Project Name: Task Management API
Deadline: Tuesday, 19 September, 2023

Project Description: You are tasked with creating a RESTful API for a task management system. The API should allow users to perform basic CRUD (Create, Read, Update, Delete) operations on tasks.

The project is created on the Flask framework for Python web apps. It is modularized in the following files:

- App.py: contains the routes and their implementation for each of the tasks that were required to be done (e.g. create, update, delete a task).
- Auth_middleware.py: contains the authorization methods for user verification on the API routes.
- Config.py: contains database and auth configurations.
- Database.py: contains Base class and database initializer object.
- Helpers.py: contains some helper auth functions
- Models.py: contains the models that are to be created as tables in the database along with their attributes (E.g. Task, User)
- Requirements.txt: contains all the required dependencies needed to run this project.

Run the project:

(Note: You need to have Python installed on your device)

- Clone the project from github.
- Open the project in any code editor (E.g. Visual Studio Code)
- Make sure you are in the project folder
- Create a virtual environment following these steps to ensure all dependencies are installed in one environment:
    - pip install virtualenv
    - python<version> -m venv ,env-name>
    - Run env/Scripts/activate.ps1

- Now that your virtual environment is created, run the following command to install all dependencies:
    - pip freeze > requirements.txt
    - pip install -r requirements.txt
- Once all dependencies are installed in the virtual env, run the project using:
    - python app.py

CRUD APIs:

Use any API platform for running the APIs (E.g. Postman)

1. SignUp API:

Use the following API to sign up to create a new user:

http://127.0.0.1:5000/signup {Method: POST}

Request body:

```
{
  "email" : "testuser123@gmail.com",
  "password" : "12345"
}
```

Response:

- If email already exists in the database ‘Users’ table:

Status Code: 409

```
{
"success": False,
"error": "User with this email already exists"
}
```

- If new user is added successfully to the database:

StatusCode: 201

```
{
 "access_token": "<some access token value>"
}
```

- In case of an exception (E.g. database failed to add new user):

Status code: 500

```
{
"success": False,
"error": "unable to create user"
}
```

2. Login API:

Use the following API to login the user in order to get authenticated, to be able to access all routes/ APIs:
:

http://127.0.0.1:5000/login {Method: POST}

Request body:

```
{
  "email": "testuser123@gmail.com",
  "password": "12345"
}
```

Response:

- If user successfully logs in:

Status code: 200

```
{
 "access_token": "<some access token value>"
}
```

- If email is incorrect:

Status code: 404

```
{
  "error": "User not found",
  "success": false
}
```

- If password is incorrect:

Status code: 400

```
{
  "error": "incorrect password",
  "success": false
}
```

Authorization:

NOTE: Any API used without having the access token in the bearer token will return the following error:

Status code: 401

```
{
   "error": "authentication token not found"
}
```

3. Create Task API:

The access_token generated from the login API is to be passed as bearer token in the Authorization headers of the create task API. To create a task use the following request body:

To create a new task:

http://127.0.0.1:5000/ {Method: POST}

Request body:

```
{
   "title" : "TestTask",
   "description" : "Creating a task management API project",
   "due_date": "2022-09-22"
}
```

The title and due date are mandatory fields for the create task API. The status of the task is by default set to “IN_PROGRESS”. In order to pass it as COMPLETED, you need to add another field. Like so,

```
{
   "title" : "TestTask",
   "description" : "Creating a task management API project",
   "due_date": "2022-09-22",
   “status” : “COMPLETED”
}
```

Response body:

Status code: 201


```
{
  "data": {
    "description": "Creating a task management API project",
    "due_date": "2022-09-22",
    "id": 3,
    "status": "IN_PROGRESS",
    "title": "TestTask"
  },
  "success": true
}
```


- If title or due date is missing:

Status code 400:


```
{
  "error": "Title or due date missing",
  "success": false
}
```

4. Get All Tasks API:

The access_token generated from the login API is to be passed as bearer token in the Authorization headers of the get all tasks API. To get all the tasks of the database:

http://127.0.0.1:5000/ {Method: GET}

Request body:

No request body required

Response body:

Status code: 200

A list of “data” objects containing information of all tasks added in the database is returned.


```
{
  "data": [
    {
      "description": "Creating a task management API project",
      "due_date": "2022-09-22",
      "id": 1,
      "status": "IN_PROGRESS",
      "title": "TestTask"
    },
    {
      "description": "Creating a task management API project",
      "due_date": "2022-09-22",
      "id": 2,
      "status": "IN_PROGRESS",
      "title": "TestTask 1"
    },
    {
      "description": "Learning NODEJS",
      "due_date": "2022-09-22",
      "id": 3,
      "status": "IN_PROGRESS",
      "title": "TestTask 2"
    },
    {
      "description": "Learning Python",
      "due_date": "2023-09-22",
      "id": 6,
      "status": "IN_PROGRESS",
      "title": "TestTask 3"
    }
  ], "success": true
}
```

5. Get Task by ID API:

The access_token generated from the login API is to be passed as bearer token in the Authorization headers of the get all tasks API. To get all the tasks of the database:

http://127.0.0.1:5000/id {Method: GET}

REQUIRED: ID to be passed in the query path of the API.
Request body:

No request body required

Response body:

Status code: 200
A data object against the given ID will be returned.


```
{
  "data": [
    {
      "description": "Creating a task management API project",
      "due_date": "2022-09-22",
      "id": 1,
      "status": "IN_PROGRESS",
      "title": "TestTask"
    }
```

6. Update Task by ID:

The access_token generated from the login API is to be passed as bearer token in the Authorization headers of the update by ID API.

http://127.0.0.1:5000/id {Method: PUT}

Request body:

REQUIRED: ID to be passed in the query path of the API.

```
{
    "title": "Flask Project",
    "description": "Learning Python",
    "status": "COMPLETED",
    "due_date": "2023-09-20"
}
```

Note: Date format needs to be YYYY-MM-DD
None of the fields are required in the request body.

Response:

Status code: 200


```
{
    "data": {
        "description": "Learning Python",
        "due_date": "2023-09-20",
        "id": 1,
        "status": "COMPLETED",
        "title": "Flask Project"
    },
    "success": true
}
```

7. Delete Task by ID:

The access_token generated from the login API is to be passed as bearer token in the Authorization headers of the delete by ID API.

http://127.0.0.1:5000/id {Method: DELETE}

Request body:

REQUIRED: ID to be passed in the query path of the API.

No request body required

Response:

Status code: 200


```
{
  "data": {
    "description": "Learning Python",
    "due_date": "2023-09-22",
    "id": 1,
    "status": "COMPLETED",
    "title": "TestTask7"
  },
  "message": "Task successfully deleted",
  "success": true
}
```


- If ID is not found:

Status code: 404

```
<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>Task not found</p>
```

Database:

The database used for this project is “sqlite”. The SQLAlchemy extension is used to perform database operations. SQLAlchemy is an open-source Python library for working with relational databases. It provides a high-level, SQL-agnostic interface for interacting with databases.

Security:

1. On user sign up, the password is hashed before being saved in the database. The hash_password() function in helpers.py uses the becrypt() function.
   - **bcrypt.gensalt()** – It is used to generate salt. Salt is a pseudorandom string that is added to the password. Since hashing always gives the same output for the same input so if someone has access to the database, hashing can be defeated. for that salt is added at the end of the password before hashing. It doesn’t need any arguments and returns a pseudorandom string.
   - **bcrypt.hashpw()** – It is used to create the final hash which is stored in a database.

2. Authentication token is generated on Login, if the user successfully logs in with the valid credentials.
The sign_jwt() function in helpers.py takes user ID as input and generates a jwt token, with a validity of 1 hour. This token is required as an Authorization header (Bearer Token) by all the CRUD APIs mentioned above.

3. The @auth_required decorator on every CRUD route ensures that a valid bearer token is present in the Authorization header of the API.

4. Rate limiting:
The default rate limit for an API is set to "200 per day", "50 per hour".
   - Sign up: exempted from rate limiting
   - Login: 5 per minute
   - Create a new task: 2 per minute
   - Update an existing task: 2 per minute