Lab 8

Due date: Mar 14, 2020, 4:00am

Objective

In this lab, you will build a RESTful API that provides functionality to retrieve, add, edit, and delete data. The application will utilize the standard data format JSON.
In order to make Flask applications portable, you will use a requirements file.

Getting started

1. Import the starter project into Eclipse.
2. Rename the project to `lab8`*`FirstnameLastname`* where *`FirstnameLastname`* is your first and last name.
3. Open a local terminal and navigate to the project folder `lab8`*`FirstnameLastname`*. (If you select the project folder in the Project Explorer before clicking the `Open a Terminal` icon, the terminal will open in the project folder.)
4. In the terminal, setup a virtual environment for your project, called `venv`. The files created for the virtual environment differ depending on the underlying operating system. Therefore, the virtual environment folder is not included in the starter project and needs to be created.
5. Set the interpreter for the project in Eclipse. Use the interpreter from the newly created virtual environment.
6. Activate the virtual environment.
7. **Previously, we have installed all required packages one-by-one. Instead, we can take a snapshot of all installed packages at any time when the virtual environment is activated and store the list of packages in a file that is, by convention, called `requirements.txt`. This makes it easier to setup your project on a different machine. To install all packages from the requirements file use the command:**

   ```
   python -m pip install -r requirements.txt
   ```

   The requirements file included in this starter project includes all packages necessary for this lab. Use the above command (i.e. `pip install -r requirements.txt`) to install those packages.

   Whenever you add new packages, you should re-create the requirements file. This way it is easy to setup the development environment on a different machine. To create the requirements file use the command:

   ```
   python -m pip freeze > requirements.txt
   ```

   However, you will not need to do so in this lab since the included requirements file includes already all necessary packages.

8. The starter project also includes file `.flaskenv`. Note that this file is a simple textfile and its syntax does not depend on the underlying operating system.
9. Run Flask.

The Starter Project

The starter project contains some code for a RESTful web API that is supposed to facilitate the sharing of baby name data. The resource with the baby name data is stored in the JSON file `names.json`. The file data is represented by a dictionary with a single key-value pair where the key is the string "names" and the value is a list of dictionaries:

```
{“names”: [{…},…,{…}]}
```

Each dictionary in the list represents a baby name and must have a key-value pair with key "name" and a unique string representing the baby name. For example:

```
{“name”: “Harper”, “ranking”: 9, “gender”: [“M”, “F”]}
```

In particular, that means that there cannot be two dictionaries that map the key "name" to the same baby name. Otherwise, there are no restrictions placed on each dictionary in the list. The file `names.json` in the starter project is already populated with some sample data.

The starter project contains three python modules:

- app.py

  This module contains the Flask application with the implementation for a single route with the endpoint /names. You will complete this module as instructed below.
- name_db.py

  The class NameDB should implement the functionality to read name data from names.json and to write name data to names.json. You will complete the class as instructed below.
- names.py

  The class Names maintains a list of names, where each name is represented by a dictionary. This class is already implemented and does not need to be modified.

The only endpoint installed by the starter project is `/names`. This endpoint should return all names stored in names.json. You can test this endpoint by requesting the URL `http://localhost:5000/names` through a browser. However, since the functionality to read from `names.json` is not yet implemented, an empty list is returned. Note that the list is wrapped in a dictionary as a list is not a valid JSON string.

Tasks

Part 1: Reading from and writing to a JSON file

In name_db.py, implement the methods `read_names` and `write_names`. See the documentation in names_db.py for details.

Part 2: Implementing the routes

Add routes to app.py so that the API handles the following HTTP requests:

- A GET request to the endpoint \names (already implemented)

  The API will send an HTTP response that returns the JSON representation of all names. A status code of 200 should be returned.

- A GET request to the endpoint \name\<*name*>:

  The API will send an HTTP response that returns a JSON representation of the name data for name <*name*>. A status code of 200 should be returned if the name data for <*name*> is returned successfully. If the resource does not contain name data for <*name*>, the status code 404 should be returned.

- A POST request to the endpoint \name. The payload of the POST request must contain the valid JSON encoding of a name. In particular, the key “name” has to be included. For example, the payload may be as follows:

  ```
  {“name”: “Harper”, “ranking”: 9, “gender”: [“M”, “F”]}
  ```

  The API will add the submitted data to the names list and to the JSON file if no name data with the specified value for the “name” key exists. In that case, the HTTP response will return the JSON representation of the added data and the status code 201. If the operation failed because the specified name key exists already or no name key has been included in the submitted data, the status code 400 is returned with a suitable error message. The error message should be a JSON string.

  The file names.json should be updated in case of changes to the name data.

- A PUT request to the endpoint \name. The payload of the PUT request must contain the valid JSON encoding of a name. In particular, the key “name” has to be included. For example, the payload may be as follows:

  ```
  {“name”: “Harper”, “ranking”: 9, “gender”: [“M”, “F”]}
  ```

  The API will add the submitted data to the names list and to the JSON file if no name data with the specified value for the “name” key exists. In that case, the HTTP response will return the JSON representation of the added data and the status code 201. If name data with the specified value for the “name” key exists, then that name data is replaced by the name data sent in the payload of the PUT request. The HTTP response will return the JSON representation of the new data and the status code 200. If the operation failed because the payload data is invalid, the status code 400 is returned with a suitable error message. The error message should be a JSON string.

The file names.json should be updated in case of changes to the name data.

- A DELETE request to the endpoint \name\<*name*>:

  The API will delete the data of the specified name if it exists. If name data has been deleted, the JSON-encoding of the deleted data and the status code 200 are returned through the HTTP response. If the specified name data does not exist, the HTTP status code 404 is returned with a suitable JSON-encoded error message.

  The file names.json should be updated in case of changes to the name data.

Submission

Archive the project folder `lab8`*`FirstnameLastname`* and submit the archived folder as `lab8`*`FirstnameLastname`*`.zip` to the course website. The archived project should contain all files and folders in folder `lab8`*`FirstnameLastname`*, except for the folder `venv` with your virtual environment.