SENSIDEV

Sensidev - Python Test

Context

Imagine you are building a (very simplified) analytics backend system to be used by web & mobile software developers to create fancy interfaces with data provided by your system.

Requirements

As a User,
I want to compare latest current weather conditions between different locations,
So that I better understand differences over a period of 24 hours.

Technical

Build a CRUD system that manages a list of locations with their weather parameters.
Each location can have one or many parameters e.g. Temperature, Humidity, Wind speed etc.
You are responsible to get raw historical data from a public weather API at your choice, and compute the aggregate fields (described below).

Prerequisites

- Github/Bitbucket/GitLab account, you'll create a single **private** repo.
- Heroku or similar account, you'll deploy your code on a server for reviewers to test.

Stack

- Python/Django
- Django Rest Framework

API

Activities and resources your backend system should handle.

Activity	**Verb**	**Noun**
User can **list** her **locations**.	`GET`	`/locations/`
User can **view** a **location** detail page.	`GET`	`/locations/:id/`
User can **update** a **location** detail page.	`PATCH`	`/locations/:id/`

User can **create** a new **location**.	`POST`	`/locations/`
User can **list weather parameters** within a **location**.	`GET`	`/locations/:id/parameters/`
Users can **add weather parameters** to a **location**.	`POST`	`/locations/:id/parameters/`
User can **view** a **weather parameter** detail page.	`GET`	`/locations/:id/parameters/:id/`
User can **delete weather parameter** from a **location**.	`DELETE`	`/locations/:id/parameters/:id/`
User can **delete** a **location**.	`DELETE`	`/locations/:id/`

Fields

Location

- **Id -** primary key, e.g. `3`
- **Description -** string user can modify, e.g. `"My home town"`
- **Parameters -** URL link to parameters API point e.g. `"https://example.com/locations/3/parameters/"`
- **Aggregation -** for each parameter compute average, min and max for last 24 hours, e.g.

```
[
  {
    "id": 1,
    "name": "Temperature",
    "avg": 23.3,
    "min": 22.1,
    "max": 29.1,
    "units": "°C"
  },
  {
    "id": 1,
    "name": "Relative Humidity",
    "avg": 55.3,
    "min": 50,
    "max": 65.4,
    "units": "%"
  }
]
```

- **Details** - add as many 3rd party weather API fields you consider necessary e.g.
    - Location_id or Key - their identification for a location
    - Name - their location name
    - Url - the url from where to collect data

Parameter

- **Id** - primary key e.g. 11
- **Name** - string user can modify e.g. `"Temperature"`
- **Location** - URL link to location e.g. `"https://example.com/locations/3"`
- **Values** - last available values as list e.g.

```
[
  {"x": 1599040000, "y": 26.3},
  {"x": 1599040100, "y": 25.2},
  {"x": 1599040200, "y": 22.1},
  {"x": 1599040300, "y": 22.3},
  {"x": 1599040400, "y": 24.2},
  ...
]
```

- **Aggregation** - similar to locations, but only one object for the current parameter.
- **Units of measurements** e.g. `"°C"`
- **Details** - add as many 3rd party weather API fields you consider necessary e.g.
    - Parameter_id or Key - their identification for a parameter
    - Name - their parameter name

Admin

Optionally you can leverage the power of the Django Admin UI system to manage your resources.

Tips

- First, do a little research about what free public weather API to use.
    - https://rapidapi.com/blog/access-global-weather-data-with-these-weather-apis/
- Git related
    - Commit often, usually when you reach a working state of a concept.
    - Commit on `develop` branch, and only after you finished create a pull request.
- Use Postman or similar to test your REST API, and share the collection with us.
- Take your time and ask relevant questions to clarify requirements.
- Structure your questions to fit within a 30 min session with a senior developer.