
  - Server Error – 5xx



- `200` -- everything went okay, and the result has been returned (if any)
- `301` -- the server is redirecting you to a different endpoint. This can happen when a company switches domain names, or an endpoint name is changed.
- `401` -- the server thinks you're not authenticated. This happens when you don't send the right credentials to access an API (we'll talk about authentication in a later post).
- `400` -- the server thinks you made a bad request. This can happen when you don't send along the right data, among other things.
- `403` -- the resource you're trying to access is forbidden -- you don't have the right permissions to see it.
- `404` -- the resource you tried to access wasn't found on the server.

Templates in Flask

Jinja2 templates are something of a language unto themselves, but this snippet should give you an idea of how they work. Blocks delineated with {% %} contain template logic, and blocks with {{ }} contain expressions to be inserted at that point. (When we called this template with render_template above, we passed username as a keyword argument; the same would be done for any other variables we'd use.)
Note that Jinja2 templates have constraints on the code that can be run inside them, for security's sake. Therefore, you will want to do as much of the processing as possible for a given page *before* passing it to a template

- Flask includes the Jinja2 template engine to programmatically generate HTML output from data. You use the render_template function to generate HTML, and pass in variables to be used in the template.

```
from flask import render_template
@app.route('/hi/')
def greet(username=None):
            return render_template('hello.html', username=username)
```

- Templates referred to by render_template are by default found in a subdirectory of the Flask project directory, named templates. To that end, the following file would be in templates/hello.htm

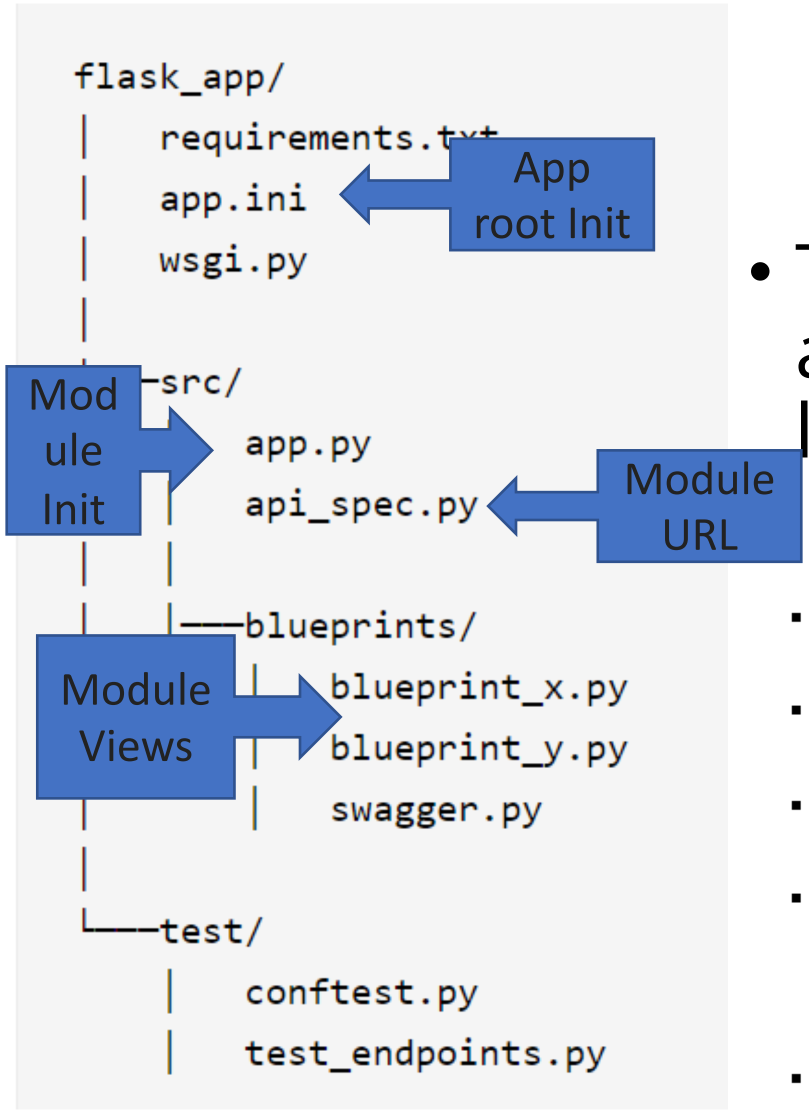
```
Hi there
{% if username %}
Hello {{ username }}!
{% else %}
Hello, whoever you are!
{% endif %}
```

[Get started with Flask 2.0 - ProQuest (ntu.edu.sg)]

The layout of the Python Flask Framework

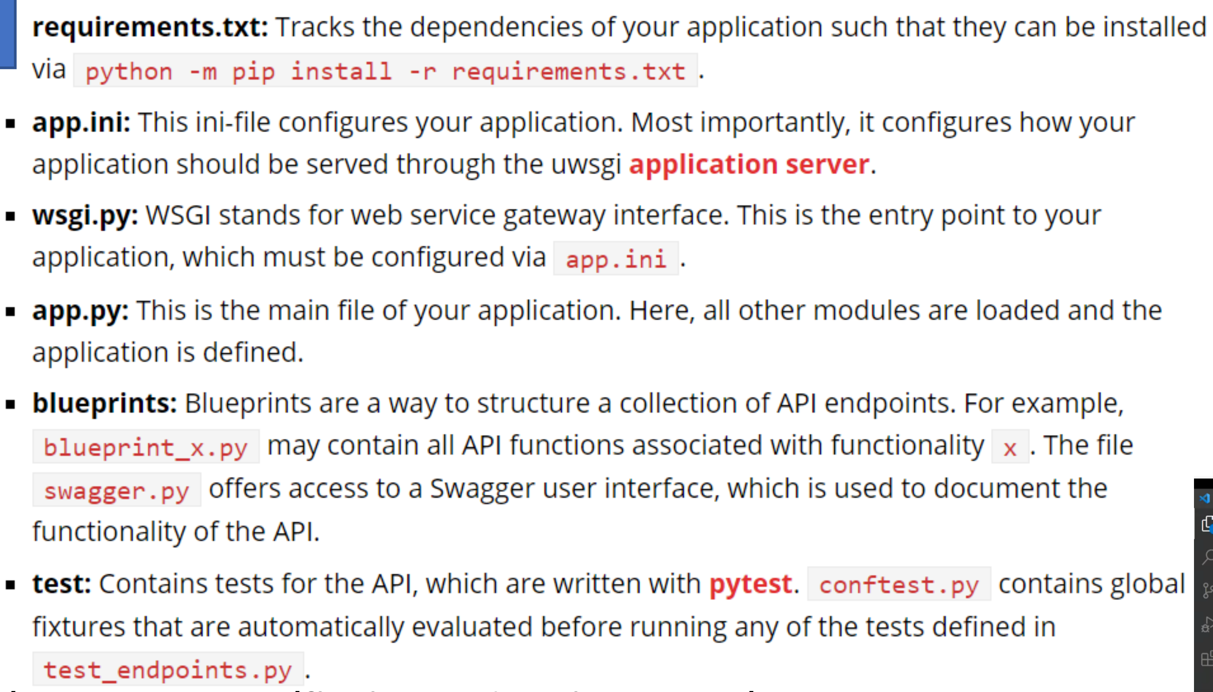
- Module Init – (project_root/app_name/admin/__init__.py) – required to enable the app
- Module URL – (project_root/app_name/admin/url.py) – Url definitions of each module
- App root Init – (project_root/app_name/__init__.py) – Not necessary to define the entire app within __init__.py
- Module Views – (project_root/app_ame/admin/views.py) – Defines views for each module. Separate '.py.' Files as the project scale to ensure they are accessible to URLs.
- Module Templates – (project_root/app_name/admin/templates/admin/main.html) – Normal template folder.

How to Structure a RESTful Flask Application



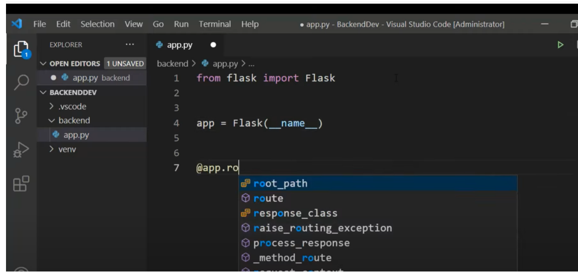
```
flask_app/
|    requirements.txt
|    app.ini
|    wsgi.py
|
|—src/
|   |    app.py
|   |    api_spec.py
|   |
|   |——blueprints/
|   |   |    blueprint_x.py
|   |   |    blueprint_y.py
|   |   |    swagger.py
|
└——test/
    |    conftest.py
    |    test_endpoints.py
```

- The file structure for a minimal Flask application that offers only a REST API should look something like this:



requirements.txt: Tracks the dependencies of your application such that they can be installed via `python -m pip install -r requirements.txt`.

- **app.ini:** This ini-file configures your application. Most importantly, it configures how your application should be served through the uwsgi **application server**.
- **wsgi.py:** WSGI stands for web service gateway interface. This is the entry point to your application, which must be configured via `app.ini`.
- **app.py:** This is the main file of your application. Here, all other modules are loaded and the application is defined.
- **blueprints:** Blueprints are a way to structure a collection of API endpoints. For example, `blueprint_x.py` may contain all API functions associated with functionality `x`. The file `swagger.py` offers access to a Swagger user interface, which is used to document the functionality of the API.
- **test:** Contains tests for the API, which are written with **pytest**. `conftest.py` contains global fixtures that are automatically evaluated before running any of the tests defined in `test_endpoints.py`.



```
from flask import Flask


app = Flask(__name__)


@app.ro
```

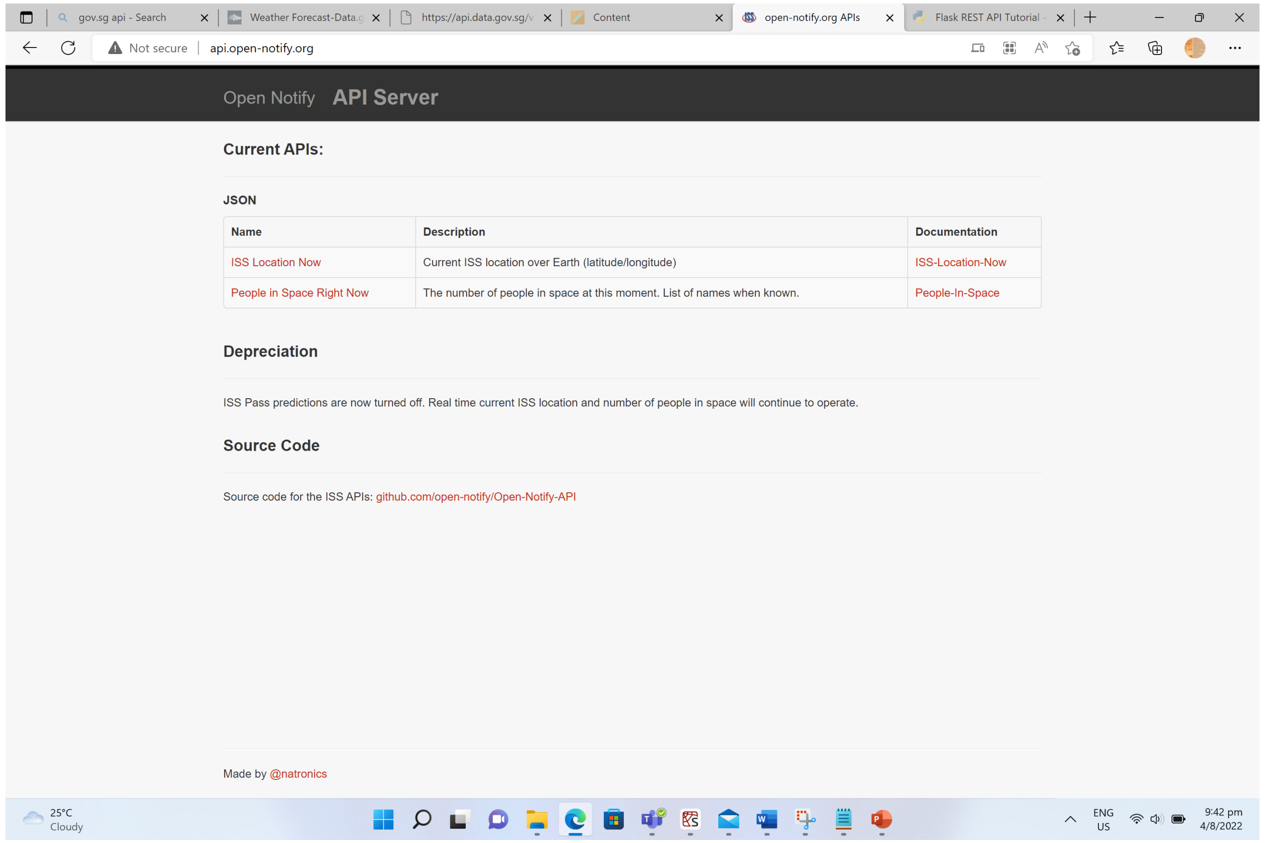
```
root_path
route
response_class
raise_routing_exception
process_response
_method_route
```

https://www.datascienceblog.net/post/programming/flask-api-development/

Optional : Handling external APIs

- http://api.open-notify.org

- How to handle the json return?



Open Notify **API Server**

Current APIs:

JSON

Name	Description	Documentation
ISS Location Now	Current ISS location over Earth (latitude/longitude)	ISS-Location-Now
People in Space Right Now	The number of people in space at this moment. List of names when known.	People-In-Space

Depreciation

ISS Pass predictions are now turned off. Real time current ISS location and number of people in space will continue to operate.

Source Code

Source code for the ISS APIs: github.com/open-notify/Open-Notify-API

Made by @natronics