EMPLOYEES/REPORT DATABASE MANAGEMENT

This is a REST API written in Python to manage a database of employees and related reports. The app is meant to run locally with local databases that can easily be created via some Python scripts (see instruction below)

Requirements:

- Python 3.6 or +
- Python modules: flask, sqlite3, requests, uuid, subprocess
- pytest for testing the app

any missing python module (including pytest) should be easily installed with pip (e.g: pip install flask). Make sure you are using pip3 version.

The following setup instructions (as well as the launch of the Python API) are supposed to be performed from a terminal. After the app is launched (and running), its functionalities can be accessed via browser (see instructions below) or via terminal (e.g. curl).

1 SETUP INSTRUCTIONS

Download the repository from github:

git clone `https://github.com/flaviaaleotti/employee_reports.git`

and enter the employee_reports/ directory. Inside the repository, you will find two python scripts that will automatically generate two default SQL databases (for employees and related reports). Generate the employees.db and reports.db databases by running

```
python build_employees_database.py
python build_report_database.py
```

from terminal inside the employee_reports/ directory.

The pre-generated databases employees.db and reports.db are already available in the repository, but you can run the above scripts to reset the database to the starting status at anytime in the future.

Now you are ready to start the API! Run from terminal:

```
python RESTAPI.py
```

2 TESTING

Before using the API through your browser, it is better to test it with pytest in order to check the endpoints. To run the tests, open another terminal (leave the API running, otherwise all tests will fail), go to the employee_reports/ directory and simply type

```
pytest
```

you can also run the above pytest command with -v option to increase verbosity if you want to check each single endpoint test.

This command will run the tests that are stored inside the PYTEST/ directory. If all tests are successfully passed, you can open your browser and start using the API!

REMEMBER: the API should be always kept running in order to use it, so make sure that you do not close the terminal where you have your 'python RESTAPI.py' command running :)

3 USING THE API

This API is designed to help you interact with the employee.db and report.db databases. These are **SQL** databases, in which items are stored in tables. Each row in a table corresponds to an element (an employee or a report in our case) and is called a **record**, while each column corresponds to a particular piece of information characterizing the records (e.g: name, surname, job title etc. in case of an employee). The table columns are called **fields**.

3.1 DATABASE DETAILS

SQL databases are conceptually simple and easy to interact with through Python flask module, and that is why they were chosen for this API.

The **employee items** are characterized by the following fields:

- **id** : this is a unique identified (integer number) characterizing the employee
- **first_name** : first name of employee (string)
- **last_name** : last name of employee (string)
- **username** : employee username (string). The username is required to be unique within the database.
- **email**: institutional email address of employee, characterized by the company.com domain (string)
- **title** : employee job title (string)
- **department** : company department (string)

In the default database, the username field is created by combining the first letter of first name and the last name (e.g., employee Mario Rossi would have username mrossi), but this is not mandatory and you can crate new employees with whatever username (as long as the uniqueness is preserved).

In the default database, the department field is a capital single letter ('A', 'B', 'C'...), but any other string will be accepted upon creation of a new employee. The same holds for all other string fields (except username, see above).

The **report items** are characterized by the following fields:

- **id** : this is a unique identified (integer number) characterizing the report
- **title** : report title (string)
- **description** : report description (string)
- **employee_username** : username (string) of the employee related to the report.
- **priority**: report priority (string, either 'high' or 'low')

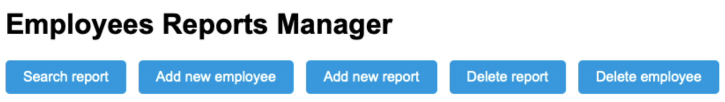
The employee_username field is the username of an existing employee, and the priority string can only assume 'high' or 'low' values (any other value will not be accepted).

For both employees and reports, the id identifier is automatically generated by the database whenever a new item is added. The id field is immutable and cannot be modified.

3.2 HOME PAGE

To navigate and use the API functionalities with your browser, open a browser window and go to `http://localhost:5000`

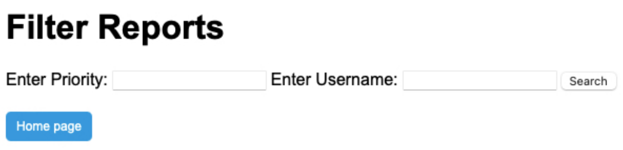
If the API is still running in your terminal (python RESTAPI.py, see before) you should see the following homepage



Employees Reports Manager

3.3 FILTER REPORTS

By clicking on "Search report", you will be redirected to the following Filter Reports page:



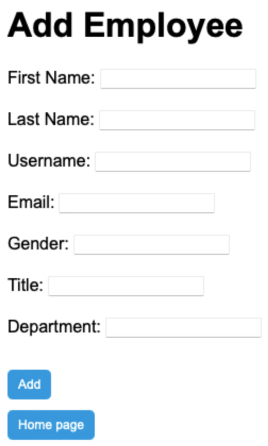
Filter Reports

Enter Priority: Enter Username:

in which you can enter a username or priority (or both) and press "Search" to filter the existing reports.

3.4 ADDING A NEW EMPLOYEE

From the home page, you can reach the 'Add Employee' page by clicking on the "Add new employee" button. The Add Employee page looks like this



Add Employee

First Name:

Last Name:

Username:

Email:

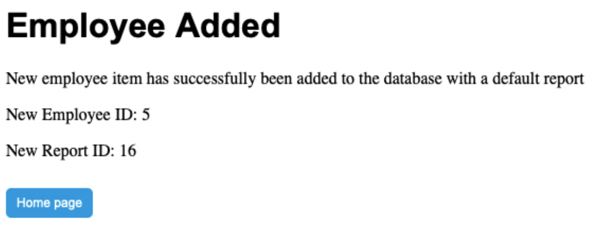
Gender:

Title:

Department:

you cal fill all the table fields (all fields are required) and press "Add".

If you provided a valid username, the procedure should be successful, and the new employee will be added to emplloyee.db database, showing this html page:



Employee Added

New employee item has successfully been added to the database with a default report

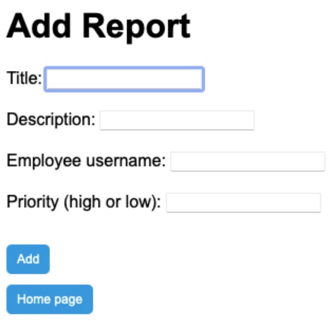
New Employee ID: 5

New Report ID: 16

Because each employee must have at least one report, the creation of a new employee is automatically creating also a new default item in the reports.db database (title: First report), linked to the username of the new employee.

3.5 ADDING A NEW REPORT

From the home page, you can reach the 'Add Report' page by clicking on the "Add new report" button. The Add report page looks like this



Add Report

Title:

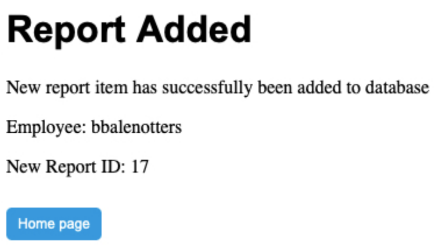
Description:

Employee username:

Priority (high or low):

you cal fill all the table fields (all fields are required) and press ”Add”.

If you provided a valid employee username, the procedure should be successful, and the new report will be added to report.db database, showing this html page:



Report Added

New report item has successfully been added to database

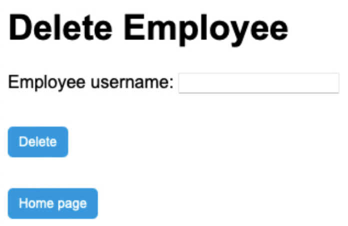
Employee: bbalenotters

New Report ID: 17

3.6 DELETING AN EMPLOYEE

If an employee stops working for our company, we can delete its record from the employee.db database easily.

From the home page, click on the ”Delete employee” button to reach the ”Delete employee” page



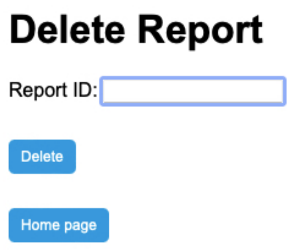
Delete Employee

Employee username:

Insert the username and press ”Delete”. If the provided username exists, the employee will be deleted from employee.db database, and all its related reports will also be deleted from reports.db database.

3.7 DELETING A REPORT

From the home page, click on the ”Delete report” button to reach the ”Delete report” page



Delete Report

Report ID:

Insert the report id and press ”Delete”. If the provided report id exists, the report will be deleted from report.db database.

3.8 OTHER FUNCTIONALITIES

Employees and reports can also be updated. However, implementing the update method trough HTML pages has proved to be harder than expected an I was not able to implement it in the html interface.

However, you can still edit one or more fields of an employee/report through command line with curl.

The only restrictions are:

- id (of report or employee) cannot be edited
- employee_username field of a report cannot be edited directly. In order to edit this field, you should edit the username field of the employee item, and in this way all its related reports will be updated as well
- username of an employee can be edited, but it must remain unique

3.8.1 UPDATING AN EMPLOYEE

In order to update an existing employee, we must know its id (as it is the only immutable field). One or more of the remaining fields can be edited at the same time with curl.

Here is an example that you can run on the default database, to change the job title of employee 1 from ”junior developer” to ”senior developer”, their last name to ”Curltest” and their username to ”fcurltest”:

```
curl -X PUT -H "Content-Type:␣application/json" -d '{"title":"senior
developer","last_name":"Curltest","username":"fcurltest"}'
http://localhost/update/employees/1
```

where the -X PUT option is needed to edit an item, the -H ”Content-Type: application/json” tells curl that we will pass the update content as json format, and -d precedes the actual data for the update request. The last part of the command contains the url for the update request, ending with the employee id (1 in our case).

If the update was successful, the message ”Employee fields and related reports successfully updated” should be printed in your terminal.

3.8.2 UPDATING A REPORT

Similar to what we have seen for employees, also reports can be updated (e.g., to change the priority from high to low or viceversa).

In order to update an existing report, we must know its id (as it is the only immutable field). One or more of the remaining fields (except for the employee username) can be edited at the same time with curl.

Here is an example that you can run on the default database, to change the priority of report 1 from ”low to high” and its title to ”Updated priority”:

```
curl -X PUT -H "Content-Type:␣application/json" -d '{"priority":"high",
"title":"Updated␣priority"}' http://localhost/update/reports/1}
```

where the -X PUT option is needed to edit an item, the -H ”Content-Type: application/json” tells curl that we will pass the update content as json format, and -d precedes the actual data for the update request. The last part of the command contains the url for the update request, ending with the report id (1 in our case).

If the update was successful, the message ”Report fields successfully updated” should be printed in your terminal.

3.8.3 SHOWING ALL DATABASE ITEMS

Having access to the full database is not a desirable feature for an API (for security/privacy reasons). However, for this simple case, it might be useful to see the full list of employees or reports in the database.

These functionalities are not accessible through the html interface directly, but are present in the API code an can be executed by going to the corresponding route (see the list_all_employees() and list_all_reports() methods).

4 TECHNICAL DETAILS

Here are some technical details explaining the structure of the code and the modeling decisions.

The directory templates/ contains some simple html templates that are used for the html interface with the API. They are very simple and only contain some code to generate the buttons/forms/links as well as to display messages.

The initial part of the RESTAPI.py code contains the functions to render the html pages when the corresponding route (url) is reached.

The filter (GET) and create (POST) functionalities are easily implemented in html, as they only require some forms in which the user can enter values that are used to search the database (GET) or create a new item (POST).

The delete (DELETE) functionalities require a two-step process: first we need to get an id (or username) through a form, then we need to perform a DELETE request for that specific item (i.e., when the user presses ”Delete”).

The update (PUT) functionality is a three-step process, as it first required to get an id (through a first form), then we need to retrieve the fields to be changed and the new values, and eventually we can send the update request for the corresponding item on the selected fields. I was not able to implement this through html interface (but it is still available with curl, as mentioned before)

One html template (that is not initially present in the repository) will be created on-the-fly by the filter_reports() function every time it is executed. This is done to visualize the results in a nice way, including links for next/prev page and home page. Every time a new search is performed, the html template for the results will be overwritten.

The RESTAPI.py code (after the definition of html interface and templates) contains functions for the basic API functionalities GET, POST, PUT and DELETE. Here is a short description of the functions:

- **list_all_employees()** and **list_all_reports()** functions are used to list the full employees.db and reports.db database (respectively). These functions are not meant to be used by general public, and are not present in the html interface. They are left for ”expert” users that can execute them by typing the route in their browser (or through curl);

- **get_employee_id()** and **get_report_id()** are used to search an item by id in the employees.db and reports.db database (respectively). These functions are only used for testing and have no html interface, but can again be used through curl or browser;
- **filter_reports()** function is the function that filters the reports.db database based on employee username and/or priority (filtering according to both fields is allowed). As mentioned before, it builds an html template with the results. Results are returned with paginated size 10, but ”Next page” and ”Previous page” buttons are provided in the html output page, so that the user does not have to perform more than one research;
- **create_new_employee_item()** adds a new employee item to employees.db database (POST method). It requires an input (either json or html forms) that is used to fill-in all the fields of the employees table (see above). The username passed for the new employee is also used to create a new default item in the reports.db database. If the function is called from a terminal (e.g. curl or pytest) it returns a json-type dictionary containing the details of the new employee and report items; if the function is called through the html interface, instead, a template html page with a message and the new IDs is displayed;
- **create_new_report_item()** adds a new report item to reports.db database (POST method). It requires an input (either json or html forms) that is used to fill-in all the fields of the reports table (see above). If the function is called from a terminal (e.g. curl or pytest) it returns a json-type dictionary containing the details of the new employee and report items; if the function is called through the html interface, instead, a template html page with a message and the new ID is displayed.
- **delete_report(report_id)** function checks that the report associated with the passed id exists, and deletes it;
- **delete_employee(username)** function checks that the employee associated with the passed username exists, and deletes it. If it exists, the passed username is also used to search and delete all reports associated with the deleted employee;
- **update_employee(employee_id)** function updates one or more fields associated with the selected employee. It first checks that the passed id corresponds to an existing employee and that all the passed fields are valid fields of the employees.db database, then performs the update. If ’username’ is among the updated fields, an update of all reports connected to the employee is also performed;
- **update_report(report_id)** function updates one or more fields associated with the selected report. It first checks that the passed id corresponds to an existing report and that all the passed fields are valid fields of the reports.db database, then performs the update.