Project – first part:

Project goal:

Building a Python frontend and backend stack.

Solution architecture:

Development language: Python.

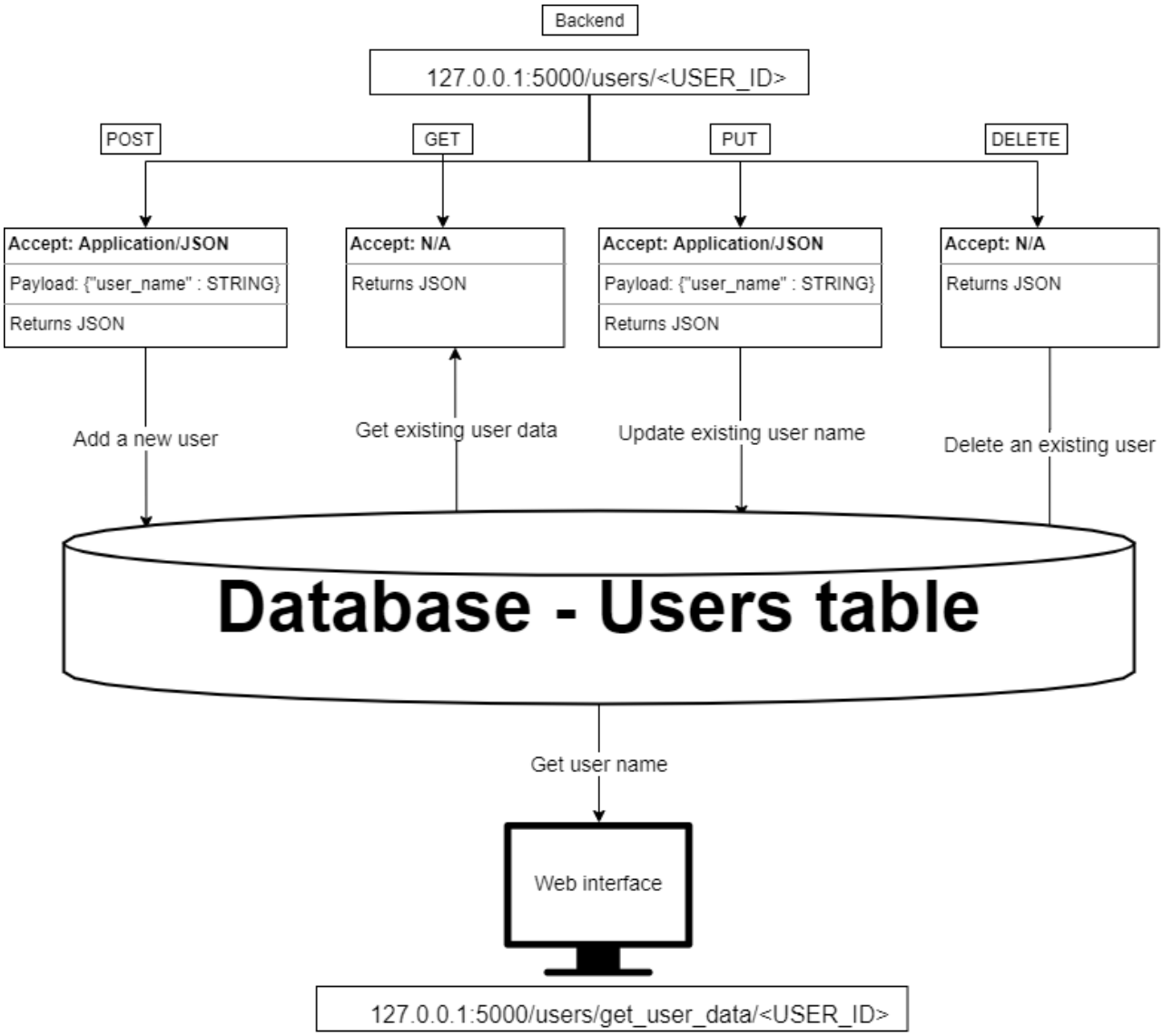
Libraries: pymysql, requests, json, flask, Selenium webdriver

Distribution type: Private.

General guidelines:

- Where necessary protect code blocks with error handling ways.
- Each method has to be documented with comments.
- Stick to the specifications document.

General diagram:



REST API (module name: rest_app.py):

The REST API gateway will be: 127.0.0.1:5000/users/<USER_ID>

1. **POST** – will accept **user_name** parameter inside the JSON payload.
   A new user will be created in the database (Please refer to **Database** section) with the id passed in the URL and with **user_name** passed in the request payload.
   ID has to be unique!

   Example: when posting the below (marked) JSON payload to 127.0.0.1:5000/users/**1**
   A new user will be created in the DB (Please refer to **Database** section) with the id **1** and the name john.
   {"**user_name**": "john"}

   On success: return JSON : {"status": "ok", "user_added": <USER_NAME>} + code: 200
   On error: return JSON : {"status": "error", "reason": "id already exists"} + code: 500

2. **GET** – returns the user name stored in the database for a given user id.
   Following the example: 127.0.0.1:5000/users/1 will return **john**.

   On success: return JSON : {"status": "ok", "user_name": <USER_NAME>} + code: 200
   On error: return JSON : {"status": "error", "reason": "no such id"} + code: 500

3. **PUT –** will modify existing user name (in the database).
   Following the above example, when posting the below JSON payload to 127.0.0.1:5000/users/1
   george will replace john under the id 1
   {"**user_name**": "george"}

   On success: return JSON : {"status": "ok", "user_updated": <USER_NAME>} + code: 200
   On error: return JSON : {"status": "error", "reason": "no such id"} + code: 500

4. **DELETE –** will delete existing user (from database).
   Following the above (marked) example, when using delete on 127.0.0.1:5000/users/1
   The user under the id 1 will be deleted.

   On success: return JSON : {"status": "ok", "user_deleted": <USER_ID>} + code: 200
   On error: return JSON : {"status": "error", "reason": "no such id"} + code: 500

Database (module name: db_connector.py):

1. Use (any) remote MySQL service.
2. The REST API (Please refer to **REST API** section) will read and write data using a MySQL table called **users:**

- **users** table will have 3 columns:
  - **user_id** – primary key, int, not null
  - **user_name** - varchar[50], not null
  - **creation_date** – varchar[50] which will store user creation date (in any format)
    For example:

user_id	user_name	creation_date
1	John	2020-08-01 13:10:36
2	Jack	2021-01-02 10:04:10

3. Table can be created manually (and not from code).

Web interface (module name: web_app.py):

The Web interface will be: 127.0.0.1:5001/users/get_user_data/<USER_ID>

1. The web interface will return the user name of a given user id stored inside users table (Please refer to **Database** section).
2. The user name of the user will be returned in an HTML format with a locator to simplify testing.
3. In case the ID doesn't exist return an error (in HTML format)
   For **example**:

```
@app.route("/get_user_name")
def get_user_name(user_id):
    user_name = get_user_name_from_db(user_id)
    return "<H1 id='user'>" + user_name + "</H1>"
```

```
@app.route("/get_user_name")
def get_user_name(user_id):
    user_name = get_user_name_from_db(user_id)
    if user_name == None:
    return "<H1 id='error'>" no such user: + user_id + "</H1>"
```

Testing:

1. Create 3 python modules for testing frontend, backend and both.
2. The modules will be able to run independently.

Frontend testing – for web interface testing (module name = frontend_testing.py):

1. Name the module frontend_testing.py
2. The script will:
    - Start a Selenium Webdriver session.
    - Navigate to web interface URL using an existing user id.
    - Check that the user name element is showing (web element exists).
    - Print user name (using locator).

Backend testing – for REST API and Database testing (module name = backend_testing.py):

1. Name the module backend_testing.py
2. The script will:
    - Post a new user data to the REST API using POST method.
    - Submit a GET request to make sure status code is 200 and data equals to the posted data.
    - Check posted data was stored inside DB (users table).

> Example:
>
> **Step 1:** **POST** the below (marked) JSON payload to 127.0.0.1:5000/users/1
> {"**user_name**": "john"}
>
> **Step 2:** Call 127.0.0.1:5000/users/1 using **GET** method and make sure the user_name "john" returned in the response and response code is 200.
>
> **Step 3:** Query (using pymysql) users table and make sure "john" is stored under id 1

Combined testing – for Web interface, REST API and Database testing
(module name = combined_testing.py):

The script will:

- Post any new user data to the REST API using **POST** method.
- Submit a **GET** request to make sure data equals to the posted data.
- Using pymysql, check posted data was stored inside DB (users table).
- Start a Selenium Webdriver session.
- Navigate to web interface URL using the new user id.
- Check that the user name is correct.

Any failure will throw an exception using the following code: **raise Exception("test failed")**

Project files:

```
.
|   backend_testing.py
|   combined_testing.py
|   db_connector.py
|   frontend_testing.py
|   rest_app.py
|   web_app.py
```

Extras

1. Read about PyDoc and use it to document your project using HTML.

2. Read about prepared statements (For MySQL) and use it for insert statement.

3. Create another table to write your users data and save the date as DATETIME (and not varchar).

4. Create another table (in DB) and call it config, the table will contain:
   - The API gateway URL (e.g: 127.0.0.1:5001/users)
   - The browser to test on (e.g: Chrome)
   - A user name to be inserted

   Use it to run your tests, meaning: instead of using "Hard-coded" URL, browser type and user name – take the data from the DB.

5. In case an ID was already taken (in POST request) create the user under another ID.
   For example: ID 1 is taken, so if we POST to this address: 127.0.0.1:5000/users/**1** according to spec, we will get an error.
   Instead of giving an error create the user under another free ID (for instance 999).

6. Read about PyPika (Python query builder) and use it for your DB implementation.