Locale.AI

Backend Write Up

DJANGO CRUD API

Introduction

This is a writeup for the source code of a CRUD API built with Django. This backend is hosted on Heroku, and stores its data on a postgreSQL database hosted on ElephantSQL, which provides access to a free DB as a service. It can also be hosted on a local machine by following the instructions given later, but it uses a locally stored SQLite DB.

Cloning the repository

Use

```
git clone https://github.com/Arihant-Joshi/localeAIbackend/tree/master
```

on bash, or download the zip from the browser, and unzip.

Running a local server instance

1. Make sure you have the correct version of python, as given in runtime.txt.
2. Install all dependencies, as given in **requirements.txt**, using pip-

```
pip3 install requirements.txt
```

3. Replace ./backend/settings.py with settingsLOCAL.py. Make sure you change the name as well. The settingsLocal.py changes the settings of the Database, as well as hard codes the settings like SECRET_KEY, which are environment variables.
4. Run the manage.py file with following arguments

```
python3 manage.py makemigrations
python3 manage.py migrate
```

This creates a SQLite database locally.

5. Add a superuser to access the database.

```
python3 manage.py createsuperuser
```

   Enter the credentials as required.

6. Run the server

```
python3 manage.py runserver
```

   This runs the server at http://localhost:8000

Using the API

The following ways of using the API have been provided.

1. **Using the browser** - The following assumes {URL} to be the url of the server, whether it's http://localhost:8000, or https://locale-ai.herokuapp.com.

   Making Requests to the API is password protected, which was provided by you in step-5 of running the server locally.

   NOTE- Username and Password for accessing the server on heroku are provided in the E-Mail, to not make it public.

   The following functions can be performed-

   - Go to {URL}/api/basic to Get All data entries, and POST a new entry.
   - Go to {URL}/api/basic/{booking_id} to get only the entry with unique booking_id, update it(PUT), or DELETE it

When entering the Data in POST/PUT fields, use a JSON Dictionary format, with Column names in double quotes, and Date Fields following the formats specified in the DATETIME_INPUT_FORMATS field in settings.py.

Alternatively, one can go to {URL}/admin, login there, navigate to the Locationss model under the app LOCATIONS_CRUD, and make changes there(This method does not use the API).

2. **Using the CLI** - A python client Application has been provided in this Git Repository, which makes use of the backend's API to make changes to the Database, using JSON/CSV files as input(PUT/POST), or output(GET). In case of failed requests, the CLI outputs the HTTP Error Status Code.

Structure of the Application

This web application uses python's Django framework to keep track of and access the data models. Some feature of this Application are-

1. backend/settings.py - This django files stores various settings required to deploy the application. This keeps track of various applications used, models declared, language, time zone, static file locations, and the SECRET_KEY. As is advisable, the Secret_Key is by default stored as an environment variable on the heroku server.
2. backend/urls.py - This contains various regex expressions of the url, which lead to various Views.
3. API Views - Currently, there are three user defined API views, as given in locations_crud/api.py-
    a. LocationsAPI(URL:"/api/basic/") - Provides GET(all) and POST
    b. LocationsAPIDetails(URL:"/api/basic/<booking_id>") - Provides GET(by booking_id), PUT and DELETE
    c. UserAuthAPI(URL:"/api/user") - Provides the login session token, to authenticate the user on, say, a CLI.
4. Models - We need only one Model - Locations, defined in locations_crud/models.py. In order to get the attributes of each field(Unique, Optional), and Field Type, the given Data.csv file was read into a pandas dataframe, and the following operations run on them.

```
import pandas as pd

data = pd.read_csv("Data.csv")

print("Unique Values")
print(data.nunique(),"\n\n")
print("Null Values")
print(data.isna().sum(),"\n\n")
print("DataTypes")
print(data.dtypes,"\n\n")
```

This showed location_id as a Unique(Number of unique values equal to rows), Required(No null values) Integer, and so on.

5. Serializers - As given in locations_crud/serializations.py, we have two user defined Serializers.
    a. LocSerializer - Accessed by LocationsAPI APIView's POST method. Declares all fields of the model as declared in models.py.
    b. LocSerializerOptional - Accessed by LocationsAPIDetails APIView's PUT method. Overrides the required methods to Optional, since this is used only to update an already existing field.

Possible Expansions

1. A basic frontend with a login page, and allowing a user to upload/download the data as csv or json would make accessing it through the browser easier.
2. The following improvements would have to be made for a production level backend-
    a. A load balancer could distribute client requests across various backend servers. This allows multiple requests to be processed simultaneously, and allows modularity, as a backend server can be easily replaced or have its load distributed in case one server crashes.
    b. A request logging system could be implemented. This is useful in case a server crashes while a request is being processed, and changes are being made to the database. Now, when the server is restarted, it can check the log for any pending requests, and pick up from there.