**CHARACTER XYZ BOOTCAMP**

**API DEVELOPMENT**

**WORKING PROCEDURE:**

**Working of nest js:**

* Nest js works with modules.
* The modules are the classes with ‘@module’ decorator which makes use of the organized application structure.
* The module in turn calls the controller which handles the incoming and outgoing requests and responses.
* The services are the main business logic behind the responses.
* The additional optional helper files provide methods or classes to the logic in the services.
* The services are included into the controllers in form of dependency injections so the services are annotated with ‘@injectable’ decorator.
* The included services are used in form of a service object to refer to all the methods in the services and then use them to efficiently handle requests and provide them with proper logic.

**Nest js application for student data mode:**

* The nest js application accesses the database using several tools. Here in this project the TypeOrm module is used for the connection between the database (MySql).
* The entity file is the one which defines the schema of the data-model used in the database.
* The entities form the base for the student objects defined in the nest js code .
* DTO(Data Transfer Object) is used to refer to the fields in the body of the api request or the returned response.

**Student Records CRUD API Description**

**Student Records CRUD API**

Student Records CRUD API is the api to do all the basic(create ,update ,read ,delete) in the database.

Here the database is student database and the entities have the following fields:

·        name

·        age

·        department

·        gpa

·        registrationDate

·        createdAt

·        modifiedAt

The following are the api requests for the CRUD operations:

**POST (CREATE):**

The create request is used to make a new student entity and enter it into the database.

The body of the request is a createStudentDto object and the response is also a createStudentDto object .

The body of the json can’t be empty if it is empty it returns bad request status code.

Create new student record:

http://localhost:3000/student/CREATE

The create request is done as below

**Example:**

**Request:**

curl --location 'http://localhost:3000/student/CREATE'

--data

{

    "name":"anirudh",

    "department":"cse",

    "gpa":9.8,

    "registration\_date":"2020-10-10",

    "age":19,

    "attendance":100

}

**Response:**

{

    "status": "201 Created",

    "stud": {

        "name": "adarsh",

        "department": "cse",

        "age": 20,

        "gpa": 9,

        "attendance": 80,

        "createdAt": "2023-03-16T15:34:31.863Z",

        "modifiedAt": "2023-03-16T15:34:31.863Z"

    }

}

**GET(read all):**

The read request without id is used to return all the data in db as it is.

The returned response is of type Student array as it has multiple student entities.

Read all the students

The below request is used to fetch all the student details

**Example:**

**Request:**

curl --location 'http://localhost:3000/student/READ'

**Response:**

{

    "status": "200 OK",

    "type": {

        "contentType": "application/json"

    },

    "allStudents": [

        {

            "id": "8",

            "name": "umadevi",

            "department": "cse",

            "age": 20,

            "gpa": 9.6,

            "registrationDate": "2021-10-09T18:30:00.000Z",

            "attendance": 90,

            "createdAt": "2023-03-16T07:11:50.000Z",

            "modifiedAt": "2023-03-16T07:11:50.000Z"

        },

        {

            "id": "9",

            "name": "anirudh",

            "department": "cse",

            "age": 19,

            "gpa": 9,

            "registrationDate": "2022-10-09T18:30:00.000Z",

            "attendance": 100,

            "createdAt": "2023-03-16T07:13:57.000Z",

            "modifiedAt": "2023-03-16T07:13:57.000Z"

        },

        {

            "id": "10",

            "name": "rajesh",

            "department": "cse",

            "age": 21,

            "gpa": 9.3,

            "registrationDate": "2020-10-09T18:30:00.000Z",

            "attendance": 90,

            "createdAt": "2023-03-16T07:14:18.000Z",

            "modifiedAt": "2023-03-16T07:14:18.000Z"

        }

    ]

}

**GET(read by id):**

 The read by id request is used for fetching student entities based on the student id.

The response is of student type.

The response is generated only based on the id if it is found.

Read student detail by id:

http://localhost:3000/student/READ/

**Example:**

**Request:**

curl --location 'http://localhost:3000/student/READ/14'

**response:**

{

    "status": "200 found",

    "type": {

        "contentType": "application/json"

    },

    "students": {

        "id": "8",

        "name": "umadevi",

        "department": "cse",

        "age": 20,

        "gpa": 9.6,

        "registrationDate": "2021-10-09T18:30:00.000Z",

        "attendance": 90,

        "createdAt": "2023-03-16T07:11:50.000Z",

        "modifiedAt": "2023-03-16T07:11:50.000Z"

    }

}

**GET(read sorted registration\_date):**

In the request all the student entities are fetched based on the registration date.

The sorting of students is done using the merge sort algorithm on the student entity array returned.

Read students sorted based on registration date

The above request is used to fetch the sorted array of students based on the registration date

The below request is used to retrieve details based on id

**Example:**

**Response:**

curl --location 'http://localhost:3000/student/registration\_date'

**Response:**

{

    "status": "200 found",

    "type": {

        "contentType": "application/json"

    },

    "sortedStudents": [

        {

            "id": "14",

            "name": "anirudh",

            "department": "cse",

            "age": 19,

            "gpa": 9.8,

            "registrationDate": null,

            "attendance": 100,

            "createdAt": "2023-03-16T15:46:13.000Z",

            "modifiedAt": "2023-03-16T15:46:13.000Z"

        },

        {

            "id": "13",

            "name": "uma",

            "department": "cse",

            "age": 19,

            "gpa": 9.5,

            "registrationDate": null,

            "attendance": 90,

            "createdAt": "2023-03-16T15:45:59.000Z",

            "modifiedAt": "2023-03-16T15:45:59.000Z"

        },

        {

            "id": "12",

            "name": "adarsh",

            "department": "cse",

            "age": 20,

            "gpa": 9,

            "registrationDate": null,

            "attendance": 80,

            "createdAt": "2023-03-16T15:45:34.000Z",

            "modifiedAt": "2023-03-16T15:45:34.000Z"

        }

    ]

}

**GET(read by weighted\_score):**

The request here sorts all the entities in the DB and the sorting is done on the basis of weighted score.

The return type is a Student array object containing all the sorted entities.

Read student’s based on weighted score

The above request is used to fetch the descending sorted array of students based on the weighted average

weighted average = 0.7\*gpa + 0.3\*attendance

**Example:**

**Request:**

curl --location 'http://localhost:3000/student/weighted\_score'

**Response:**

{

    "status": "200 found",

    "type": {

        "contentType": "application/json"

    },

    "sortedStudents": [

        {

            "id": "14",

            "name": "anirudh",

            "department": "cse",

            "age": 19,

            "gpa": 9.8,

            "registrationDate": null,

            "attendance": 100,

            "createdAt": "2023-03-16T15:46:13.000Z",

            "modifiedAt": "2023-03-16T15:46:13.000Z"

        },

        {

            "id": "13",

            "name": "uma",

            "department": "cse",

            "age": 19,

            "gpa": 9.5,

            "registrationDate": null,

            "attendance": 90,

            "createdAt": "2023-03-16T15:45:59.000Z",

            "modifiedAt": "2023-03-16T15:45:59.000Z"

        },

        {

            "id": "12",

            "name": "adarsh",

            "department": "cse",

            "age": 20,

            "gpa": 9,

            "registrationDate": null,

            "attendance": 80,

            "createdAt": "2023-03-16T15:45:34.000Z",

            "modifiedAt": "2023-03-16T15:45:34.000Z"

        }

    ]

}

**PATCH(UPDATE):**

The update is used to update/modify the fields of a particular entity.

The body of the request is of type updateStudentDto which can’t be empty.

The response is the student object of the student entity with updated values.

Update student records

http://localhost:3000/student/UPDATE/

The below request is used to update the records based on the id

**Example:**

**Request:**

curl --location --request PATCH 'http://localhost:3000/student/UPDATE/13' \

--data

{

    "gpa":9.3

}

**Response:**

{

    "status": "200 OK",

    "type": {

        "contentType": "application/json"

    },

    "stud": {

        "id": "8",

        "name": "umadevi",

        "department": "cse",

        "age": 20,

        "gpa": 10,

        "registrationDate": "2021-10-09T18:30:00.000Z",

        "attendance": 90,

        "createdAt": "2023-03-16T07:11:50.000Z",

        "modifiedAt": "2023-03-16T07:52:37.000Z"

    }

}

**DELETE(DELETE operation):**

The delete request is used to permanently remove the records in DB based on the student id.

If the given id is not found then the response is ‘no response found’.

If the given id is found it deletes the occurrence and the updates are saved.

Delete student records

http://localhost:3000/student/DELETE/

the above request is used to delete student records based on id

**Example:**

**Request:**

curl --location --request DELETE 'http://localhost:3000/student/DELETE/12'

**Response**

{

    "status": "204 no content",

    "deleteResponse": 1

}

**ERROR HANDLING:**

**POST(empty post body):**

Create errors

This is the empty body error for create

**Example:**

**Request:**

curl --location 'http://localhost:3000/student/CREATE'

--data '{}'

**Response:**

400 Bad request

**GET(id not found):**

Read student id not found error

The below response is returned when id is not found

**Example:**

**Response:**

curl --location 'http://localhost:3000/student/READ/8'

**Request:**

{

    "statusCode": 400,

    "message": "Record not found"

}

**PATCH(id not found):**

Update record not found

http://localhost:3000/student/UPDATE/

The below is the response when the update id not found

**Example:**

**Request:**

curl --location --request PATCH 'http://localhost:3000/student/UPDATE/8'

**Response:**

{

    "statusCode": 403,

    "message": "no such row found"

}

**DELETE(id not found):**

Delete record not found

http://localhost:3000/student/DELETE/

The below is returned when the id is not found for delete request

**Example:**

**Request:**

curl --location --request DELETE 'http://localhost:3000/student/DELETE/9'

**Response:**

404 not found

**TOOLS USED:**

**Nest CLI Tools:**

* **Generate:**
  + Used to create files specific to the application.
  + nest g controller/module/service nameOfFile.
* **Run:**
  + Npm run start:dev / npm run start

Used to start the nest js application .

The :dev is used to see all the logged changes in the terminal.

**POSTMAN:**

* Used to test out an api without front end
* The several requests can be selected using the postman tool.
* Creating a collection helps in effective documentation.

Link for postman generated documentation:

https://documenter.getpostman.com/view/26303552/2s93Jxrgiv