restaurantAPI_docs Documentation

Release v1.0

Navendra Jha

Apr 25, 2018

Contents

Online Restaurant API system using flask, flask-restful, sqlalchemy, marshmallow

The project has been developed using Flask- A python Micro-web framework and other additional packages describe below in Tech Stack Section.

Github link for the project - https://github.com/navi25/RestaurantAPI

CHAPTER 1

Installation

Before we begin, kindly install following on your system:-

- python3.x
- Virtualenv

CHAPTER 2

How to Run the App?

- cd path/to/workspace
- git clone https://github.com/navi25/RestaurantAPI
- cd RestaurantAPI
- virtualenv -p ‘which python3’ venv
- source venv/bin/activate
- pip install -r requirements.txt
- python3 run.py

Everything should be ready. In your browser open http://127.0.0.1:5000/

Since Redis-Server is used for database optimisation After running the app, type in following in terminal to establish redis-connection

- redis-server

CHAPTER 3

REST Endpoints

There are three major objects in the app:-

- Restaurants
- Menu
- Food Items (Menu Items)

The endpoints and the corresponding REST operations are defined as follows:-

- **RESTAURANTS**
  - http://127.0.0.1:5000/api/v1.0/restaurants/
    - **GET** : This method on above URL returns all the restaurants available in the database in json format
    - **POST** : This method posts a new restaurant and accept *application/JSON* format for the operation with "name" as the only and the required parameter for the JSON.
    - **PUT** : Same as POST with additional feature of updating the restaurant object too.
    - **Delete** : This method deletes the given restaurant if the *restaurant_id* exists.
- **Menu**
  - http://127.0.0.1:5000/api/v1.0/menus/
    - **GET** : This method on above URL returns all the menu available in the database in json format
    - **POST** : This method posts a new menu and accept *application/JSON* format for the operation with "name" and "restaurant_id" as the required parameter for the JSON.
    - **PUT** : Same as POST with additional feature of updating the menu object too.
    - **Delete** : This method deletes the given menu if the *menu_id* exists.
- **Food**
  - http://127.0.0.1:5000/api/v1.0/foods/
    - **GET** : This method on above URL returns all the foods available in the database in json format

- **POST** : This method posts a new food and accept *application/JSON* format for the operation with “name” and “restaurant_id” as the required parameter for the JSON.
- **PUT** : Same as POST with additional feature of updating the menu object too.
- **Delete** : This method deletes the given menu if the *food_id* exists.

CHAPTER 4

Additional endpoints

- http://127.0.0.1:5000/api/v1.0/restaurants/{id}

  Returns the particular restaurant with id = id if it exists
- http://127.0.0.1:5000/api/v1.0/restaurants/{id}/foods

  Returns all the foods available in the particular restaurant with id = id, if the restaurant it exists
- http://127.0.0.1:5000/api/v1.0/restaurants/{id}/foods/{food_id}

  Returns the particular food with id = food_id in the particular restaurant with id = id if it exists.
- http://127.0.0.1:5000/api/v1.0/restaurants/{id}/menus

  Returns all the menus available in the particular restaurant with id = id, if the restaurant it exists
- http://127.0.0.1:5000/api/v1.0/restaurants/{id}/menus/{menu_id}

  Returns the particular menu with id = menu_id in the particular restaurant with id = id if it exists.

CHAPTER 5

Unit Testing Endpoints

The Tests for all the modules are located in **tests** directory and can be fired in two ways:-

- Individually by running their individual test modules
- All at once by running **TestAll** module which look for all the available modules in the directory and fires the test cases one by one.

The Flask's Unittest modules were used for developing the testcases.

CHAPTER 6

Tech stack

- Flask - Web Microframework for Python
- Flask-restful - Extension for flask for quickly building REST APIs
- Swagger - Automatic Documentation for the REST endpoints
- Flask-migrate - An extension that handles SQLAlchemy database migrations for Flask applications using Alembic.
- Marshmallow - A serializer and deserializer framework for converting complex data types, such as objects to and from native Python data types.
- Flask-sqlalchemy - This is an extension of flask that add supports for SQLAlchemy
- Flask-marshmallow - An integration layer for flask and marshmallow.
- Marshmallow-sqlalchemy - This adds additional features to marshmallow.
- Sqlite3 - Database for the project. It comes built in with python.
- RedisDB - Key-Value based No-SQL DB to oprimize relational

database by improving Read by caching data efficiently. - Flask-Redis - An flask extension of [RedisPy](http://redis-py.readthedocs.io/en/latest/)

to easliy used Redis with Python and Flask easily.

CHAPTER 7

Development Thought process

- Used Micro service Architecture for proper decoupling of service.
- Documentation is hard, hence used an automatic document generating tool – Swagger to ease out the process.
- Test driven development is useful and leads to less errors in later stages of development.
- Dependency injection helps a lot in Test driven development and also in making the project more modular and flexible. Though couldn't use in the current project but would surely update the project using flask-injector.
- RedisDB is used as caching layer to improve read efficiency.
- Used Flask because it's flexible and can be plugged with all the necessary modules on the go.