OPEN FOOD TRUCKS FINDER

Simple command-line program that will print out a list of food trucks, given a source of food truck data from the San Francisco government's API.

Setup & Install Dependencies For Program

1. Install virtualenv using `pip install virtualenv`
2. Pull down from git or unzip the show_open_food_trucks/ service source code.
3. Get to the root directory of the service `cd /path-to-project/show_open_food_trucks/`
4. Create a new virtual environment `virtualenv venv`
5. Active the new virtual environment `source venv/bin/activate`
6. Install requirements into virtual environment `venv/bin/pip install -r requirements.txt`

Running Program

1. Run the program by `venv/bin/python show_open_food_trucks.py` or `python show_open_food_trucks.py`

- If there are more than 1 page. you will be prompted to enter a page number to view additional results

Assumptions / Additional Details:

- Using the ***applicant*** key/value response to sort by alphabetically
    - I do an in-order sort. So as we are parsing the request we build a list of FoodTruck object they are sorted.
- Using the ***start24*** and ***end24*** key/value response to determine if a food truck is open
    - Get current time, than compare with these 2 values to see if they are between them
- As it displays NAME and ADDRESS, I added HOURS that show what hours the food truck is open
- Using python module ***tabulate*** to format the output for the command-line

Roadmap: (proof-of-concept to fully featured web application)

Process of building the web application

To turn this proof-of-concept command-line program to a fully featured web application we'd first want to determine a core set of components and functionality that are required by the web application. For example some core functionality would be, user logins, favorite food trucks list, recommended food trucks, closest food truck to me, list view of food trucks, details view of food trucks, and filtering/searching (by different dates, locations...etc)

Once we have decided our core set of functionality we want to support, we can start looking at the components and technology tools we will use to build this web application at scale. These components would outline details related to what database to use, what web-service(s) framework or frontend technology do we use to support the core functionality requirements. Maybe we want to run a micro-service architecture and break these components out into their own services. This step is where we'd make these decisions.

Once we've determined our components technologies, the next process I'd take is to determine (if applicable) what data models and/or persistent data structure (relational / non-relational) we need to support our functionality. (i.e data-flow diagram) Next we need to determine the structure of our REST API layer (again, if applicable) and how this API layer interconnects with the data model layer. We also want to at this step vet Socrata API and if it can support users either in realtime. Or offloading it to a backend process in semi-realtime or daily-time.

Once this is all mapped out we can begin breaking this out and start building each of these core features supported by the underline component technologies we decided on previously.

Process of building containerized services and CI/CD pipelines

To support the web application we would want to setup each of our service(s) / app(s) in a Docker container and setup corresponding CI/CD pipelines with test and production environments using a combination of Jenkins jobs, github hooks and cloud service (like AWS, or GCP) for hosting.

Process of setting up cloud infrastructure at scale

Initial setup would consist of a load balancer the directs traffic to a set of containerized apps (In AWS or GCP). Separation of RDS read / write instances and determine what caching technology to use. Since this data doesn't update to often. You could use a redis server to cache some of the outbound responses.

Final thought

Above at a high-level outlines the steps I'd take a proof-of-concept or prototype app and scale it to a fully feature web application. There are some things I did leave out that should also be discussed. One being tests (integration, load, unit tests) and the other being, what security is needed for the application. (HTTPS, password management...etc.) All in all this would be the basic approach I'd use to tackle this kind of project.

If I left anything out, I'm happy to discuss further.

Self-Evalution Code Review (area's of improvment)

- Method ***__paginate_food_trucks_list*** and ***__build_food_trucks_list*** should be combined to improve efficiency.
    - One idea was to build the pagination at the same time I am initially validating and building the list of FoodTruck objects.

- Revisit line 48: `self.__build_open_food_trucks()` I'm building this in the class constructor. Maybe have a explicit public method that needs to be called.
- Change out print() to some kind of logger

Example Response from Socrata API


```
[{
   "dayorder":"2",
   "dayofweekstr":"Tuesday",
   "starttime":"10AM",
   "endtime":"6PM",
   "permit":"19MFF-00105",
   "location":"773 MARKET ST",
   "locationdesc":"Pushcart located on Market St. 7 linear feet West of the Fire Hydrant. Must maintain 8 linear feet clearance f
   "optionaltext":"Kettle Corn, Funnel Cakes, Lemonade, Beverages, Flan, Hot Dogs, Falafel, Hot and Cold Sandwiches, French Fries
   "locationid":"1341056",
   "start24":"10:00",
   "end24":"18:00",
   "cnn":"8746103",
   "addr_date_create":"2011-11-15T13:48:04.000",
   "addr_date_modified":"2011-11-15T13:50:08.000",
   "block":"3706",
   "lot":"096",
   "coldtruck":"N",
   "applicant":"Kettle Corn Star",
   "x":"6011164.82111",
   "y":"2114324.40143",
   "latitude":"37.786160934428665",
   "longitude":"-122.40512731130576",
   "location_2":{
      "type":"Point",
      "coordinates":[
         -122.405127311306,
         37.7861609344287
      ]
   }
}]
```