ASSIGNMENT 3: API CRUD WITH SPECIFICATION

Create a new project and upload the source code to a **new** GitHub repository. Build a backend API using Node.js on port 4000. You can use either Express or NestJS to meet the project requirements. Deploy your backend API to cloud provider like Render.com

R1: Create API endpoint return your information with your full name and your student code.

Description:

* This API retrieves user information in the form of a JSON object.

Request:

* Method: GET
* URL: <http://localhost:4000/info>

Response Body:

{

"data": {

"fullName": "Nguyen Van A",

"studentCode": "QNUO1234"

}

}

Response Details:

* data: The main object containing user information.
* name: A string representing the user’s name (e.g., "Nguyen Van A").
* code: A string representing the user’s unique code (e.g., "HELO1234").

R2: Build an API for managing a student list with MongoDB, where each student is represented as a document. The data will be stored in a MongoDB collection, where each student document has the following fields::

* \_id: Unique identifier for the student (auto-generated by MongoDB).
* fullName: Name of the student.
* studentCode: Unique student code.
* isActive: Boolean indicating whether the student is active (true) or graduated (false).

Here’s a REST API specification for managing students using MongoDB for data storage:

Base URL:

* http://localhost:4000

Endpoints Overview:

1. Create a Student

* Method: POST
* Endpoint: /students
* Request Body:

{

"name": "John Doe",

"studentCode": "ST12345",

"isActive": true

}

* Response:
  + 201 Created
  + Response Body:

{

"success": true,

"message": "Student created successfully",

"data": {

"\_id": "60a6c72bf25b4e1f88d81a44",

"name": "John Doe",

"studentCode": "ST12345",

"isActive": true

}

}

* + 400 Bad Request (if validation fails)

2. Get All Students

* Method: GET
* Endpoint: /students
* Response:
  + 200 OK
  + Response Body:

{

"success": true,

"data": [

{

"\_id": "60a6c72bf25b4e1f88d81a44",

"name": "John Doe",

"studentCode": "ST12345",

"isActive": true

},

{

"\_id": "60a6c72bf25b4e1f88d81a55",

"name": "Jane Smith",

"studentCode": "ST67890",

"isActive": false

}

]

}

* + 500 Internal Server Error (if something goes wrong with the server)

3. Get a Student by ID

* Method: GET
* Endpoint: /students/:id
* Response:
  + 200 OK
  + Response Body:

{

"success": true,

"data": {

"\_id": "60a6c72bf25b4e1f88d81a44",

"name": "John Doe",

"studentCode": "ST12345",

"isActive": true

}

}

* + 404 Not Found (if student ID does not exist)

4. Update a Student

* Method: PUT
* Endpoint: /students/:id
* Request Body:

{

"name": "John Doe Updated",

"isActive": false

}

* Response:
  + 200 OK
  + Response Body:

{

"success": true,

"message": "Student updated successfully",

"data": {

"\_id": "60a6c72bf25b4e1f88d81a44",

"name": "John Doe Updated",

"studentCode": "ST12345",

"isActive": false

}

}

* + 400 Bad Request (if validation fails)
  + 404 Not Found (if student ID does not exist)

5. Delete a Student

* Method: DELETE
* Endpoint: /students/:id
* Response:
  + 200 OK
  + Response Body:

{

"success": true,

"message": "Student deleted successfully"

}

* + 404 Not Found (if student ID does not exist)

Error Responses:

* 400 Bad Request: For invalid requests or validation errors.
  + Example:

{

"success": false,

"message": "Invalid student code format"

}

* 500 Internal Server Error: For any unexpected server errors.
  + Example:

{

"success": false,

"message": "Something went wrong on the server"

}

Notes:

* Make sure to validate incoming data, such as ensuring the studentCode is unique and isActive is a boolean.
* The MongoDB schema would be used to store student data as shown in the previous message. This specification supports basic CRUD functionality for managing a student list.

Submit file studentCode\_assignment\_3.doc contains link to github repository and url of deployed backend API.