The project covers a python web programming exercise with flask and sqlalchemy

Prerequisites:

- python basics
- flask: how to make routes, get data from http requests, get and put data with sessions/cookies
- databases: how to build ORM with `SqlAlchemy` and how to use it from python code
- know how to linting code with `pylint` OR/AND `flake8`
- know how to test code with `pytest` OR `unittest`

Useful links:

- python official tutorial: https://docs.python.org/3/tutorial/ (https://docs.python.org/3/tutorial/)
- flask documentation: http://flask.pocoo.org/docs/latest (http://flask.pocoo.org/docs/latest)
- sqlalchemy documentation: http://docs.sqlalchemy.org/en/latest/ (http://docs.sqlalchemy.org/en/latest/)
- testing (common): http://pythontesting.net/start-here/ (http://pythontesting.net/start-here/)
- testing (flask specifics): http://flask.pocoo.org/docs/0.12/testing/ (http://flask.pocoo.org/docs/0.12/testing/)
- pylint documentation: https://pylint.readthedocs.io/en/latest/ (https://pylint.readthedocs.io/en/latest/)
- flake8 documentation: http://flake8.pycqa.org/en/latest/ (http://flake8.pycqa.org/en/latest/)

Common requirements:

- use python3 (3.6 is prefered)
- use virtual envs (conda, virtualenv, etc)
- use github to expose results
- use linters and write tests to cover most code
- write README with description of your work: why, for what, tech stack, implementation details
- try divide your work by git commits
- use comments to explain your code

Notes about linters

When you install and run linters

```
pylint your_module.py
```

You get some report abput quality of your code.

In case of any error/warning/etc you will look some similar output:

```
************* Module pylint.checkers.format
W: 50: Too long line (86/80)
W:108: Operator not followed by a space
     print >>sys.stderr, 'Unable to match %r', line
            ^
W:141: Too long line (81/80)
W: 74:searchall: Unreachable code
W:171:FormatChecker.process_tokens: Redefining built-in (type)
W:150:FormatChecker.process_tokens: Too many local variables (20/15)
W:150:FormatChecker.process_tokens: Too many branches (13/12)
```

Try to fix your code until you see errors.

Structure

The result directory should look like this:

```
api/
    api.py
db/
    conn.py
    model.py
tests/
    test_api.py
    test_conn.py
    test_model.py
requirements.txt
README.md
```

Minimal part

Implement the following

flask api:

- use json to send and receive data
- implement CRUD (create (POST), read (GET), update (PUT), delete (DELETE))
- POST:
  - add a server:

```
{
    "server": {
        "name": "<server_name>",
        "IP": "<ipv4>",
    }
}
```

- response:

```
{
    "status": "<CREATED|WRONG_REQUEST|NAME_ALREADY_EXIST>"
}
```

- PUT:
  - update a server IP:

```
{
    "server": {
        "name": "<server_name>",
        "IP": "<ipv4>",
    }
}
```

  - response:

```
{
    "status": "<UPDATED|WRONG_REQUEST|NAME_DID_NOT_FOUND>"
}
```

- GET:
  - get all servers (without body, just `GET` request to /)
  - response:

```
{
    "servers": [
        "<server_name>",
        "<another_server_name>",
        ...
    ]
}
```

  - get monitoring info about the specified server:

```
{
    "server": "<server_name>"
}
```

  - response:

```
{
    "server": "<server_name>",
    "pings": [
        "<datetime>": <response_ping_time_ms>,
        "<datetime>": <response_ping_time_ms>,
        ...
    ]
}
```

- DELETE:

- delete server's data in both tables:

```
{
    "server": "<server_name>"
}
```

- response:

```
{
    "status": "<DELETED|WRONG_REQUEST|NAME_DID_NOT_FOUND>"
}
```

- each server has: ID, name, IP(add some real, like 8.8.8.8), datetime(when was added)
- each record about ping request has: ID, server_id, response_time, datetime(when was added)
- so, the result db tables:
  - (server table) id/name/ip/datetime
  - (data table) id/server_id/response_time/datetime
- simple worker: just a script, loop over all records and ping each server. Check ping reponse time and a new record to the data table

Example of ping request (but need to be done from python script)

```
> ping 8.8.8.8

Pinging 8.8.8.8 with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time=35ms TTL=48
Reply from 8.8.8.8: bytes=32 time=35ms TTL=48
Reply from 8.8.8.8: bytes=32 time=35ms TTL=48
Reply from 8.8.8.8: bytes=32 time=36ms TTL=48

Ping statistics for 8.8.8.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 35ms, Maximum = 36ms, Average = 35ms
```

So, we need to take `35ms` from `Average = 35ms` and put it to the data table. In this case, the tables will look like this:

server table:

```
1, google_dns, 8.8.8.8, 2018-01-01 00:00:05
2, another_google_dns, 8.8.4.4, 2018-02-07 05:01:00
```

data table

```
1, 1, 35, 2018-01-02 07:01:01   <--  the record that should be added acco
rding to the ping response above
2, 1, 37, 2018-01-02 08:01:01
3, 1, 33, 2018-01-02 09:01:01
4, 2, 56, 2018-01-02 07:02:01
5, 2, 54, 2018-01-02 08:02:01
6, 2, 48, 2018-01-02 09:02:01
```

Advanced parts (choose any or all)

- use travis ci to run your tests automatically: https://github.com/marketplace/travis-ci (https://github.com/marketplace/travis-ci)
- implement token based authentication: https://blog.miguelgrinberg.com/post/restful-authentication-with-flask (https://blog.miguelgrinberg.com/post/restful-authentication-with-flask)
- implement front-end to log in and see ping results (see http://www.chartjs.org/ (http://www.chartjs.org/) and https://developers.google.com/web/updates/2015/03/introduction-to-fetch (https://developers.google.com/web/updates/2015/03/introduction-to-fetch))
- implement async worker to ping servers. Use asyncio: https://docs.python.org/3/library/asyncio.html (https://docs.python.org/3/library/asyncio.html), https://hackernoon.com/asyncio-for-the-working-python-developer-5c468e6e2e8e (https://hackernoon.com/asyncio-for-the-working-python-developer-5c468e6e2e8e)
- implement api using GraphQL instead. See http://graphql.org/ (http://graphql.org/), https://bcb.github.io/graphql/flask (https://bcb.github.io/graphql/flask)
- use setup.py to run tests + linters