FastAPI

FastAPI

- A modern, fast (high-performance) web FW for building APIs with Python
- https://fastapi.tiangolo.com/
- First step is to install - 'pip install fastapi[all]' to include all optional dependencies and features
- Also includes 'uvicorn' (provides server for running code)

In [ ]:
```
### Hello, world!
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {'message': 'Hello World'}
```

- Run the server using `uvicorn <filename>:app --reload`
- `<filename>` is the Python "module"
- `:app` is the Python object containing the FastAPI instance
- `--reload` (used for development / causes server restart as code changes)
- Use http://127.0.0.1:8000 to access GET path to base request
- Use http://127.0.0.1:8000/docs to access Swagger UI for API
- Allows testing of API in a graphical way

- FastAPI uses OpenAPI to define a schema for your API (Application Programming Interface)
- Includes definitions of format for data expected to be sent to and returned from your API
- Uses "JSON Schema" which is a standard for JSON-based data
- FastAPI automatically generates a JSON schema that can be viewed at http://127.0.0.1:8000/openapi.json
- Used to build out the graphical Swagger UI for testing; can also be used for validation

In [ ]:
```
from pydantic import BaseModel
```

```
class Customer(BaseModel):
    '''
    Uses pydantic to manage model + serialization to/from JSON
    '''
    id: int
    first_name: str
    last_name: str
    address: str
    city: str
    state: str
    zip: str
```

- REST (Representational State Transfer) defines standards for accessing resources via URI
- Access to a "thing" in REST is driven through a path
- That path can access top-level (like `/`) or it can drill down into a resource hierarchy
- `/customers/1` (for retrieving customer 1), `/customers` (for retrieving all customers), or `/customers/1/orders` for retrieving customer 1's orders
- REST utilizes a set of verbs that we can use to dictate the type of action we want to take; most common include:
  - GET
  - POST
  - PUT
  - DELETE
- REST requests can report back various status codes indicating success or failure

https://restfulapi.net/

https://www.developer.com/web-services/best-practices-restful-api

- In our Python code, we use a combination of decorator and method
- For example, `@app.get()`, `@app.post()`, `@app.put()`, and `@app.delete()`
- Apply to method that defines logic for route

In [ ]:

```
import uvicorn
from fastapi import FastAPI, Request
from order_system.database.database import Database
from order_system.models.customer import Customer
from typing import List
```

```python
app = FastAPI()

database = Database()

@app.get("/")
async def root():
    return {'message': 'Hello World!!'}

@app.post("/customers")
async def insert_customer(customer: Customer):
    database.insert_customer(customer)
    return customer

@app.get("/customers/{id}")
async def get_customer_by_id(id):
    customer = database.get_by_id(id)
    if customer:
        return database.get_by_id(id)
    else:
        return {}

@app.get('/customers', response_model=List[Customer])
async def get_all_customers():
    return database.get_all()

@app.put('/customers/{id}')
async def update_customer(id, body: Customer):
    if database.update_customer(id, body):
        return {'message': 'Customer updated successfully'}
    else:
        return {'message': 'Customer not found'}

@app.delete('/customers/{id}')
async def delete_customer(id):
    if database.delete_customer(id):
        return {'message': 'Customer deleted successfully'}
    else:
        return {'message': 'Customer not found'}

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=5000)
```

Lab: API using FastAPI

- Using the customers example as a template, create a new API for bank accounts
- Include properties of your choosing to represent but make sure the account has a unique id
- Use an in-memory dictionary (similar to the example) for data storage
- Test each route using the `/docs` endpoint on the running API