User Microservice

User management microservice written with Flask Framework.

- Since this project is a microservice, Flask should be more than enough due to its lightweight and fast architecture.
- Flask alone is not enough for production use, so i have added gunicorn to make it more scalable.
- *healthcheck* endpoint is actually not required but services in clouder providers like **AWS-ECS** or load balancers require this healtcheck.
- In order to use this microservice **5000** and **3306** ports should be available on your local development machine.

This project assumes you had already installed these tools:

1. docker
2. docker-compose

Database Usage

You can connect the database using an ordinary mysql client. Configurations are below:

```
MYSQL_HOST = 127.0.0.1
MYSQL_PORT = 3306
MYSQL_USER = root
MYSQL_PASSWORD = root
MYSQL_DB = user_db
```

API Usage

In order to use the API, all you have to do is run docker compose on the root directory of this project.

Below are the sample usage:

```
cd <this directory>
docker-compose up -d
```

After starting the system, API will expose itself through port: `5000`

There are different endpoints for this API:

1. `/health-check`

- This endpoint accepts *GET* request.
  - This endpoint is just useful for understanding of the application live status.

2. `/register`

- This endpoint accepts *POST* request.
  - If user is not on the database; it hashes the password, inserts the user into database and returns a **token**.
  - Body Params:

```
{
"username": "arincelhan", [must]
"email": "elhanarinc@gmail.com", [must]
"password": "arinc456", [must]
```

```
"name": "arinc", [optional]
"surname": "elhan", [optional]
"age": 27 [optional]
}
```

3. `/login`

- This endpoint accepts *POST* request.
  - If user is on the database, it returns a **token**.
  - Body Params:

```
{
  "email": "dummy@gmail.com", [must]
  "password": "123456" [must]
}
```

4. `/reset-password`

- This endpoint accepts *POST* request.
  - User can reset and create a new password for his/hers account by using this endpoint.
  - `Authorization` token should be on the request header.
  - Body Params:

```
{
"password": "arinc123" [must]
}
```

5. `/delete`

- This endpoint accepts *DELETE* request.
  - User can remove himself/herself from the database by using this endpoint.
  - `Authorization` token should be on the request header.

6. `/update`

- This endpoint accepts *PUT* request.
  - User can update his/hers name, surname and age by using this endpoint.
  - `Authorization` token should be on the request header.
  - Body Params:

```
{
"name": "john", [must]
"age": 32, [must, should be greater than 15]
"surname": "doe" [must]
}
```

7. `/users`

- This endpoint accepts *GET* request.
  - User can get information about the users on the database.
  - `Authorization` token should be on the request header.
  - User can search through on `email` or `username` fields by using **search_param** query parameter.

- **search_param** can be `NULL`.
- Ex:

```
    127.0.0.1:5000/users?search_param=johndoe
    127.0.0.1:5000/users
```

Test Scenarios

login and **register** test scenarios has been covered. In order to run tests, commands below could be used:

```
cd <this directory>
cd api
virtualenv be-venv -p python3
source be-venv/bin/activate
pip install -r requirements.txt
cd src
python -m unittest discover
```