**Foodtrucks Documentation**

Foodtrucks is a location based web application designed for the backend track position application of Uber.

Foodtrucks is developed on the top of [Django REST Framework](http://www.django-rest-framework.org/), which is a RESTful version of [Django](https://www.djangoproject.com/). Foodtrucks enables REST API for easy access to server data via http.

1. Architecture
2. Models

Only one table is created in the database, FoodTruck, keeping a foodtruck’s information like ID, applicant, food items, location, etc. For the purpose of RESTful interface, a serializer is also created on top of it. Refer to foodtruck/models.py and foodtruck/serializers.py for detailed implementations.

1. APIs
   1. Retrieve all / Create requests:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| URL | HTTP method | Returns | Parameters | Normal Response |
| /foodtrucks/ | GET | list of all records |  | 200 |
| /foodtrucks/ | POST |  | foodtruck object (objected is required) | 201 |

Example:

curl –X POST http://hostport/foodtrucks -d ‘{“objectid”:1, “applicant”:”John”, “latitude”: 37.77, “longitude”: -122.2}’ -H "Content-Type: application/json"

* 1. Retrieve / Update / Delete by ID:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| URL | HTTP method | Returns | Parameters | Normal Response |
| /foodtrucks/<id> | GET | specified record |  | 200 |
| /foodtrucks/<id> | PUT |  | foodtruck object (objected is required) | 200 |
| /foodtrucks/<id> | DELETE |  |  | 204 |

Examples:

curl http://hostport/foodtrucks/1

curl –X PUT http://hostport/foodtrucks/1 -d ‘{“objectid”:1, “applicant”:”John”, “latitude”: 37.77, “longitude”: -122.2}’ -H "Content-Type: application/json"

curl –X DELETE http://hostport/foodtrucks/1

* 1. Retrieve by keyword:

Search for the food trucks that contain the keyword (case insensitive) in either applicant names or food items.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| URL | HTTP method | Returns | Parameters | Normal Response |
| /foodtrucks/bykeyword | GET | list of records | <keyword> | 200 |

Example:

http://hostport/foodtrucks/ bykeyword?keyword=pizza

* 1. Retrieve by location:

Given a target latitude, longitude, radius of search, and limit of number of results, find the food trucks nearby the target location

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| URL | HTTP method | Returns | Parameters | Normal Response |
| /foodtrucks/bylocation | GET | list of records | <latitude>,<longitude>,  <radius> (default=1.0mi),  <limit> (default = 15) | 200 |

Examples:

http://hostport/foodtrucks/ bylocation?latitude=37.777&longitude=-122.222

http://hostport/foodtrucks/ bylocation?latitude=37.777&longitude=- 122.222&radius=0.5

http://hostport/foodtrucks/ bylocation?latitude=37.777&longitude=-122.222&rasius=0.5&limit=10

1. Tests

There are two test sets: functional tests and load tests.

1. Functional tests

Functional tests are conducted within the Django framework locally to test the basic functionalities of the web server.

Required packages: numpy, mysql. If you don’t have mysql installed, open code\_challenge/local\_settings.py, modify

*DATABASES = {*

*'default': {*

*'ENGINE': 'django.db.backends.mysql',*

*…*

*}*

*}*

to

*DATABASES = {*

*'default': {*

*'ENGINE': 'django.db.backends.sqlite3',*

*…*

*}*

*}*

b. Load tests

Load tests utilizes [funkload](http://funkload.nuxeo.org/) to test the throughput of the website. Three cycles are conducted, each with 10, 20 and 100 virtual clients sending requests to server. At each cycle, the clients keep sending requests concurrently every 0.01s, for 60s. The request is searchByLocation task, which involves heavy database query and computation.

Detailed reports can be found under tests/loadtests

The results of the latest test

|  |  |  |  |
| --- | --- | --- | --- |
|  | 10 clients | 20 clients | 100clients |
| Success | 457 | 463 | 1254 |
| Failure | 0 | 1 | 48 |

To test, start MySQL server if installed, then run script:

cd tests && ./test.sh.

\*script is tested under OSX.