

Task 1: School Management System CRUD Operations and API Development

use postman to test each CRUD operation:

1-create:

The screenshot shows the Postman interface. The URL is `http://localhost/school/crud.php`. The method is set to `POST`. The request body is a JSON object:

```
1 {
2   "action": "create",
3   "name": "John Doe",
4   "date_of_birth": "1990-01-01",
5   "address": "123 Main St",
6   "contact_information": "john.doe@example.com"
7 }
```

The response status is `200 OK`, time `38 ms`, and size `298 B`. The response body is:

```
1 {"message": "Student created successfully"}
```

The screenshot shows the phpMyAdmin interface connected to the `school_management` database. The current table is `students`. The table structure is:

	id	name	date_of_birth	address	contact_information
Edit	1	John Doe	1990-01-01	123 Main St	john.doe@example.com

Update:

The screenshot shows the Postman application interface. In the top navigation bar, 'My Workspace' is selected. On the left sidebar, there are sections for 'Collections', 'Environments', and 'History'. The main workspace displays an API endpoint at `http://localhost/school/crud.php`. A POST request is being prepared with the URL `http://localhost/school/crud.php`. The 'Body' tab is selected, showing raw JSON data:

```
1 {
2   "action": "update",
3   "id": 1,
4   "name": "Jane Doe",
5   "date_of_birth": "1990-01-01",
6   "address": "123 Main St",
7   "contact_information": "janedoe@example.com"
8 }
```

Below the body, the response status is 200 OK with 53 ms and 298 B. The response body is displayed as:

```
1 {"message": "Student updated successfully"}
```

The screenshot shows the phpMyAdmin interface connected to a MySQL server at 127.0.0.1:3306. The database selected is 'school_management' and the table is 'students'. The page title is 'Showing rows 0 - 3 (total, Query took 0.0006 seconds.)'. The SQL query shown is:

```
SELECT * FROM `students`
```

Below the query, there is a table with the following data:

	Edit	Copy	Delete	1 Jane Doe	1990-01-01	123 Main St	janedoe@example.com
	Edit	Copy	Delete	2 sarah Doe	1990-01-01	123 Main St	john doe@example.com
	Edit	Copy	Delete	3 kinda Doe	1990-01-01	123 Main St	john doe@example.com
	Edit	Copy	Delete	4 jack Doe	1990-01-01	123 Main St	john doe@example.com

delete:

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'API Documentation #reference', 'Data Visualization', and 'RESTful API Basics #blueprint'. The main area shows a POST request to 'http://localhost/school/crud.php'. The 'Body' tab is selected, showing raw JSON input:

```
1 {
2   "action": "delete",
3   "id": 1
4 }
```

Below the request, the response status is 200 OK with a message: "Student deleted successfully".

The screenshot shows the phpMyAdmin interface. The left sidebar lists databases and tables, including 'school', 'school_management', and 'students'. The main panel shows the 'students' table in the 'school_management' database. The table has columns: id, name, date_of_birth, address, and contact_information. There are three rows of data:

	id	name	date_of_birth	address	contact_information
<input type="checkbox"/>	2	sarah Doe	1990-01-01	123 Main St	johndoe@example.com
<input type="checkbox"/>	3	kinda Doe	1990-01-01	123 Main St	johndoe@example.com
<input type="checkbox"/>	4	jack Doe	1990-01-01	123 Main St	johndoe@example.com

Task2:

1. 1. Create Student (POST):

The screenshot shows the Postman interface. On the left, the 'My Workspace' sidebar lists 'Collections', 'Environments', and 'History'. In the center, a 'POST' request is made to 'http://localhost/school/api.php/students'. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2   "name": "mark Doe",
3   "date_of_birth": "2000-01-01",
4   "address": "123 Main St",
5   "contact_information": "john.doe@example.com",
6   "class": "10th"
7 }
8 }
```

The response status is 200 OK, time 56 ms, size 386 B. Below the body, there is a preview of the JSON response:

```
1 {
2   "id": 8,
3   "name": "mark Doe",
4   "date_of_birth": "2000-01-01",
5   "address": "123 Main St",
6   "contact_information": "john.doe@example.com",
7   "class": "10th"
8 }
```

The screenshot shows a table of student records. The columns are: ID, Name, Date of Birth, Address, Contact Information, and Class. There are three rows:

ID	Name	Date of Birth	Address	Contact Information	Class
5	John Doe	2000-01-01	123 Main St	john.doe@example.com	10th
8	mark Doe	2000-01-01	123 Main St	john.doe@example.com	10th
9	mary Doe	1990-01-01	123 Main St	john.doe@example.com	11th

At the bottom, there are buttons for 'Edit', 'Copy', 'Delete', 'Check all', 'With selected:', 'Edit', 'Copy', 'Delete', and 'Export'. There are also filters for 'Show all', 'Number of rows: 25', 'Filter rows: Search this table', and 'Sort by key: None'.

2. Update using put

The screenshot shows the Postman application interface. In the top navigation bar, 'Home' and 'Workspaces' are selected. A search bar at the top right contains the text 'Search Postman'. The main workspace is titled 'My Workspace' and contains three items: 'API Documentation #reference', 'Data Visualization', and 'RESTful API Basics #blueprint'. On the left sidebar, there are sections for 'Collections', 'Environments', and 'History'. The central area displays a request card for 'http://localhost/school/api.php/students/5' with a 'PUT' method. The 'Body' tab is selected, showing a raw JSON payload:

```

1 {
2   "address": "456 Elm St"
3 }
4

```

Below the request card, the response details are shown: Status: 200 OK, Time: 31 ms, Size: 358 B. The response body is also displayed in JSON format:

```

1 {
2   "id": 5,
3   "name": "",
4   "date_of_birth": "0000-00-00",
5   "address": "456 Elm St",
6   "contact_information": "",
7   "class": ""
8 }

```

At the bottom of the screen, the Windows taskbar is visible with various pinned icons.

The screenshot shows the phpMyAdmin interface. The left sidebar displays a tree view of databases and tables, including 'advanced_task', 'bookstore', 'companydb', 'ecommerce', 'information_schema', 'library', 'mysql', 'my_data', 'performance_schema', 'phpapi', 'phpmyadmin', 'rest_api_demo', 'school', 'school2', 'smartbuy', 'students', 'test', 'user_admin', and 'zain_db'. The 'school' database is currently selected. Inside 'school', the 'school_management' database is selected, and the 'students' table is viewed. The table has the following columns: id, name, date_of_birth, address, contact_information, and class. There are 6 rows of data:

	id	name	date_of_birth	address	contact_information	class
<input type="checkbox"/>	2	sarah Doe	1990-01-01	123 Main St	john.doe@example.com	
<input type="checkbox"/>	3	kinda Doe	1990-01-01	123 Main St	john.doe@example.com	
<input type="checkbox"/>	4	jack Doe	1990-01-01	123 Main St	john.doe@example.com	10th
<input type="checkbox"/>	5	John Doe	2000-01-01	456 Elm St	john.doe@example.com	10th
<input type="checkbox"/>	8	mark Doe	2000-01-01	123 Main St	john.doe@example.com	11th
<input type="checkbox"/>	9	mary Doe	1990-01-01	123 Main St	john.doe@example.com	11th

At the bottom of the screen, the Windows taskbar is visible with various pinned icons.

3. 3. Delete Student (DELETE):

The screenshot shows the Postman application interface. In the center, there is a request card for a DELETE operation to the URL `http://localhost/school/api.php/students/4`. The 'Body' tab is selected, showing a JSON response with the message:

```
1 {  
2   "message": "Student deleted successfully"  
3 }
```

. Below the response, the status is listed as `Status: 200 OK Time: 29 ms Size: 288 B`. The bottom status bar indicates the system is online and shows the date and time as 7/27/2024 4:56 AM.

The screenshot shows the phpMyAdmin interface. The left sidebar displays a tree view of databases and tables, including 'school', 'school_management', and 'students'. The main panel shows the 'students' table in the 'school_management' database. A green header bar indicates 'Showing rows 0 - 4 (5 total). Query took 0.0004 seconds.' Below this, a SQL query is shown: `SELECT * FROM `students``. The table data is as follows:

	Edit	Copy	Delete	2	sarah Doe	1990-01-01	123 Main St	johndoe@example.com	class
	Edit	Copy	Delete	3	kinda Doe	1990-01-01	123 Main St	johndoe@example.com	10th
	Edit	Copy	Delete	5	John Doe	2000-01-01	456 Elm St	john doe@example.com	10th
	Edit	Copy	Delete	8	mark Doe	2000-01-01	123 Main St	john doe@example.com	10th
	Edit	Copy	Delete	9	mary Doe	1990-01-01	123 Main St	john doe@example.com	11th

At the bottom, there are buttons for Print, Copy to clipboard, Export, Display chart, and Create view.

Task3: create tables and Relationships:

phpMyAdmin

Server: 127.0.0.1:3306 Database: school_management

Structure SQL Search Query Export Operations Privileges Routines Events Triggers Designer

Run SQL query/queries on database school_management:

```
1 CREATE TABLE Teacher (
2     TeacherID INT AUTO_INCREMENT PRIMARY KEY,
3     Name VARCHAR(100) NOT NULL,
4     ContactInfo VARCHAR(255)
5 );
6
```

Clear Format Get auto-saved query Bind parameters

Delimiter : Show this query here again Retain query box Rollback when finished Enable foreign key checks Go Hide query box

MvSQL returned an empty result set (i.e. zero rows). (Query took 0.0267 seconds.)

Console

5:50 AM 7/27/2024

This screenshot shows the phpMyAdmin interface for the 'school_management' database. The left sidebar lists various databases and tables. In the main area, an SQL query is being run to create a 'Teacher' table with columns for TeacherID (primary key), Name, and ContactInfo. The results indicate an empty result set.

phpMyAdmin

Server: 127.0.0.1:3306 Database: school_management

Structure SQL Search Query Export Operations Privileges Routines Events Triggers Designer

Run SQL query/queries on database school_management:

```
1 CREATE TABLE subjects (
2     id INT PRIMARY KEY AUTO_INCREMENT,
3     name VARCHAR(100) NOT NULL,
4     description VARCHAR(200) NOT NULL
5 );
6
7 CREATE TABLE exams (
8     id INT PRIMARY KEY AUTO_INCREMENT,
9     subject_id INT NOT NULL,
10    date DATE NOT NULL,
11    max_score INT NOT NULL
12 );
13
14 CREATE TABLE student_subject (
15    student_id INT NOT NULL,
16    subject_id INT NOT NULL,
17    PRIMARY KEY (student_id, subject_id)
18 );
19
```

Clear Format Get auto-saved query Bind parameters

Delimiter : Show this query here again Retain query box Rollback when finished Enable foreign key checks Go Hide query box

Console

5:54 AM 7/27/2024

This screenshot shows the phpMyAdmin interface for the 'school_management' database. The left sidebar lists various databases and tables. In the main area, three tables are being created: 'subjects', 'exams', and 'student_subject'. The 'subjects' table has columns for id (primary key, auto-increment), name, and description. The 'exams' table has columns for id (primary key, auto-increment), subject_id, date, and max_score. The 'student_subject' table has columns for student_id and subject_id, with a primary key on both.

Create API Endpoints

1. Create Teacher (POST):

The screenshot shows the Postman application interface. In the left sidebar, under 'My Workspace', there are collections like 'API Documentation #reference', 'Data Visualization', and 'RESTful API Basics #blueprint'. The main workspace shows a POST request to 'http://localhost/school/task3.php'. The request body is set to raw JSON:

```
1 {
2   "name": "John Doe",
3   "contactInfo": "johndoe@example.com"
4 }
```

The response status is 200 OK, with the message: "Teacher created successfully".

The screenshot shows the phpMyAdmin interface connected to 'Server 127.0.0.1:3306'. The database selected is 'school_management'. The 'teacher' table is currently selected. The table structure includes columns: TeacherID, Name, and ContactInfo. One row is visible: TeacherID 4, Name John Doe, and ContactInfo johndoe@example.com.

2. Get All Teachers (GET):

The screenshot shows the Postman application interface. In the left sidebar, there's a 'My Workspace' section with collections like 'API Documentation #reference', 'Data Visualization', and 'RESTful API Basics #blueprint'. The main area displays a GET request to `http://localhost/school/task3.php`. The 'Body' tab is selected, showing a JSON response with four teacher objects:

```
[{"TeacherID": "4", "Name": "John Doe", "ContactInfo": "johndoe@example.com"}, {"TeacherID": "5", "Name": "Jane Jack", "ContactInfo": "johndoe@example.com"}, {"TeacherID": "6", "Name": "saizah jack", "ContactInfo": "johndoe@example.com"}, {"TeacherID": "7", "Name": "mary jack", "ContactInfo": "johndoe@example.com"}]
```

The status bar at the bottom indicates the request was successful with a 200 OK status, took 30 ms, and had a size of 721 B.

3. Get Teacher by ID (GET):

This screenshot shows a similar setup in Postman, but the URL in the header is `http://localhost/school/task3.php?TeacherID=4`. The response body now contains a single teacher object:

```
{ "TeacherID": "4", "Name": "John Doe", "ContactInfo": "johndoe@example.com"}
```

The status bar at the bottom indicates the request was successful with a 200 OK status, took 33 ms, and had a size of 431 B.

4. Create Subject (POST):

The screenshot shows the Postman application interface. In the top navigation bar, 'Home' and 'Workspaces' are selected. A search bar at the top right contains the text 'Search Postman'. On the left sidebar, under 'My Workspace', there are sections for 'Collections' (with 'API Documentation #reference', 'Data Visualization', and 'RESTful API Basics #blueprint'), 'Environments', and 'History'. The main workspace shows a POST request to 'http://localhost/school/task3.php'. The 'Body' tab is selected, showing the following JSON payload:

```
1
2 {
3     "name": "Mathematics",
4     "description": "A basic math course"
5 }
6
```

Below the body, the response status is 'Status: 200 OK' with a time of '49 ms' and a size of '351 B'. The response body is displayed in 'Pretty' format:

```
1 <...>
2
3
4
5
6 // create_subject.php
7
8
9 {"message": "Subject created successfully"}
```

The screenshot shows the phpMyAdmin interface. The left sidebar displays a tree view of databases and tables. Under the 'school' database, the 'subjects' table is selected. The main area shows the results of a SELECT query: 'Showing rows 0 - 0 (1 total, Query took 0.0004 seconds)'. The SQL query is: 'SELECT * FROM `subjects`'. The results table has columns: id, name, and description. One row is shown: id=1, name='Mathematics', description='A basic math course'. At the bottom, there are buttons for Print, Copy to clipboard, Export, Display chart, and Create view.

5. Get All Subjects (GET)

The screenshot shows the Postman application interface. In the top navigation bar, 'My Workspace' is selected. On the left sidebar, there are sections for 'Collections', 'Environments', and 'History'. The main workspace displays a request to 'http://localhost/school/task3.php' using the 'GET' method. The 'Body' tab is selected, showing the following JSON response:

```
1 <!--
2
3
4
5
6 // create_subject.php
7 //5_ get all subjects
8 [
9   {
10     "id": "1",
11     "name": "Mathematics",
12     "description": "A basic math course"
13   },
14   {
15     "id": "2",
16     "name": "English",
17     "description": "A basic English course"
18   }
19 ]
```

The status bar at the bottom right indicates the response was successful (200 OK), took 29 ms, and had a size of 463 B.

6. Get Subject by ID (GET):

The screenshot shows the Postman application interface. In the top navigation bar, 'My Workspace' is selected. On the left sidebar, there are sections for 'Collections', 'Environments', and 'History'. The main workspace displays a request to 'http://localhost/school/task3.php?id=1' using the 'GET' method. The 'Body' tab is selected, showing the following JSON response:

```
1 <!--
2
3
4
5
6 // create_subject.php
7 //5_ get all subjects
8 //6-get subjects by id
9
10 // get_subject.php
11 [
12   {
13     "id": 1,
14     "name": "Mathematics",
15     "description": "A basic math course"
16   }
17 ]<br />
18 <b>Warning</b>: Undefined variable $subjects in <b>C:\xampp\htdocs\school\task3.php</b> on line <b>298</b><br />
19 null
```

The status bar at the bottom right indicates the response was successful (200 OK), took 14 ms, and had a size of 564 B.

7. Create Exam (POST):

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost/school/task3.php`. The method is `POST`. The request body is:

```
POST / http://localhost/school/task3.php
{
  "subject_id": 1,
  "date": "2024-08-01",
  "max_score": 100
}
```

The response status is `200 OK`, time `34 ms`, and size `430 B`. The response body is:

```
7
8
9
10
11
12
13 //5- get all subjects
14
15
16 6-get subjects by id
17
18
19
20 <!-- 7-create exam(post) -->
21
22 {"message": "Exam created successfully"}
```

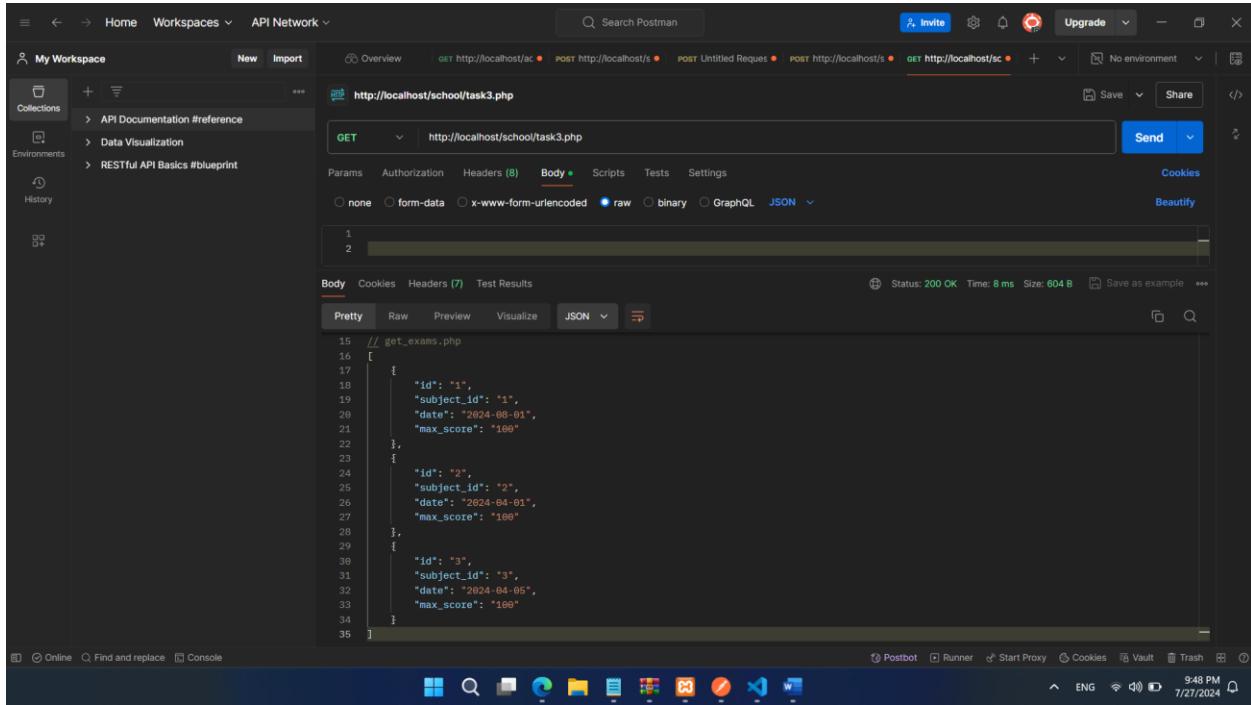
The screenshot shows the phpMyAdmin interface connected to the database `school_management`. The table is `exams`. A new row has been inserted with the following data:

	Edit	Copy	Delete	Export
1				

The query used to insert the data is:

```
SELECT * FROM `exams`
```

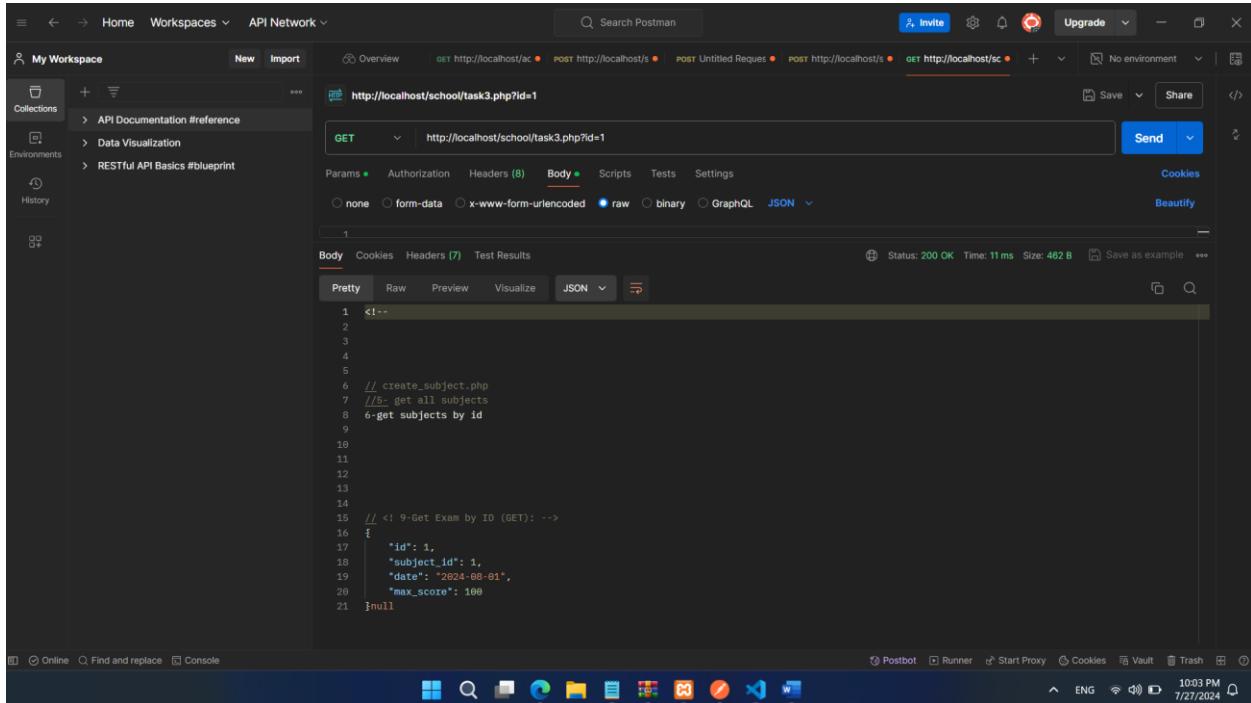
8. Get All Exams (GET):



The screenshot shows the Postman application interface. A GET request is made to `http://localhost/school/task3.php`. The response body is displayed in JSON format:

```
15 // get_exams.php
16 [
17   {
18     "id": "1",
19     "subject_id": "1",
20     "date": "2024-08-01",
21     "max_score": "100"
22   },
23   {
24     "id": "2",
25     "subject_id": "2",
26     "date": "2024-04-01",
27     "max_score": "100"
28   },
29   {
30     "id": "3",
31     "subject_id": "3",
32     "date": "2024-04-05",
33     "max_score": "100"
34   }
35 ]
```

9. Get Exam by ID (GET):



The screenshot shows the Postman application interface. A GET request is made to `http://localhost/school/task3.php?id=1`. The response body is displayed in JSON format:

```
1 <!--
2
3
4
5
6 // create_subject.php
7 //<-- get all subjects
8 6-get subjects by id
9
10
11
12
13
14
15 // <!-- 9-Get Exam by ID (GET): -->
16 {
17   "id": 1,
18   "subject_id": 1,
19   "date": "2024-08-01",
20   "max_score": 100
21 }null
```

Task 4: Make these new APIs based on our customer requests

1. Retrieve a Student's Subjects (GET): Create an endpoint to get all the subjects associated with a specific student.

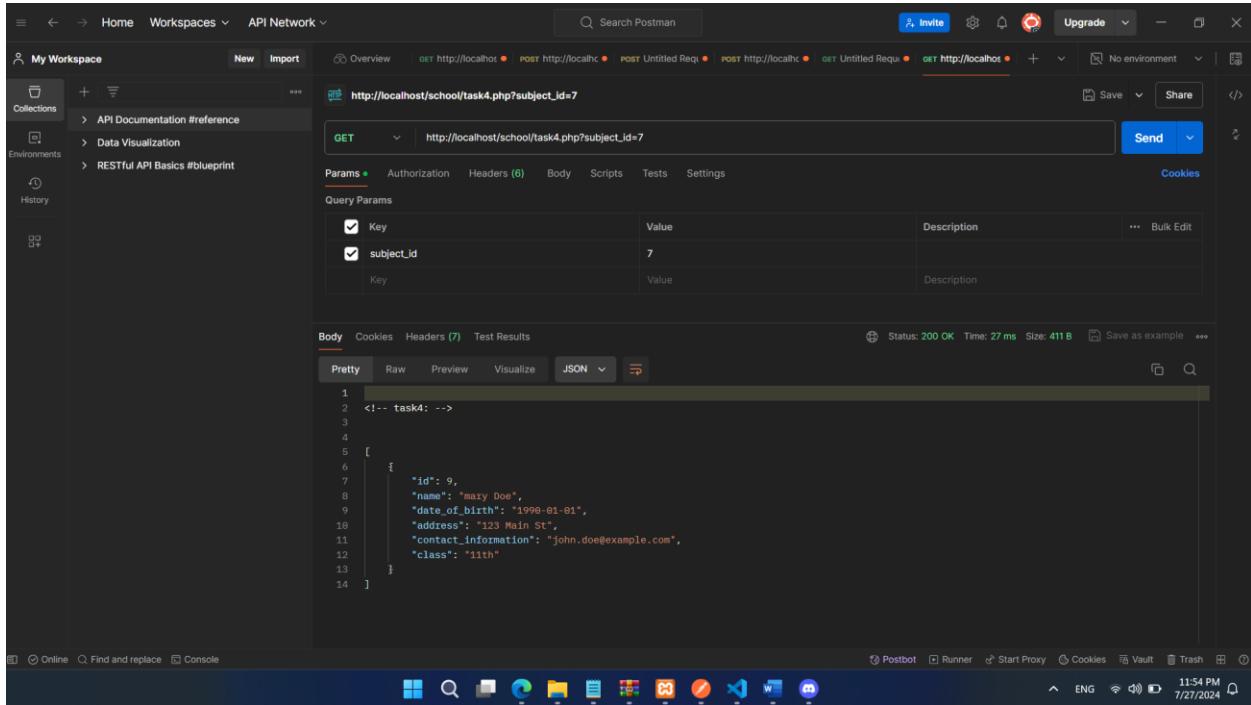
The screenshot shows the phpMyAdmin interface connected to a MySQL database named 'school_management'. The 'subjects' table is selected. The table has three columns: 'id', 'name', and 'description'. Two rows are present: one for Mathematics ('A basic math course') and one for English ('A basic English course'). The interface includes a sidebar with a tree view of databases and tables, and a bottom taskbar with various icons.

id	name	description
1	Mathematics	A basic math course
2	English	A basic English course

The screenshot shows the Postman application interface. A collection named 'My Workspace' is selected. A POST request is being prepared to the URL `http://localhost/school/task4.php?student_id=2`. The 'Params' tab is active, showing a parameter named 'student_id' with the value '2'. The 'Body' tab displays the JSON response received from the server, which contains a single subject object for English.

```
1
2 <!-- task4: -->
3 [
4   {
5     "id": 2,
6     "name": "English",
7     "description": "A basic English course"
8   }
9 ]
```

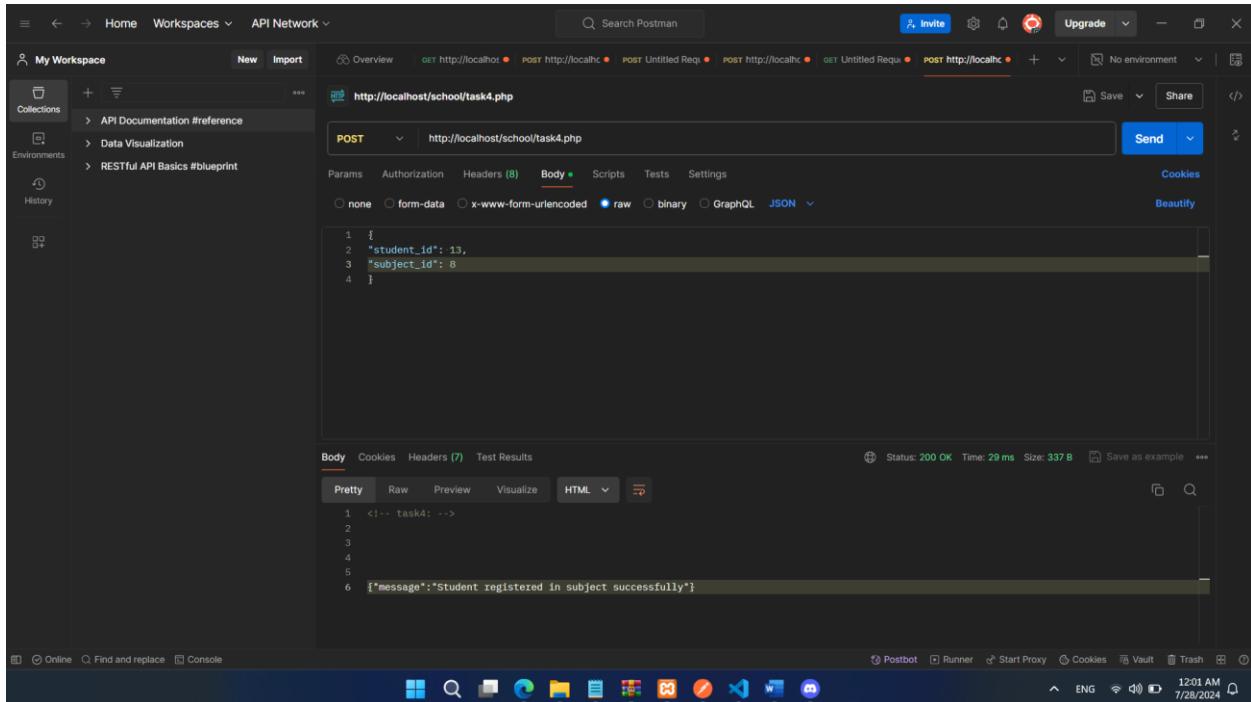
2. Retrieve a Subject's Students (GET): Create an endpoint to get all the students enrolled in a particular subject.



The screenshot shows the Postman interface with a GET request to `http://localhost/school/task4.php?subject_id=7`. The response status is 200 OK, and the response body is:

```
1<!-- task4: -->
2
3
4
5[
6    {
7        "id": 9,
8        "name": "mary Doe",
9        "date_of_birth": "1990-01-01",
10       "address": "123 Main St",
11       "contact_information": "john.doe@example.com",
12       "class": "1ith"
13    }
14]
```

3. Register Students in Subjects (POST): Implement an API endpoint to register students in subjects (many-to-many relationship).



The screenshot shows the Postman interface with a POST request to `http://localhost/school/task4.php`. The response status is 200 OK, and the response body is:

```
1<!-- task4: -->
2
3
4
5
6  {"message": "Student registered in subject successfully"}
```

The screenshot shows the phpMyAdmin interface for a database named 'school_management'. The left sidebar lists various databases and tables, including 'student_subject' which is currently selected. The main area displays the contents of the 'student_subject' table with the following data:

	student_id	subject_id
1	2	2
2	3	6
3	5	3
4	9	7
5	13	8

Below the table, there are buttons for 'Edit', 'Copy', 'Delete', 'Check all', and 'With selected:'. The status bar at the bottom right shows the date and time as 7/28/2024 12:01 AM.

4. Retrieve a Student's Exams (GET): Create an endpoint to get all the exams taken by a specific student.

The screenshot shows the Postman application interface. A collection named 'RESTful API Basics #blueprint' is selected. A specific request is shown for the URL `http://localhost/school/task4.php?student_id=15`. The 'Body' tab is selected, showing the raw JSON response:

```
1<!-- task4: -->
2
3
4
5
6
7
8[
9    {
10        "id": 1,
11        "subject_id": 1,
12        "date": "2024-08-01",
13        "max_score": 100
14    }
15]
```

The status bar at the bottom right shows the date and time as 7/28/2024 12:42 AM.

The screenshot shows the phpMyAdmin interface connected to a MySQL server at 127.0.0.1:3306. The database selected is 'school_management'. The current table is 'student_exam'. The data grid displays two rows:

student_id	exam_id
13	2
15	1

Below the table, there are buttons for Edit, Copy, Delete, and Export. The status bar at the bottom right shows the time as 12:46 AM and the date as 7/28/2024.

5. Retrieve a Subject's Exams (GET): Create an endpoint to get all the exams conducted for a particular subject.

The screenshot shows the Postman application interface. A GET request is made to the URL `http://localhost/school/task4.php?subject_id=1`. The response body is displayed in JSON format:

```

1
2 <!-- task4: -->
3
4
5
6
7
8
9 [
10   {
11     "id": 1,
12     "subject_id": 1,
13     "date": "2024-08-01",
14     "max_score": 100
15   }
16 ]

```

The status bar at the bottom right shows the time as 12:51 AM and the date as 7/28/2024.

6. Update Exam: Allow updating exam details, including scores

The screenshot shows the Postman application interface. In the top navigation bar, 'Overview' is selected. Below it, a PUT request is made to `http://localhost/school/task4.php`. The 'Body' tab is active, showing the JSON payload:

```
1 {
2   "id": 1,
3   "subject_id": 1,
4   "date": "2024-06-15",
5   "max_score": 95
6 }
7
```

Below the body, the response status is `200 OK`, time is `38 ms`, and size is `330 B`. The response body contains the message: `{"message": "Exam updated successfully"}`.

The screenshot shows the phpMyAdmin interface connected to the 'school_management' database. The 'exams' table is selected. The table structure is shown with columns: id, subject_id, date, and max_score. There are three rows of data:

	id	subject_id	date	max_score
<input type="checkbox"/>	1	1	2024-06-15	95
<input type="checkbox"/>	2	2	2024-04-01	100
<input type="checkbox"/>	3	3	2024-04-05	100