Tasks API

Intro

In this assignment you will be tasked to build your own server that will serve as an API. You will be building a fully functional server that could, if deployed to the cloud, serve real live internet traffic and be integrated as a part of a web application.

What is Express.js?

Express.js is a Node.js web application server framework, which is specifically designed for building single-page, multi-page, and hybrid web applications.

It has become the standard server framework for Node.js. Express is the backend part of something known as the MERN stack.

The MERN is a free and open-source JavaScript software stack for building dynamic web sites and web applications which has the following components;

1. MongoDB – The standard NoSQL database
2. Express.js – The default web applications framework
3. React – The JavaScript library used for web applications
4. Node.js – Framework used for scalable server-side and networking applications.

The Express.js framework makes it very easy to develop an application which can be used to handle multiple types of requests like the GET, PUT, and POST and DELETE requests.

Resources

Refer to this great official Node/Express tutorial.

Also feel free to use online resources such as Stack Overflow, MDN, W3, and Google for reference.

A Note on Collaboration

This is a partner project! You will both be keeping up with your own versions of the code, but should work through this assignment together with constant communication.

Setup

1. Pull the starter code
   - `git pull starter main`
2. Install dependencies
   - `cd tasks-api`
   - `npm install`

3. Start the Node server
    - `npm start`
4. Install pytest
    - `pip install -U pytest`

Your Task

You will be building a server that can keep track of tasks. Your server must be able to do the following:

1. Create a new task with a title property and a boolean determining whether the task has been completed. A new unique id would be created for each new task
2. List all tasks created
3. Get a specific task
4. Delete a specified task
5. Edit the title or completion of a specific task
6. (Bonus) Bulk add multiple tasks in one request
7. (Bonus) Bulk delete multiple tasks in one request

Your application will accept JSON and/or URL parameters and will return JSON data. Your server would be ready to be automatically integrated in a web system.

Endpoints

Here is a specific list of endpoints that you will be required to create, along with the method and the inputs/outputs:

Create a new Task:

```
POST /v1/tasks

Input (Body):
{
    title: "Test Task 2"
}

Output (201 code):
{
    id: 2
}
```

The id returned is a unique id for the todo that was just created.

List all tasks created:

```
GET /v1/tasks

Input (Body): None

Output (200 code):
{
    tasks: [
        {
            id: 1,
            title: "Test Task 1",
            is_completed: true
        },
        {
            id: 2,
            title: "Test Task 2",
            is_completed: false
        }
   ]
}
```

This endpoint list all tasks including their id's.

Get a specific task:

```
GET /v1/tasks/{id}

Input (URL): id

Output (200 code):
{
    id: 3,
    title: "Test Task 2",
    is_completed: false
}

On Error (ID not found - 404):
{
    error: "There is no task at that id"
}
```

This endpoint returns a specific task or returns a 404 not found response.

Delete a specific task:

```
DELETE /v1/tasks/{id}

Input (URL): id
```

```
Output (204 code): None
```

This endpoint deletes a specific task. If the task doesn’t exist still send the same response.

Edit the title or completion of a specific task

```
PUT /v1/tasks/{id}

Input (Body):
{
    title: "Test Task 2",
    is_completed: false
}

Output (204 code): None

On Error (ID not found - 404):
{
    error: "There is no task at that id"
}
```

This endpoint edits a specific task's title, is_completed, or both. If the id is not found, it returns a 404 not found response.

(Bonus) Bulk add tasks

```
POST /v1/tasks

Input (Body):
{
    tasks: [
        {title: "Test Task 1", is_completed: true},
        {title: "Test Task 2", is_completed: false},
        {title: "Test Task 3", is_completed: true}
    ]
}

Output (201 code):
{
    tasks: [
        {id: 1},
        {id: 2},
        {id: 3}
    ]
}
```

This endpoint bulk adds more than one task. Note that this feature uses the same endpoint as the single task creation endpoint, but it should support both input data formats.

(Bonus) Bulk delete tasks

```
DELETE /v1/tasks

Input (Body):
{
    tasks: [
        {id: 1},
        {id: 2},
        {id: 3}
    ]
}

Output (204 code): None

On Error (ID not found - 404):
{
    error: "There is no task at that id"
}
```

This endpoint bulk deletes more than one task.

Tests

To see if you are on the right track, run the below file (using `pytest`, while your server is running) to test your endpoints. This is not an extensive test.

This test is to ensure that you have all of the routes correct and that your response is properly formed. For example, ensuring that your `GET /v1/tasks/{id}` endpoint returns a dictionary with an `id` (a `number`), a `title` (a `string`), and whether the task is `completed` (`boolean`).

Note: please run this as the first thing that hits your server on boot up. If it successfully completes once then your assignment is on the right track. Re-running the tests could fail as written.

So in short:

- Start your webserver
- Run your tests once (`npm test`)
- If pass you are good!