Assignment – I (10%)

COMP 3123 – Full Stack Development – I

Submission: Sunday, 13th Oct 2024, 11:59 PM (Week06)

- ## Instructions:

You are required to implement the following RESTful API endpoints. Ensure that each endpoint performs the correct operation and returns the specified response code.

- ## Objectives:

1. Understand RESTful API design principles.
2. Implement CRUD operations using provided API endpoints.
3. Test the API endpoints to ensure they return the correct response codes and data.

As a newly hired Jr. Software Engineer my manager assigned me a task to develop Backend application using NodeJS, Express and MongoDB. He also wants me to apply my VCS (**GitHub**) skills to develop these projects where I will commit and push all my code to **Student#_COMP3123_Assignment1** repository.

- ## API Endpoints to Implement:

Following is the list of APIs to develop which accept all data as JSON Object whenever needed:

Sr. #	Method	Endpoint	Response Code	Description
User Management:				
1	POST	/api/v1/user/signup	201	Allow user to create new account
2	POST	/api/v1/user/login	200	Allow user to access the system
Employee Management:				
3	GET	/api/v1/emp/employees	200	User can get all employee list
4	POST	/api/v1/emp/employees	201	User can create new employee

5	GET	/api/v1/emp/employees/{eid}	200	User can get employee details by employee id
6	PUT	/api/v1/emp/employees/{eid}	200	User can update employee details
7	DELETE	/api/v1/emp/employees?eid=xxx	204	User can delete employee by employee id

- **MongoDB Database name: comp3123_assigment1**

Users Collection Schema

```
{
    "_id": ObjectId,
    "username": String,
    "email": String,
    "password": String, // This should be hashed
    "created_at": Date,
    "updated_at": Date
}
```

User can login using username/email and password

Employee Collection Schema:

```
{
    "_id": ObjectId,
    "first_name": String,
    "last_name": String,
    "email": String,
    "position": String,
    "salary": Number,
    "date_of_joining": Date,
    "department": String,
    "created_at": Date,
    "updated_at": Date
}
```

- **Sample Input and Output:**

API	Sample Input	Sample Output

POST /api/v1/user/signup	{
	"username": "johndoe",
	"email": "johndoe@example.com",
	"password": "password123"
	}
		{
		"message": "User created successfully.",
		"user_id": "64c9e5a3d9f3c1a5c9b4e8a1"
		}

POST /api/v1/user/login	{
	"email": "johndoe@example.com",
	"password": "password123"
	}
		{
		"message": "Login successful.",
		"jwt_token": "Optional implementation"
		}

GET /api/v1/emp/employees		[{ "employee_id": "64c9e5a3d9f3c1a5c9b4e8a2", "first_name": "Jane", "last_name": "Doe", "email": "jane.doe@example.com", "position": "Software Engineer", "salary": 90000, "date_of_joining": "2023-08-01T00:00:00.000Z", "department": "Engineering" }, {

		"employee_id": "64c9e5a3d9f3c1a5c9b4e8a3", "first_name": "John", "last_name": "Smith", "email": "john.smith@example.com", "position": "Product Manager", "salary": 110000, "date_of_joining": "2023-07-15T00:00:00.000Z", "department": "Product" }]

POST /api/v1/emp/employees	{
	"first_name": "Alice",
	"last_name": "Johnson",
	"email": "alice.johnson@example.com",
	"position": "Designer",
	"salary": 85000,
	
	"date_of_joining": "2023-08-10T00:00:00.000Z",
	"department": "Design"
	}
		{
		"message": "Employee created successfully.",
		"employee_id": "64c9e5a3d9f3c1a5c9b4e8a4"
		}

GET /api/v1/emp/employees/64c9e5a3d9f3c1a5c9b4e8a4		{

		"employee_id": "64c9e5a3d9f3c1a5c9b4e8a4", "first_name": "Alice", "last_name": "Johnson", "email": "alice.johnson@example.com", "position": "Designer", "salary": 85000, "date_of_joining": "2023-08-10T00:00:00.000Z", "department": "Design" }

PUT
/api/v1/emp/employees/{eid}
	{
	"position": "Senior Designer",
	"salary": 95000
	}
		{
		"message": "Employee details updated successfully."
		}

DELETE
/api/v1/emp/employees?eid=64c9e5a3d9f3c1a5c9b4e8a4
		{
		"message": "Employee deleted successfully."
		}

- **Testing & Validation**

1. **Validation**:
    - Use libraries like **express-validator** to validate incoming requests.
2. **Testing with Postman:**
    - Test all API endpoints and save the Postman collection.
    - Capture screenshots of each test.

Sample Error Response:

```
{
    "status": false,
    "message": "Invalid Username and password"
}
```

- **Notes:**

- Make use of **express.Routes()** and **modules**
- Validate the data whenever required
- Return error details or success response details whenever required
- All data must be sent back and forth in JSON Object format
- Optionally apply JWT security concept to secure all your API calls
- No late submission

- **Submission Checklist**

1. **MongoDB Console Screenshots:**
    - Provide screenshots of your MongoDB database showing collections.
2. **Postman API Collection:**
    - Export your Postman collection and include it in your submission.
3. **Screenshots:**
    - Include screenshots of each API being tested along with responses.
4. **Project ZIP File:**
    - Remove the **node_modules** folder and create a ZIP file of your project.
5. **GitHub Project Link:**
    - Provide the link to your GitHub repository.
6. **Sample User Detail:**
    - Include a sample user's details for testing login.
7. **Comments:**
    - Add any comments or notes that could be helpful.
8. **Hosting:**
    - If hosted, provide the URL to your deployed application on platforms like Vercel, Heroku, Render, etc.

- **Evaluations:** **Refer detailed rubric on Brightspace assignment**

Points	Evaluation Component
70	5 points for each 7 working API without database connection
	10 points for each 7 working API with database connection
10	10 Point for sending validation and error messages to client
10	10 point maintaining GitHub repository with read me file
10	Uploading valid screenshots with appropriate name and documentation

- **Communication:**
  - Please contact on pritesh.patel2@georgebrown.ca or SLACK channel for any question or query.
  - NO communication before 3 days for submission deadline

~~~ Wish you all the Best ~~~