**CLASS APIS**

| **API Name** | **Method** | **URL** | **Description** |
| --- | --- | --- | --- |
| Get All Classes | GET | /classes | Fetch all classes with teacher and student list. |
| Get Classes with Pagination | GET | /classes/paginate | Fetch classes with pagination and sorting. |
| Get Class by ID | GET | /classes/:id | Fetch a specific class by ID. |
| Create New Class | POST | /classes | Add a new class with details. |
| Update Class by ID | PUT | /classes/:id | Update class details by ID. |
| Delete Class by ID | DELETE | /classes/:id | Delete a class by ID. |

Testing the below methods in Postman involves sending HTTP requests to the server with appropriate routes, headers, body, and parameters. Here's a guide:

**1. Get All Classes (GET /)**

* **Method**: GET
* **URL**: http://localhost:<port>/classes
* **Steps**:
  1. Open Postman and select GET.
  2. Enter the URL above (replace <port> with your server's port).
  3. Add query parameters (optional):
     + Click Params and add key-value pairs (e.g., year: 2023).
  4. Click Send.
  5. View the list of classes in the response.

**2. Get a Class by ID (GET /:id)**

* **Method**: GET
* **URL**: http://localhost:<port>/classes/<classId>
* **Steps**:
  1. Replace <classId> with the actual ID of the class in the database.
  2. Click Send.
  3. View the specific class details in the response.

**3. Create a New Class (POST /)**

* **Method**: POST
* **URL**: http://localhost:<port>/classes
* **Steps**:
  1. Select POST.
  2. Go to the Body tab, select raw, and set Content-Type to JSON.
  3. Enter a JSON object:

{

"name": "Class A",

"year": 2024,

"teacher": "<teacherId>",

"studentFees": 500,

"studentList": ["<studentId1>", "<studentId2>"]

}

* 1. Click Send.
  2. View the created class in the response.

**4. Update a Class (PUT /:id)**

* **Method**: PUT
* **URL**: http://localhost:<port>/classes/<classId>
* **Steps**:
  1. Replace <classId> with the ID of the class to update.
  2. Go to the Body tab, select raw, and set Content-Type to JSON.
  3. Enter the updated details:

{

"name": "Updated Class A",

"year": 2025,

"teacher": "<newTeacherId>",

"studentFees": 600,

"studentList": ["<newStudentId1>", "<newStudentId2>"]

}

* 1. Click Send.
  2. View the updated class in the response.

**5. Delete a Class (DELETE /:id)**

* **Method**: DELETE
* **URL**: http://localhost:<port>/classes/<classId>
* **Steps**:
  1. Replace <classId> with the ID of the class to delete.
  2. Click Send.
  3. Verify the success message in the response:

{ "message": "Class deleted successfully" }

For Pagination API

**Query Parameters**

Add query parameters in the **Params** tab in Postman to control pagination and sorting:

| **Parameter** | **Description** | **Example Value** |
| --- | --- | --- |
| page | The page number to retrieve (default: 1) | 1, 2 |
| limit | The number of items per page (default: 10) | 5, 10 |
| sortBy | The field to sort results by (default: name) | name, year |
| sortOrder | The order of sorting (asc or desc) | asc, desc |

For example:

* **Key**: page | **Value**: 2
* **Key**: limit | **Value**: 5
* **Key**: sortBy | **Value**: year
* **Key**: sortOrder | **Value**: desc

**Sending the Request**

1. Open Postman.
2. Select GET as the request method.
3. Enter the URL: http://localhost:<port>/classes/paginate.
4. Click on the **Params** tab and add query parameters as shown above.
5. Click **Send**.

**Verifying the Response**

Check the response for:

1. **Pagination Details**:
   * total: Total number of classes in the database.
   * page: The current page number.
   * limit: Number of items per page.
   * totalPages: Total pages calculated based on total and limit.
2. **Sorted and Paginated Results**:
   * The classes array should reflect the sorting order and pagination.

Example Response:

{

"total": 25,

"page": 2,

"limit": 5,

"totalPages": 5,

"classes": [

{

"\_id": "64e73a1b72c911d9ad5c0b01",

"name": "Class B",

"year": 2024,

"teacher": { "name": "Mr. Smith" },

"studentFees": 500,

"studentList": [{ "\_id": "64e73a1b72c911d9ad5c0b02", "name": "John Doe" }]

},

...

]

}

**Example Scenarios**

* Retrieve the first page with 10 items sorted by year in ascending order:
  + **GET /classes/paginate?page=1&limit=10&sortBy=year&sortOrder=asc**
* Retrieve the third page with 5 items sorted by name in descending order:
  + **GET /classes/paginate?page=3&limit=5&sortBy=name&sortOrder=desc**

**Tips:**

* If the response is empty, verify that there are enough records in the database.
* Test edge cases:
  + Request beyond the total number of pages.
  + Use invalid query parameters.

**TEACHER APIS**

Here’s a concise summary of the API endpoints for the Teacher model:

| **API Name** | **Method** | **URL** | **Description** |
| --- | --- | --- | --- |
| Get All Teachers | GET | /teachers | Fetch all teachers with their assigned classes. |
| Search and Paginate | GET | /teachers/search | Search teachers by name, filter by gender, and paginate results. |
| Create New Teacher | POST | /teachers | Add a new teacher with details. |
| Update Teacher by ID | PUT | /teachers/:id | Update teacher details by ID. |
| Delete Teacher by ID | DELETE | /teachers/:id | Delete a teacher by ID. |

**1. Get All Teachers (GET /)**

**Steps:**

1. Open Postman and create a new request.
2. Set the request method to GET.
3. Enter the URL: http://<your\_server>:<port>/teachers/.
4. Click Send.

**Expected Response:**

* Status: 200 OK
* Response Body: List of all teachers.

**2. Get Teachers with Pagination and Filtering (GET /search)**

**Steps:**

1. Set the request method to GET.
2. Enter the URL: http://<your\_server>:<port>/teachers/search?page=1&limit=5&gender=Male&name=John.
   * Replace the query parameters (page, limit, gender, and name) as needed.
3. Click Send.

**Expected Response:**

* Status: 200 OK
* Response Body:

{

"data": [ /\* array of teachers \*/ ],

"total": 50,

"totalPages": 10,

"currentPage": 1

}

**3. Create a New Teacher (POST /)**

**Steps:**

1. Set the request method to POST.
2. Enter the URL: http://<your\_server>:<port>/teachers/.
3. Go to the Body tab and select raw.
4. Set the type to JSON.
5. Enter the following sample JSON:

{

"name": "John Doe",

"gender": "Male",

"dob": "1990-05-14",

"contact": "1234567890",

"salary": 50000,

"assignedClass": "<Class ObjectId>"

}

1. Click Send.

**Expected Response:**

* Status: 201 Created
* Response Body: Newly created teacher object.

**4. Update a Teacher (PUT /:id)**

**Steps:**

1. Set the request method to PUT.
2. Enter the URL: http://<your\_server>:<port>/teachers/<teacher\_id>.
   * Replace <teacher\_id> with the actual ID of the teacher to update.
3. Go to the Body tab and select raw.
4. Set the type to JSON.
5. Enter the fields you want to update, e.g.:

{

"salary": 60000,

"contact": "9876543210"

}

1. Click Send.

**Expected Response:**

* Status: 200 OK
* Response Body: Updated teacher object.

**5. Delete a Teacher (DELETE /:id)**

**Steps:**

1. Set the request method to DELETE.
2. Enter the URL: http://<your\_server>:<port>/teachers/<teacher\_id>.
   * Replace <teacher\_id> with the actual ID of the teacher to delete.
3. Click Send.

**Expected Response:**

* Status: 200 OK
* Response Body:

{

"message": "Teacher deleted successfully"

}

**STUDENT APIS**

Here is a concise table summarizing the API endpoints:

| **API Name** | **Method** | **URL** | **Description** |
| --- | --- | --- | --- |
| Get All Students | GET | /students | Fetch all students with class details. |
| Get Student by ID | GET | /students/:id | Fetch a single student by ID. |
| Create New Student | POST | /students | Add a new student. |
| Update Student by ID | PUT | /students/:id | Update student details by ID. |
| Delete Student by ID | DELETE | /students/:id | Delete a student by ID. |
| Search Students | GET | /students/search | Search, filter, sort, and paginate data. |

**1. Get All Students**

* **Method**: GET
* **URL**: http://localhost:<PORT>/students
  + Replace <PORT> with your server's port number.
* **Description**: Fetches all students with their class details.

**2. Get a Student by ID**

* **Method**: GET
* **URL**: http://localhost:<PORT>/students/<id>
  + Replace <id> with the ID of the student.
* **Description**: Fetches a student by their ID.

**3. Create a New Student**

* **Method**: POST
* **URL**: http://localhost:<PORT>/students
* **Headers**:
  + Content-Type: application/json
* **Body**:

{

"name": "John Doe",

"gender": "Male",

"dob": "2000-01-01",

"contact": "1234567890",

"feesPaid": true,

"class": "64a1234e5bc1234567890abc"

}

* **Description**: Adds a new student to the database.

**4. Update a Student by ID**

* **Method**: PUT
* **URL**: http://localhost:<PORT>/students/<id>
  + Replace <id> with the ID of the student.
* **Headers**:
  + Content-Type: application/json
* **Body**:

{

"name": "Jane Doe",

"contact": "0987654321"

}

* **Description**: Updates details of the specified student.

**5. Delete a Student by ID**

* **Method**: DELETE
* **URL**: http://localhost:<PORT>/students/<id>
  + Replace <id> with the ID of the student.
* **Description**: Deletes a student by their ID.

**6. Search, Filter, Sort, and Paginate Students**

* **Method**: GET
* **URL**: http://localhost:<PORT>/students/search
* **Query Parameters**:
  + gender: Gender of the students (e.g., Male).
  + feesPaid: Whether the fees are paid (true or false).
  + class: Class ID (e.g., 64a1234e5bc1234567890abc).
  + sortBy: Field to sort by (e.g., name).
  + order: Sort order (asc or desc).
  + page: Page number (default: 1).
  + limit: Number of results per page (default: 10).
* **Example URL**:

http://localhost:<PORT>/students/search?gender=Male&feesPaid=true&sortBy=name&order=asc&page=1&limit=5

* **Description**: Filters, sorts, and paginates students based on query parameters.