English Login

Building and Consuming a RESTful API in Laravel PHP

Blog / ... / Community / **Building and Consuming a RESTful API in...**

Building and Consuming a RESTful API in Laravel PHP

Tags

PHP

Developer insights

Code, tutorials, and best practices

Start for free

Time to read: 12 minutes

June 25, 2019

Written by

Michael Okoh Contributor ?

This post was written by a third-party contributor as a part of our Twilio Voices program. It is no longer actively maintained. Please be aware that some information may be outdated.

From your favorite social networks, down to your favorite banking applications, our modern world is driven by a lot of [APIs](). In this article, you will learn how to build a modern RESTful API and an application that will implement the API.

Prerequisites

- PHP 7.1 or Higher
- Composer
- MySql
- Laravel 5.6 or Higher
- Postman

To follow along with this tutorial, you should have a basic understanding of the PHP language. Basic knowledge of the Laravel framework is required.

Understanding our Application

You will be building a CRUD API. CRUD means Create, Read, Update, and Delete. Our API will have the following endpoints:

- `GET /api/students` will return all students and will be accepting `GET` requests.

- `GET /api/students/{id}` will return a student record by referencing its `id` and will be accepting `GET` requests.
- `POST /api/students` will create a new student record and will be accepting `POST` requests.
- `PUT /api/students/{id}` will update an existing student record by referencing its `id` and will be accepting `PUT` requests.
- `DELETE /api/students/{id}` will delete a student record by referencing its `id` and will be accepting `DELETE` requests.

The Student record will only contain `name` and `course` as details. When you are done developing these endpoints you will use the endpoints to develop an actual student records application that will make use of the API.

Setup the Laravel Application

To get started, you have to create a Laravel application. To do this you have to run the following command in your terminal:

Bash

```bash
laravel new api-project
```

Next, change your current directory to the root folder of the project:

Bash

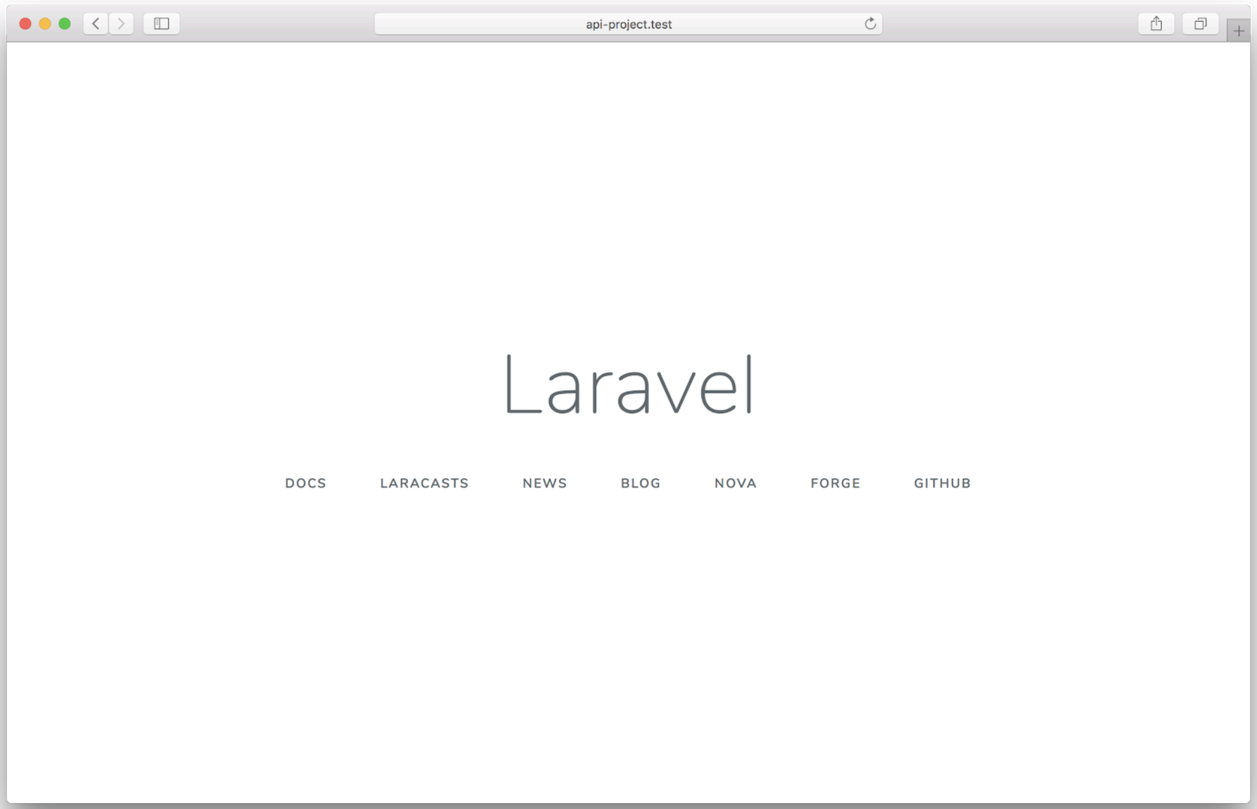
```bash
cd api-project
```

Next, start up the Laravel server if it’s not already running:

Bash

```
php artisan serve
```

You will be able to visit your application on https://localhost:8000.



Next, create a new database for your application by running:

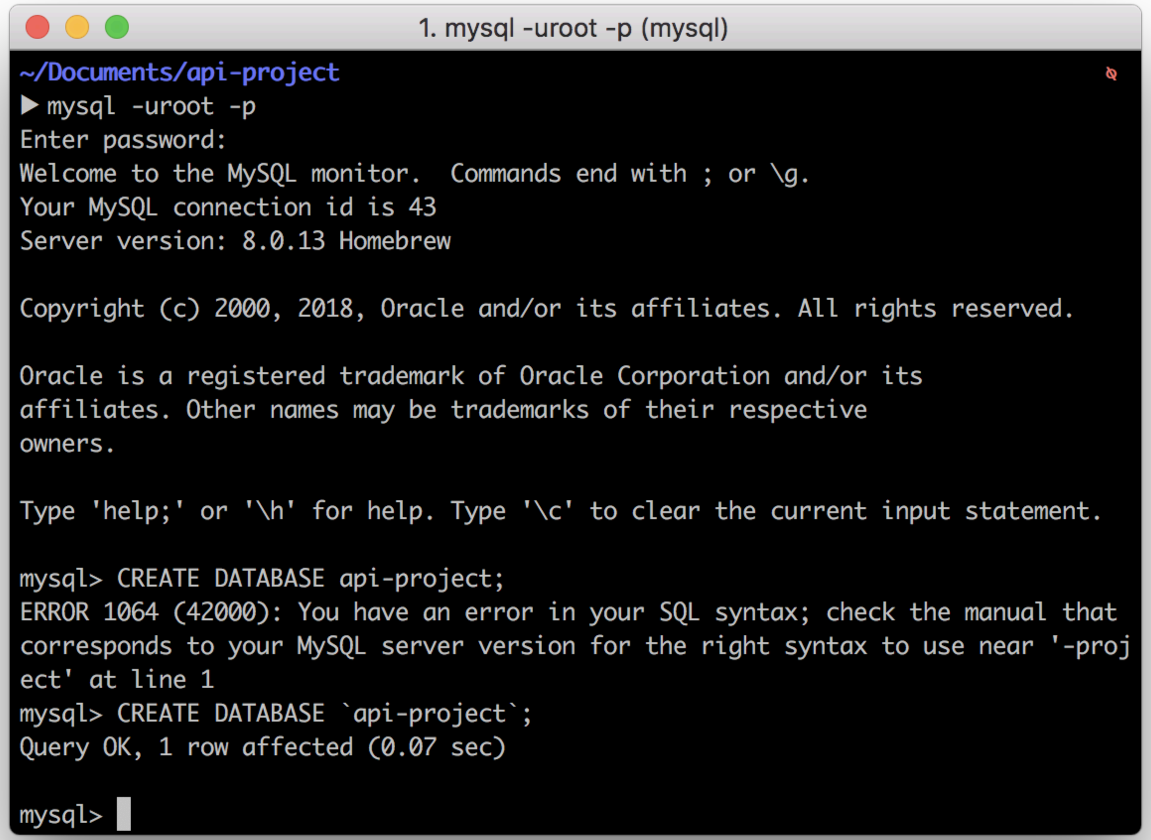
Bash

```
mysql -uroot -p
```

You will be prompted to type your MySQL password if you have any set when you authenticate with MySQL. Run the following to create a new database named `api-project`:

SQL

```sql
CREATE DATABASE `api-project`;
```



```
1. mysql -uroot -p (mysql)
~/Documents/api-project
▶ mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 43
Server version: 8.0.13 Homebrew

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE api-project;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near '-proj
ect' at line 1
mysql> CREATE DATABASE `api-project`;
Query OK, 1 row affected (0.07 sec)

mysql>
```

We can proceed to create a model along with a migration. To do this you have to run:

Bash

```bash
php artisan make:model Student -m
```

A new file named *Student.php* will be created in the *app* directory.

You will have to edit the file to specify the database table we will like to interact with and the fields that can be written to:

PHP

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Student extends Model
{
    protected $table = 'students';

    protected $fillable = ['name', 'course'];
}
```

Additionally, a migration file will be created in the *database/migrations* directory to generate our table. You will have to modify the migration file to create a column for `name` and `course` which will accept string values.

PHP

```php
// ...
public function up()
{
    Schema::create('students', function (Blueprint $table) {
        $table->increments('id');
        $table->string('name');
        $table->string('course');

```

```
            $table->timestamps();
        });
    }
// ...
```

Next, you can open the project folder in your preferred text editor and modify the *.env* file to input your proper database credentials. This will allow the application to properly connect to the recently created database:

Text

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=<your-database-name>
DB_USERNAME=<your-database-username>
DB_PASSWORD=<your-database-password>
```

Next, you will run your migration using the following command:

Bash

```
php artisan migrate
```

Setup the Routes

Now that we have the basics of the application set up, we can proceed to create a controller that will contain the methods for our API by running:

Bash

```
php artisan make:controller ApiController
```

You will find a new file named *ApiController.php* in the *app*\Http\controllers directory. Next, we can add the following methods:

PHP

```php
class ApiController extends Controller
{
    public function getAllStudents() {
      // logic to get all students goes here
    }

    public function createStudent(Request $request) {
      // logic to create a student record goes here
    }

    public function getStudent($id) {
      // logic to get a student record goes here
    }

    public function updateStudent(Request $request, $id) {
      // logic to update a student record goes here
    }

    public function deleteStudent ($id) {
      // logic to delete a student record goes here
    }
}
```

Proceed to the *routes* directory and open the *api.php* file and create the endpoints that will reference the methods created earlier in the `ApiController`.

PHP

```php
// ...
Route::get('students', 'ApiController@getAllStudents');
Route::get('students/{id}', 'ApiController@getStudent');
Route::post('students, 'ApiController@createStudent');
Route::put('students/{id}', 'ApiController@updateStudent');
Route::delete('students/{id}','ApiController@deleteStudent');
```

All routes in *api.php* are prefixed with "/api" by default.

Create a Student Record

Locate the `createStudent` method in our `ApiController`.

PHP

```php
public function createStudent(Request $request) {
  // logic to create a student record goes here
}
```

We will be using the Laravel request class to fetch the data passed to the endpoint. The endpoint will be expecting `name` of type `string` and `course` of type `string` as well. When we have successfully fetched the data we will store the data in our database.

PHP

```php
// ...
use App\Student;

class ApiController extends Controller
```

```php
{
  // ...
  public function createStudent(Request $request) {
    $student = new Student;
    $student->name = $request->name;
    $student->course = $request->course;
    $student->save();

    return response()->json([
        "message" => "student record created"
    ], 201);
  }
  // ...
}
```

The snippet above imports the `Student` model which will interact with our `students` table in the database. In the `createStudent()` method, we instantiated a new `Request` object in the method parameter followed by a new `Student` object. Lastly, for every `$student-><column-name>` the equivalent request is fetched and saved.

If the operation is successful, a JSON response will be sent back to the API user with the message "student record created" and with response code 201.

This method is already tied to the `api/students` as we previously defined it in our routes file located at *routes/api.php*:

PHP

```php
// ...
Route::post('students, 'ApiController@createStudent');
// ...
```

Testing

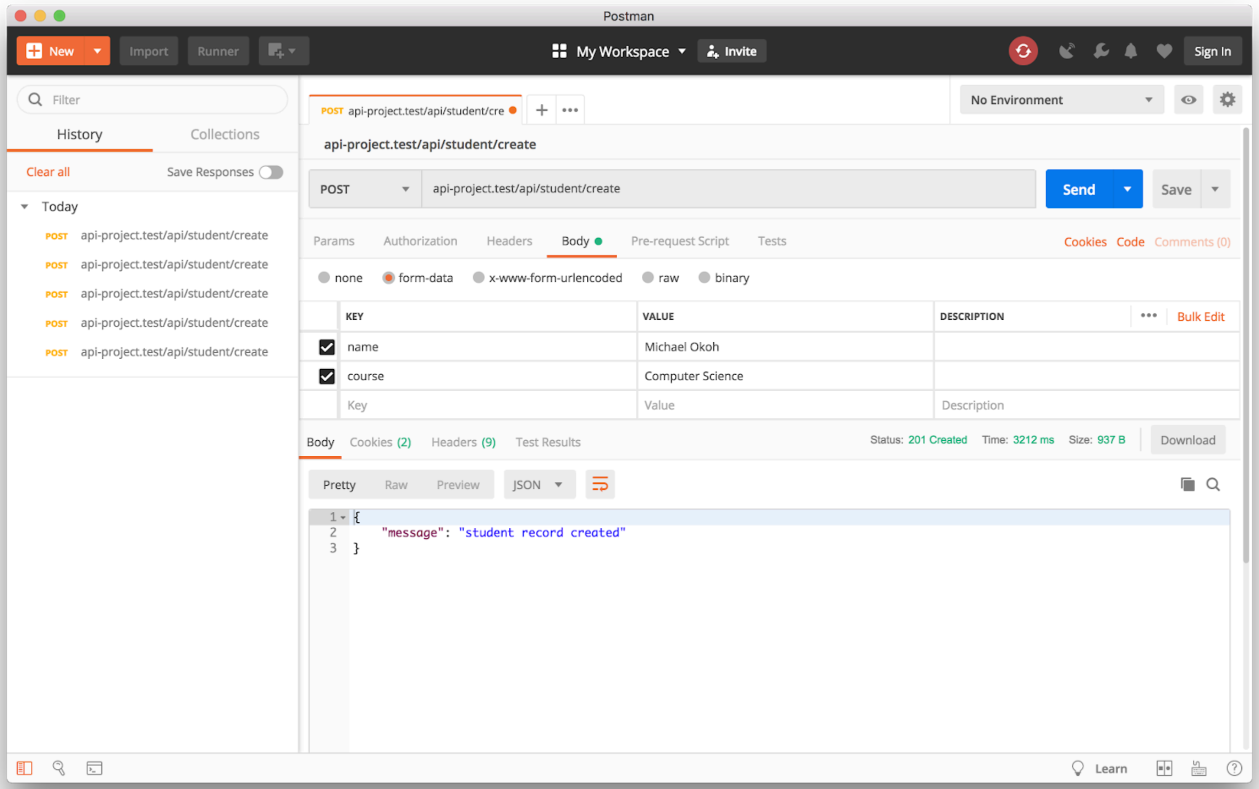
Before testing, make sure your application is running. You can use the inbuilt command as mentioned earlier:

Bash

```bash
php artisan serve
```

Or you can use Valet which is a nice tool for creating a proxy pass for all your PHP applications, providing you with a `*.test` or `*.dev` domain for your applications to test locally.

To test this endpoint open Postman and make a POST request to http://localhost:8000/api/students or if you use Valet `http://<folder-name>/api/students`. Select the `form-data` option and pass the following values as seen in the image below:


Postman
New
Import
Runner
My Workspace
Invite
Sign In
Filter
History
Collections
Clear all
Save Responses
Today
POST api-project.test/api/student/create
POST api-project.test/api/student/create
POST api-project.test/api/student/create
POST api-project.test/api/student/create
POST api-project.test/api/student/create
No Environment
Send
Save
Params Authorization Headers Body Pre-request Script Tests
Cookies Code Comments (0)
none form-data x-www-form-urlencoded raw binary
Bulk Edit
Body Cookies (2) Headers (9) Test Results
Download
Pretty Raw Preview JSON
Learn

POST api-project.test/api/student/create

KEY	VALUE	DESCRIPTION
✓ name	Michael Okoh	
✓ course	Computer Science	
Key	Value	Description

Status: 201 Created Time: 3212 ms Size: 937 B

```json
{
    "message": "student record created"
}
```

It works if it returns the success message along with the 201 response code, Now try adding a few more records to populate our database for the next task.

Return all Student Records

Now let us visit the `getAllStudents` method in our `ApiController`

PHP

```php
public function getAllStudents() {
  // logic to get all students goes here
}
```

We will use the already imported `Student` model to make a simple eloquent query to return all students in the database.

PHP

```php
class ApiController extends Controller
{
  public function getAllStudents() {
    $students = Student::get()->toJson(JSON_PRETTY_PRINT);
    return response($students, 200);
  }
  // ...
}
```

The eloquent query ends with `->toJson(JSON_PRETTY_PRINT);` which will serialize the object data return by eloquent into a nicely formatted JSON. The JSON is returned with the response code 200.

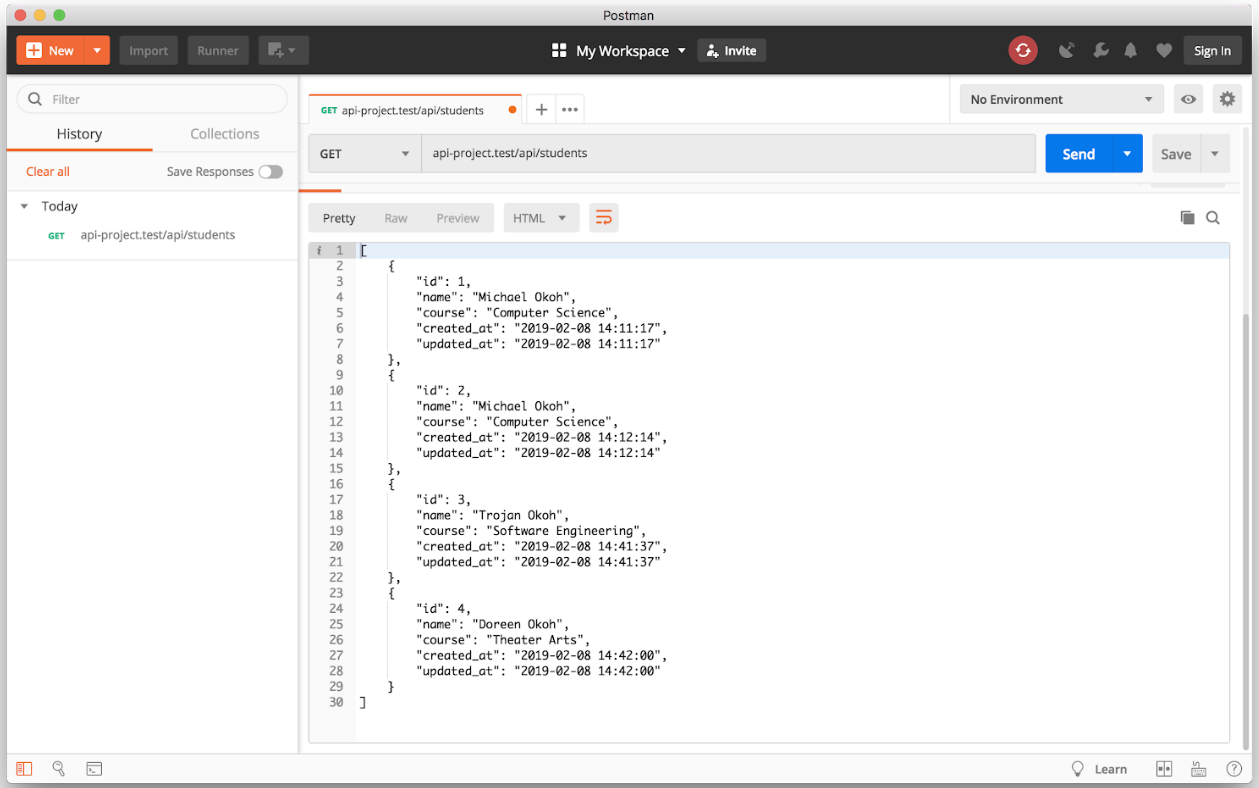
This method is already tied to the `api/students` route as we previously defined it in our routes file located at *routes/api.php*:

PHP

```php
// ...
Route::get('students', 'ApiController@getAllStudents');
// ...
```

Testing

Assuming our application is running in the background, make a GET request to the `/api/students` endpoint in Postman.



```
[
    {
        "id": 1,
        "name": "Michael Okoh",
        "course": "Computer Science",
        "created_at": "2019-02-08 14:11:17",
        "updated_at": "2019-02-08 14:11:17"
    },
    {
        "id": 2,
        "name": "Michael Okoh",
        "course": "Computer Science",
        "created_at": "2019-02-08 14:12:14",
        "updated_at": "2019-02-08 14:12:14"
    },
    {
        "id": 3,
        "name": "Trojan Okoh",
        "course": "Software Engineering",
        "created_at": "2019-02-08 14:41:37",
        "updated_at": "2019-02-08 14:41:37"
    },
    {
        "id": 4,
        "name": "Doreen Okoh",
        "course": "Theater Arts",
        "created_at": "2019-02-08 14:42:00",
        "updated_at": "2019-02-08 14:42:00"
    }
]
```

As seen in the image above, the endpoint returns all the student records in the database.

Return a student record

You will be creating an endpoint to return just a single student record. To begin you have to visit the `getStudent()` method in the `ApiController`.

PHP

```php
public function getStudent($id) {
  // logic to get a student record goes here
}
```

We will retrieve a student record by its `id` and to this, we will be making an eloquent query to return student records by their `id`.

PHP

```php
class ApiController extends Controller
{
  // ...
  public function getStudent($id) {
    if (Student::where('id', $id)->exists()) {
        $student = Student::where('id', $id)->get()->toJson(JSON_
        return response($student, 200);
      } else {
        return response()->json([
          "message" => "Student not found"
        ], 404);
      }
  }
  // ...
}
```

The snippet above first checks if a student record with the given `id` exists. If it does, it queries the database using eloquent to return the record with matching `id` in JSON with 200 as the response code. If the `id` given is not found in the database it will return a "student not found" message with a 404 response code.

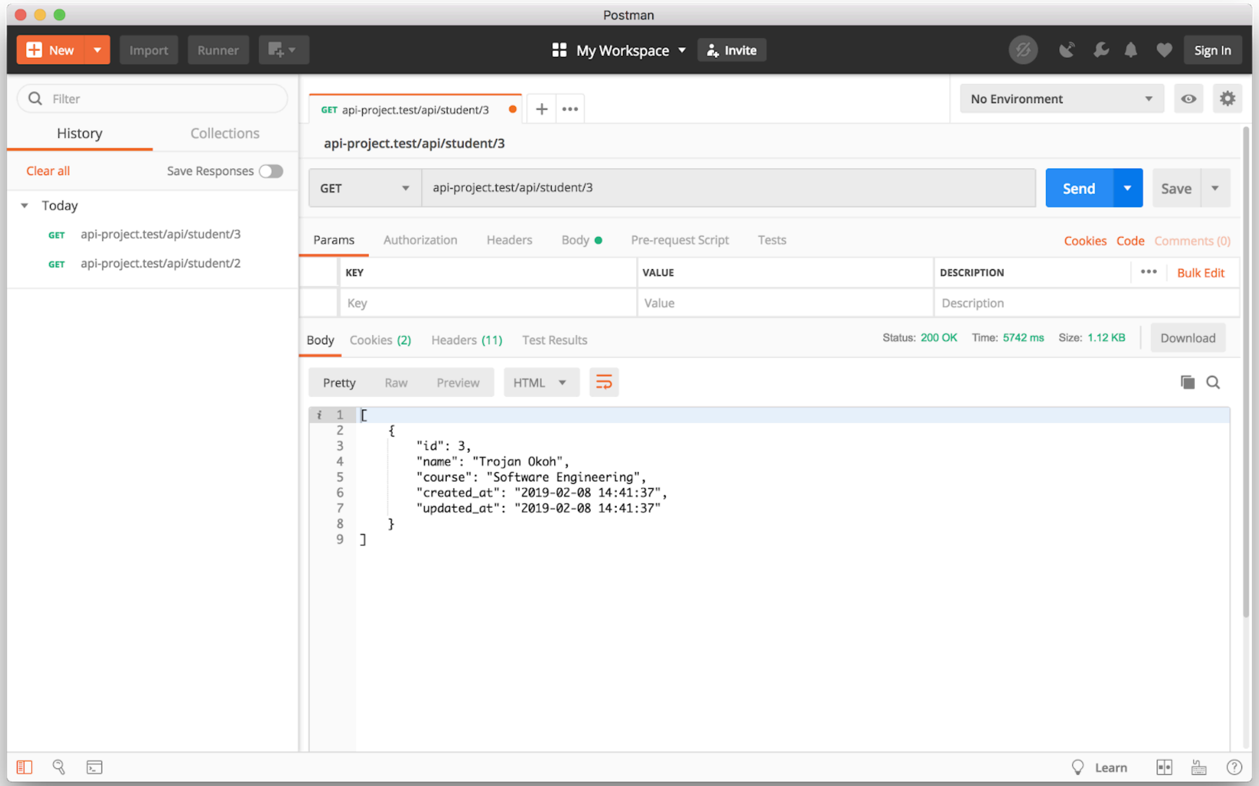
This method is already tied to the `api/students/{id}` route as we previously defined it in our routes file located at *routes/api.php*:

PHP

```php
Route::get('students/{id}', 'ApiController@getStudent');
```

Testing

Open Postman and make a GET request to the `/api/students/{id}` endpoint `{id}` can be the `id` of an existing record you may have in your database.



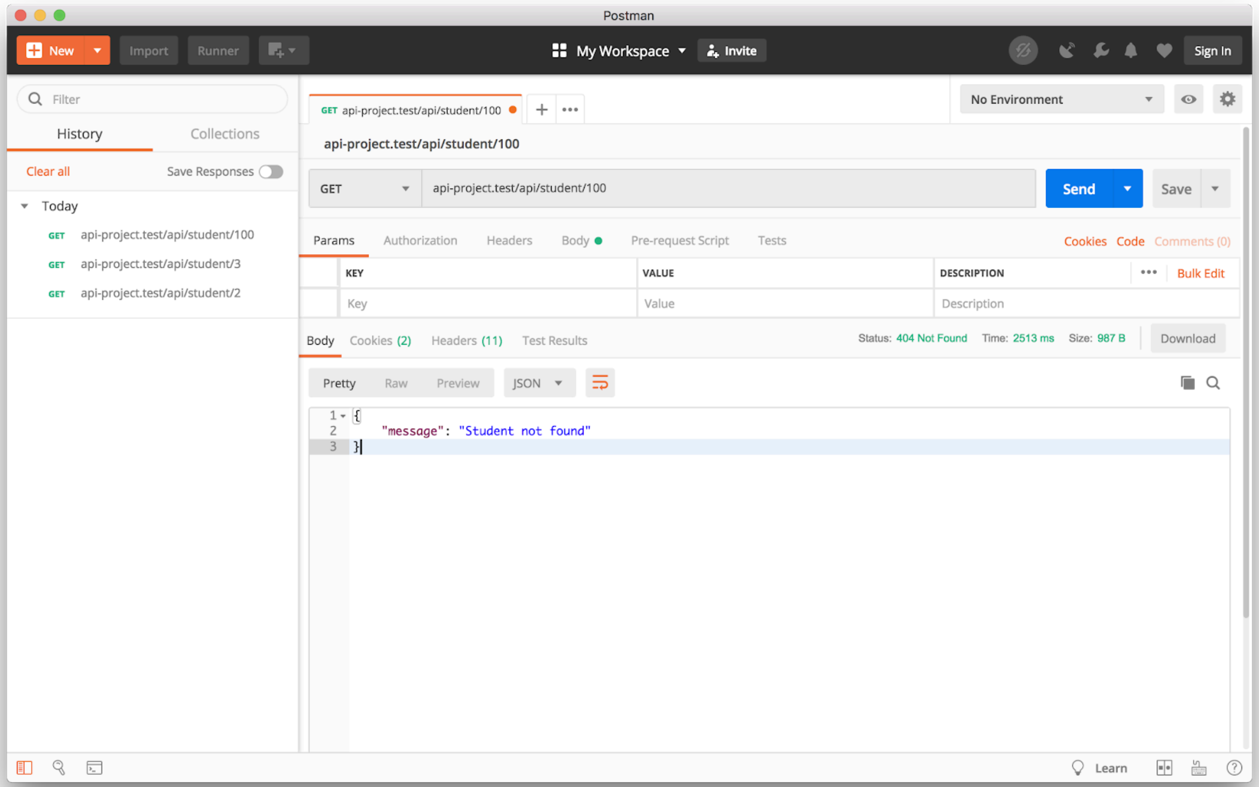
api-project.test/api/student/3

GET api-project.test/api/student/3

Status: 200 OK Time: 5742 ms Size: 1.12 KB

```
[
    {
        "id": 3,
        "name": "Trojan Okoh",
        "course": "Software Engineering",
        "created_at": "2019-02-08 14:41:37",
        "updated_at": "2019-02-08 14:41:37"
    }
]
```

As seen in the image above, I made a request to `http://api-project.test/api/students/3` and the details of the student assigned to that `id` were returned. Next, let us try requesting a non-existent student record.



GET api-project.test/api/student/100

Status: 404 Not Found Time: 2513 ms Size: 987 B

```
{
    "message": "Student not found"
}
```

As seen in the image above, a request was made to the endpoint to return the details of the student record with the `id` of 100 which is non-existent. Our API did a good job by returning an error message along with the 404 status code.

Update a student record

We will now be creating an endpoint to update the details of an existing student record. To begin you have to visit the `updateStudent()` method in the `ApiController`.

PHP

```php
public function updateStudent(Request $request, $id) {
  // logic to update a student record goes here
}
```

To do this we will have to check if the record we are trying to update exists. If it does exist it will update the records which match the `id` specified and return status code 204. If it does not exist, it will return a message indicating that the record was not found along with status code 404.

PHP

```php
public function updateStudent(Request $request, $id)
{
    if (Student::where('id', $id)->exists()) {
        $student = Student::find($id);
        $student->name = is_null($request->name) ? $student->nam
        $student->course = is_null($request->course) ? $student-
        $student->save();

        return response()->json(
            [
                "message" => "records updated successfully"
            ],
            200
        );
    } else {
        return response()->json([
            "message" => "Student not found"
        ], 404);
    }
}
```

Validation was added just in case you need to only update a single attribute such as `name` or `course`. As the request comes in it checks if name or `course` is null. If it is null, it replaces the request with its existing value. If it isn't, null it passed as the new value. All this was done using ternary operators.

The format for the ternary operator is `condition ? true : false`.

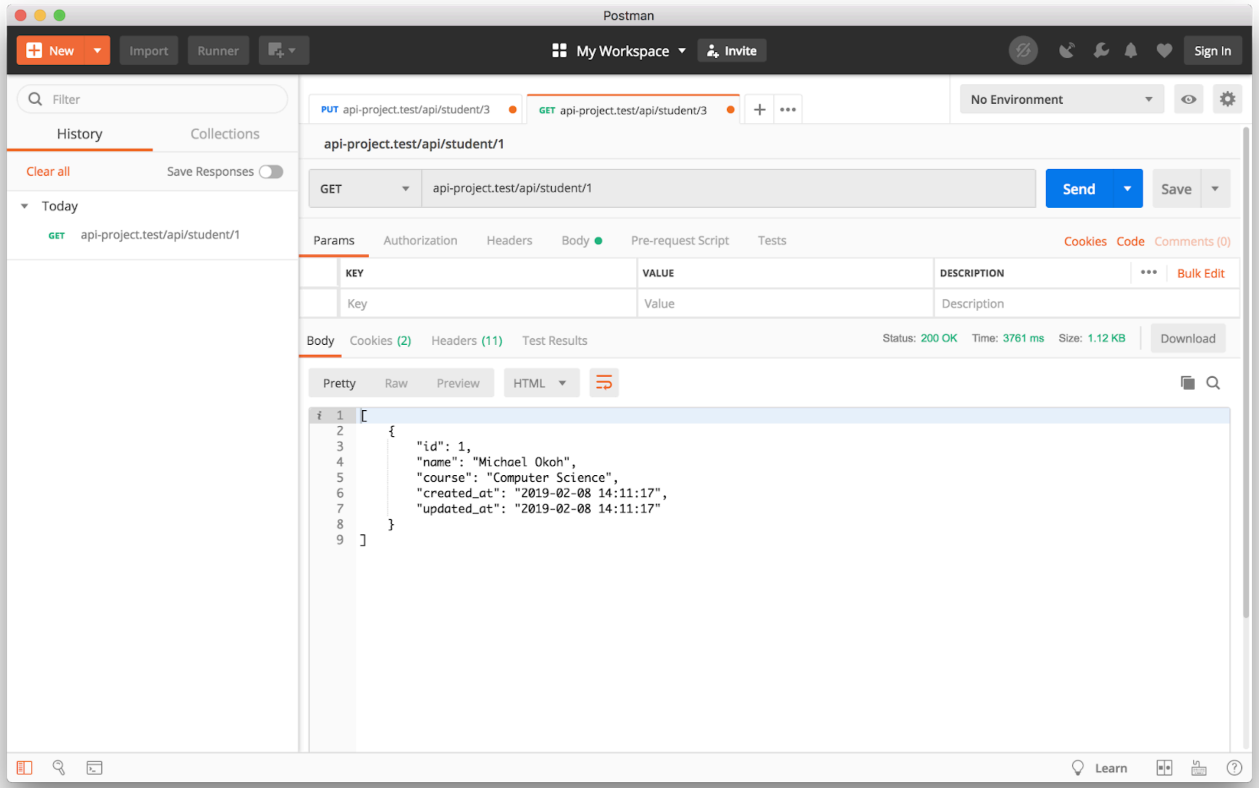
This method is already tied to the `api/students/{id}` route as we previously defined it in our routes file located at *routes/api.php*:

PHP

```
Route::put('students/{id}', 'ApiController@updateStudent');
```

Testing

To test this endpoint, return the details of the student record with the `id` of `1` by making a `GET` request to `/api/students/1`.



api-project.test/api/student/1

GET api-project.test/api/student/1

Status: 200 OK Time: 3761 ms Size: 1.12 KB

```
[
    {
        "id": 1,
        "name": "Michael Okoh",
        "course": "Computer Science",
        "created_at": "2019-02-08 14:11:17",
        "updated_at": "2019-02-08 14:11:17"
    }
]
```

The following records were returned:

Json

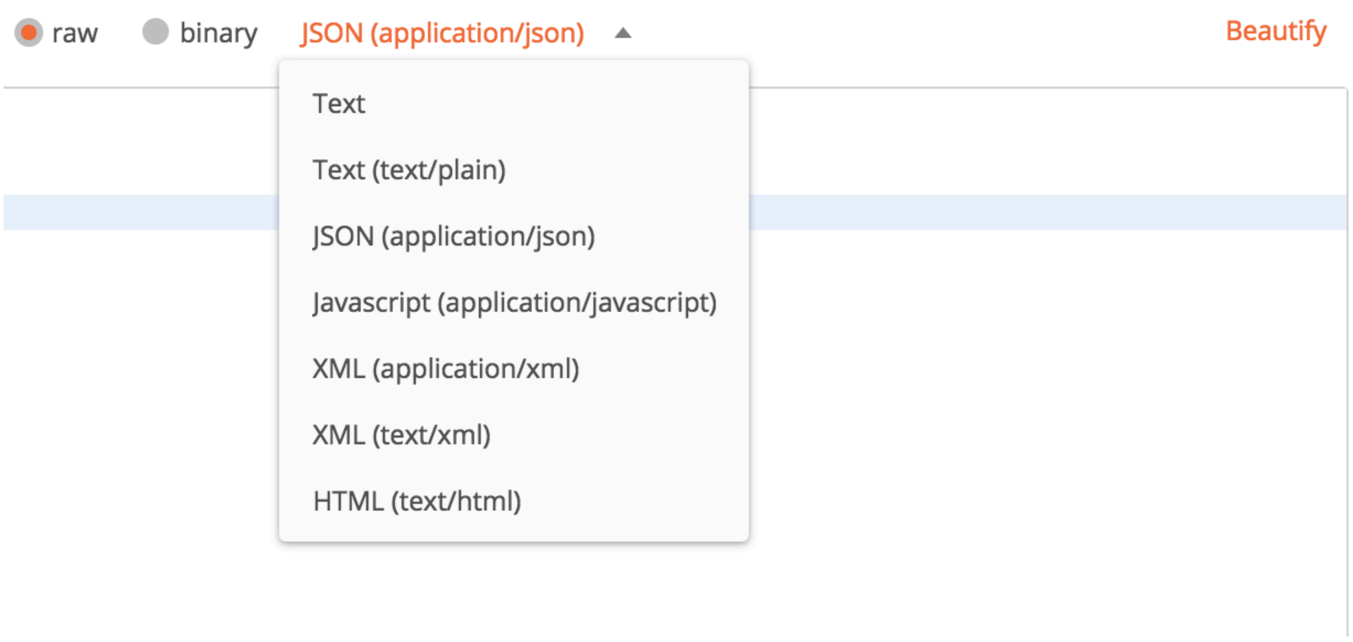
```
{
    "id": 1
}
```

Next, let us change the `course` to “Software Engineering” by making a PUT request to `api/students/1`. In order to make a PUT request, you have to pass a JSON payload via `form-data`. Now let us change the value of `name` to `Trojan Okoh` and the value of `course` to `Software Engineering`.

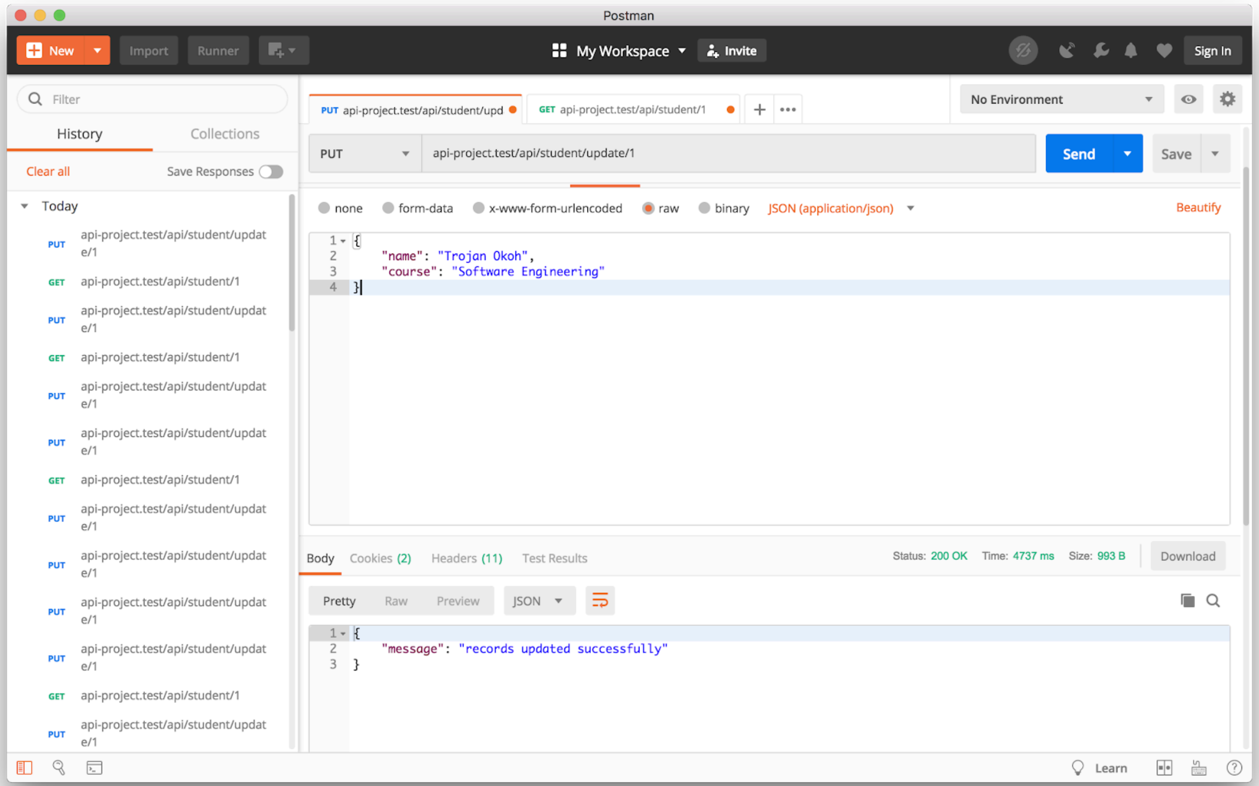
Json

```json
{
      "name": "Trojan Okoh",
      "course": "Software Engineering"
}
```

The snippet above is the JSON payload we will be using to update the records. Open Postman and change to "raw" and change type to "JSON (application/json)" as seen below.



Next, paste the JSON payload into the text area and send the PUT request to the endpoint.



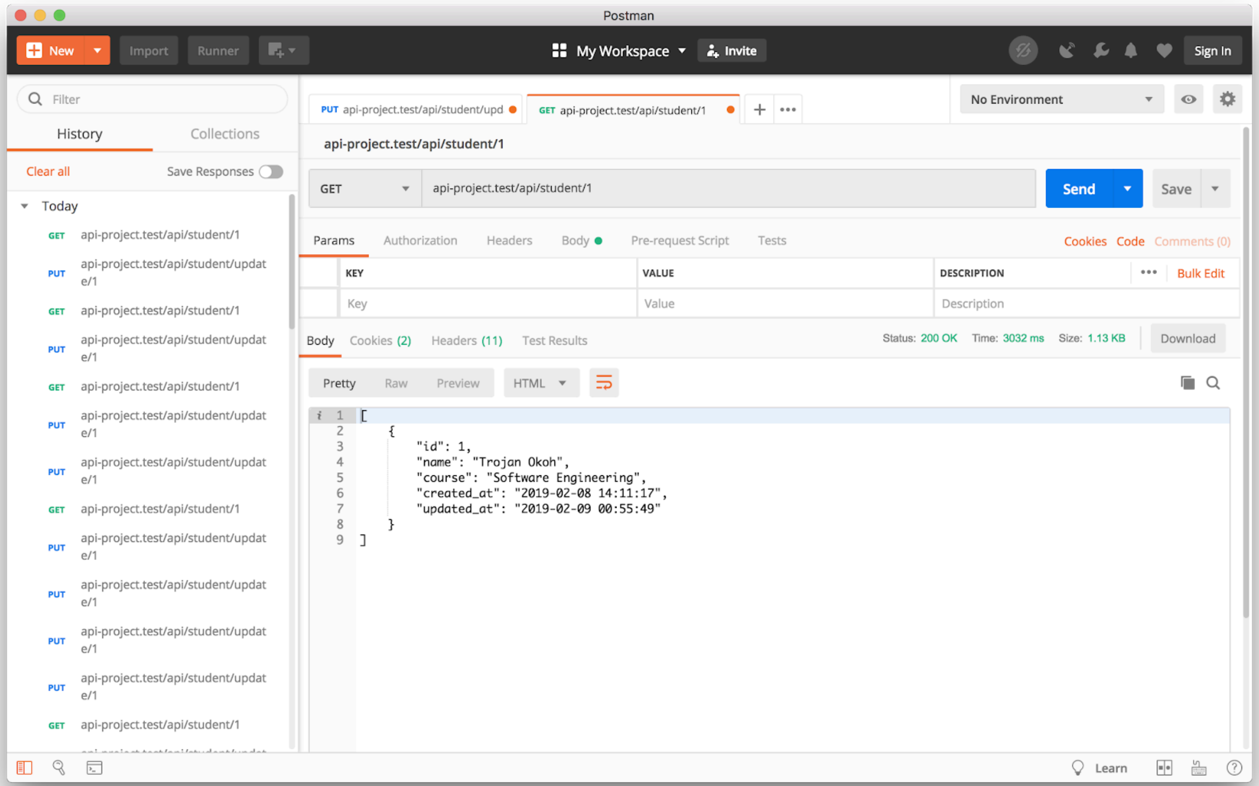
PUT api-project.test/api/student/update/1

```
{
    "name": "Trojan Okoh",
    "course": "Software Engineering"
}
```

Status: 200 OK Time: 4737 ms Size: 993 B

```
{
    "message": "records updated successfully"
}
```

As seen in the image above, the endpoint returned a success message. Now let us make a GET request to `/api/students/1` to confirm if the records were actually updated.



```
[
    {
        "id": 1,
        "name": "Trojan Okoh",
        "course": "Software Engineering",
        "created_at": "2019-02-08 14:11:17",
        "updated_at": "2019-02-09 00:55:49"
    }
]
```

Delete a Student Record

Finally, to delete a student record we will have to visit the `deleteStudent()` method in our `ApiController`.

PHP

```php
public function deleteStudent ($id) {
    // logic to delete a student record goes here
}
```

Using eloquent, we will check if the `id` of the record requested to be deleted exists. If it exists we will delete the record. If it does not exist, we will return a `not found` message along with the 404 status code.

PHP

```php
class ApiController extends Controller
{
    // ...
    public function deleteStudent ($id) {
      if(Student::where('id', $id)->exists()) {
        $student = Student::find($id);
        $student->delete();

        return response()->json([
          "message" => "records deleted"
        ], 202);
      } else {
        return response()->json([
          "message" => "Student not found"
        ], 404);
      }
    }
}
```

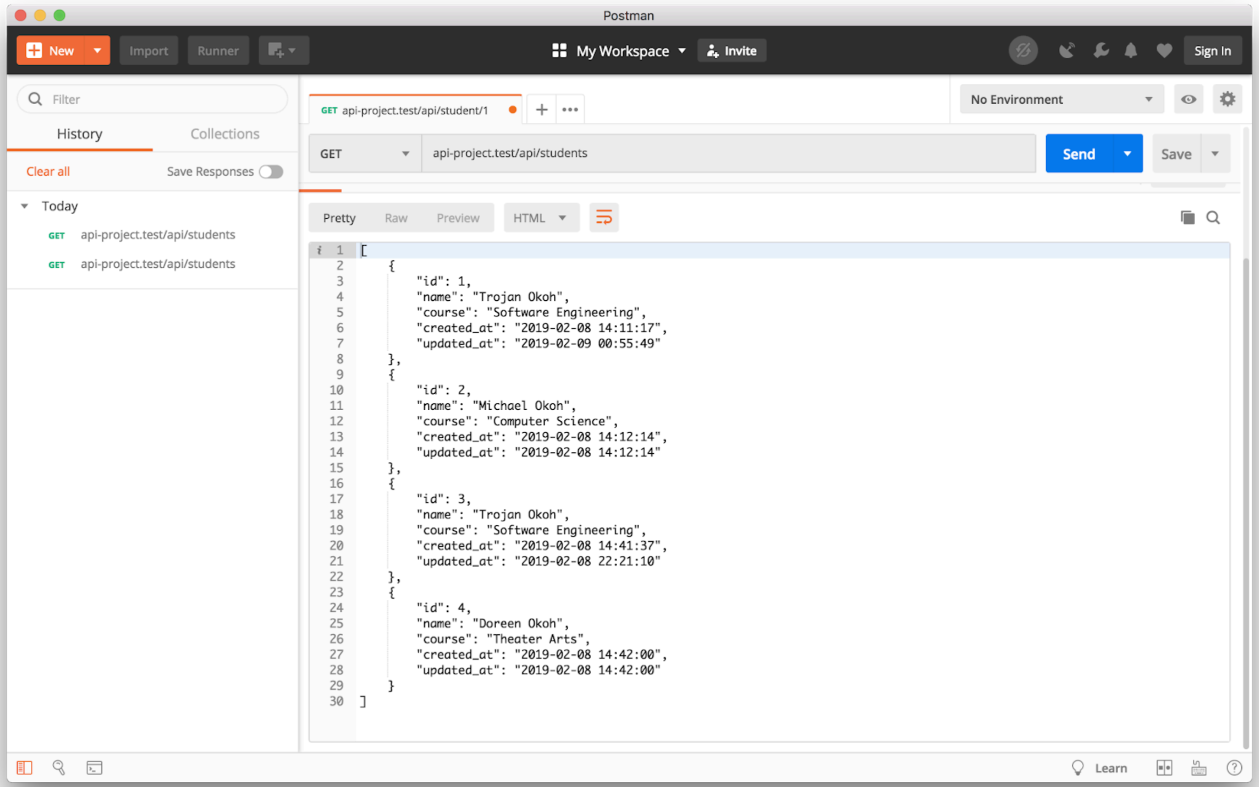
This method is already tied to the `api/students/{id}` route as we previously defined it in our routes file located at *routes/api.php*:

PHP

```php
Route::delete('students/{id}', 'ApiController@deleteStudent');
```

Testing

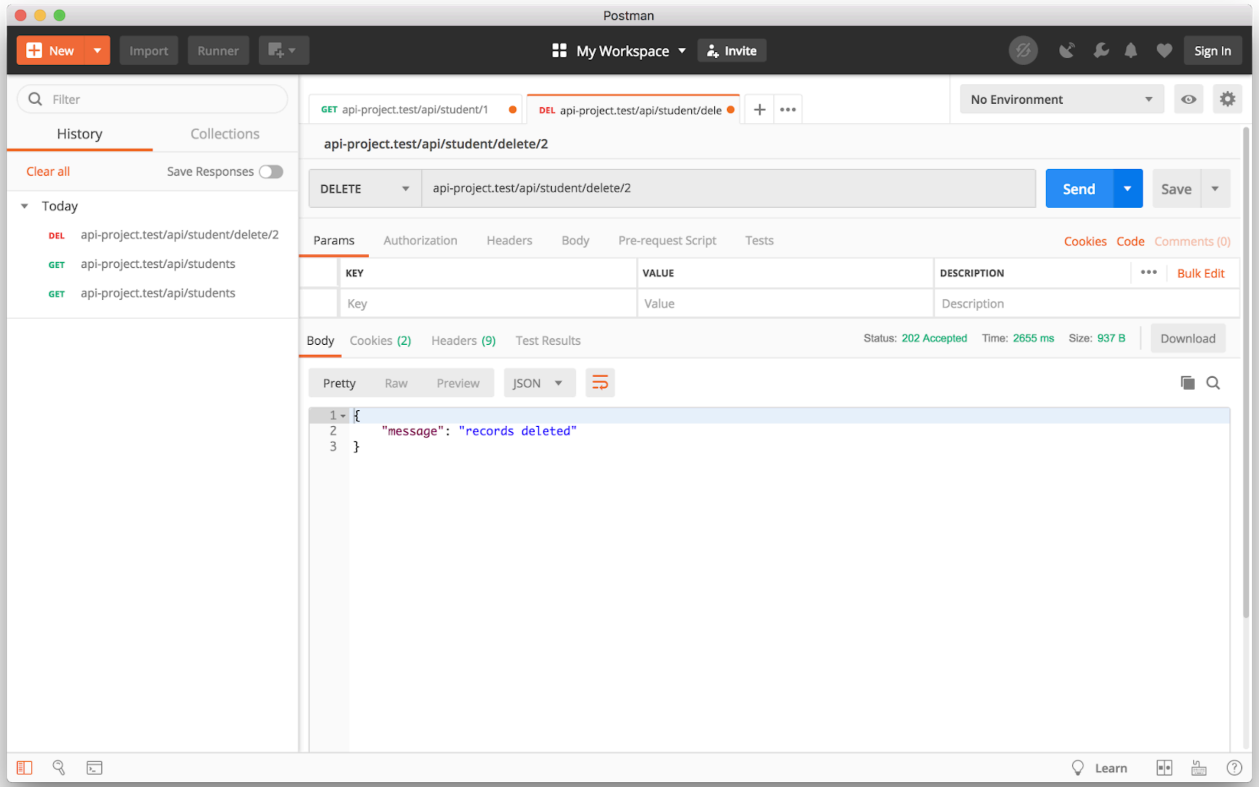
To test this endpoint, we will have to list all the records we currently have in our database by making a GET request to the `/api/students` endpoint.



GET api-project.test/api/students

```
[
    {
        "id": 1,
        "name": "Trojan Okoh",
        "course": "Software Engineering",
        "created_at": "2019-02-08 14:11:17",
        "updated_at": "2019-02-09 00:55:49"
    },
    {
        "id": 2,
        "name": "Michael Okoh",
        "course": "Computer Science",
        "created_at": "2019-02-08 14:12:14",
        "updated_at": "2019-02-08 14:12:14"
    },
    {
        "id": 3,
        "name": "Trojan Okoh",
        "course": "Software Engineering",
        "created_at": "2019-02-08 14:41:37",
        "updated_at": "2019-02-08 22:21:10"
    },
    {
        "id": 4,
        "name": "Doreen Okoh",
        "course": "Theater Arts",
        "created_at": "2019-02-08 14:42:00",
        "updated_at": "2019-02-08 14:42:00"
    }
]
```

Next, we will make a DELETE request to `students/{id}` where `{id}` is the `id` of the record we are requesting to be deleted. For the purpose of testing, I will delete the record with the `id` of `2`.



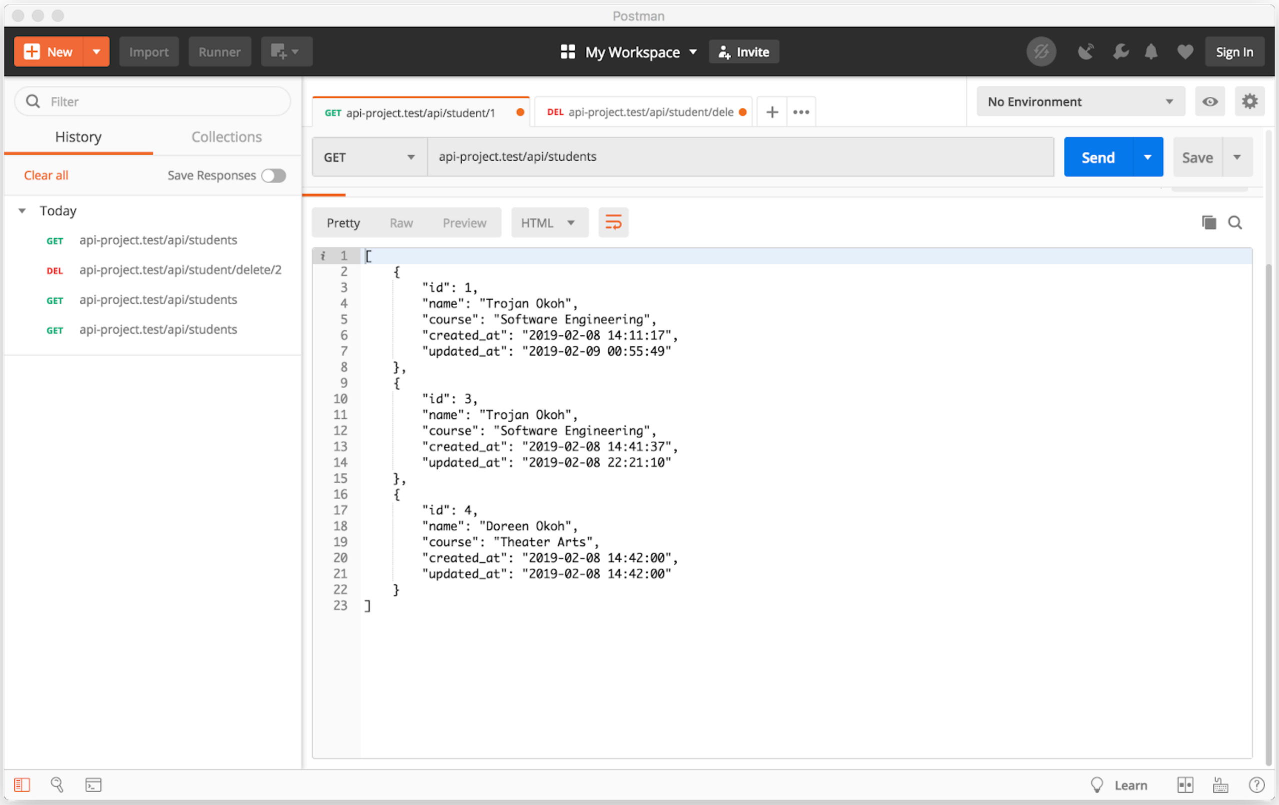
api-project.test/api/student/delete/2

DELETE api-project.test/api/student/delete/2

Status: 202 Accepted Time: 2655 ms Size: 937 B

```
{
    "message": "records deleted"
}
```

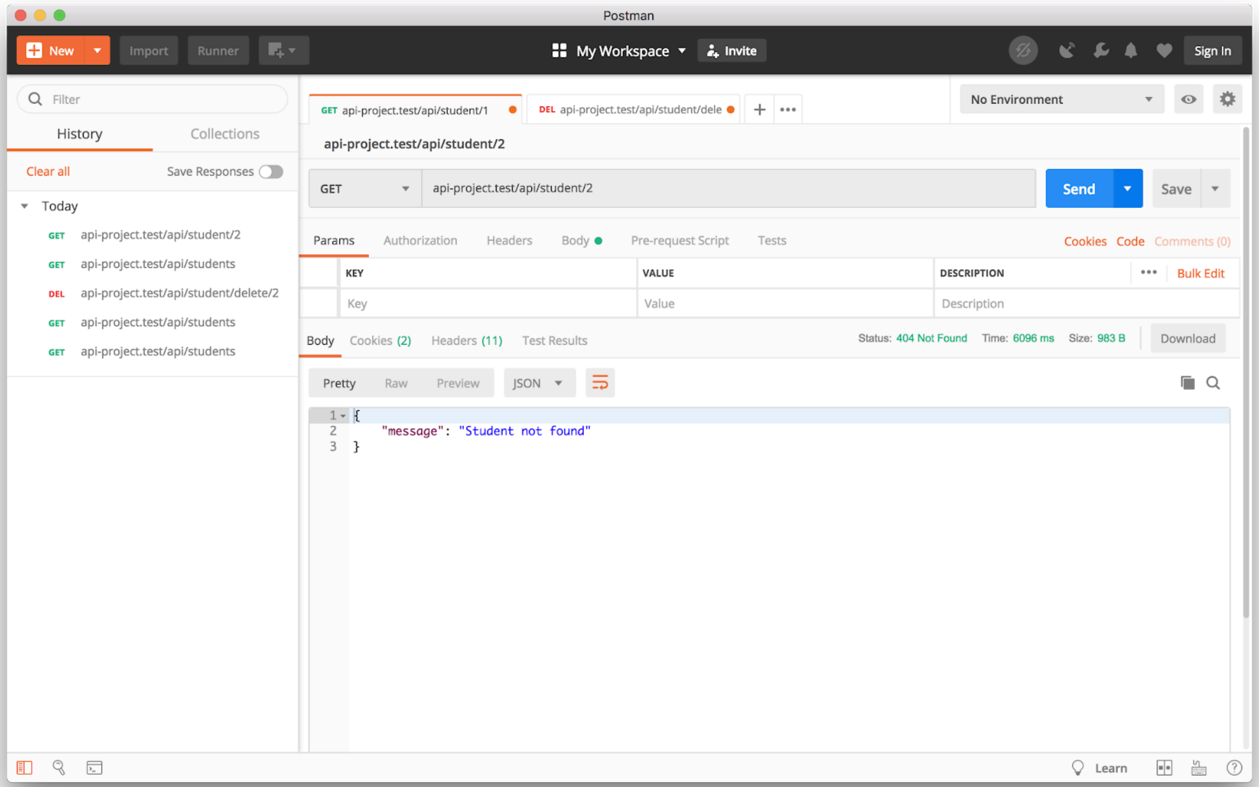
The endpoint returned a success message along with status code 202 which means the request was accepted. To confirm if the record was actually deleted, let us try making a GET request to the `/api/students` endpoint to list all the student records we have in the database.


Postman
New
Import
Runner
My Workspace
Invite
Sign In
Filter
History
Collections
Clear all
Save Responses
Today
GET api-project.test/api/students
DEL api-project.test/api/student/delete/2
GET api-project.test/api/students
GET api-project.test/api/students
GET api-project.test/api/student/1
DEL api-project.test/api/student/dele
No Environment
Send
Save
Pretty
Raw
Preview
HTML
Learn

GET api-project.test/api/students

```
[
    {
        "id": 1,
        "name": "Trojan Okoh",
        "course": "Software Engineering",
        "created_at": "2019-02-08 14:11:17",
        "updated_at": "2019-02-09 00:55:49"
    },
    {
        "id": 3,
        "name": "Trojan Okoh",
        "course": "Software Engineering",
        "created_at": "2019-02-08 14:41:37",
        "updated_at": "2019-02-08 22:21:10"
    },
    {
        "id": 4,
        "name": "Doreen Okoh",
        "course": "Theater Arts",
        "created_at": "2019-02-08 14:42:00",
        "updated_at": "2019-02-08 14:42:00"
    }
]
```

As seen in the image above, the record with the `id` of `2` no longer exist. Also, we can check by trying to request the record with the `id` of `2` by making a GET request to the `/api/students/{id}` endpoint. It should return a 404 indicating that the record could not be found.


GET api-project.test/api/student/2

Status: 404 Not Found Time: 6096 ms Size: 983 B

```
{
    "message": "Student not found"
}
```

Conclusion

Now that you have gotten to the end of this Article, let us confirm the contents of some important files.

app\http\controllers\ApiController.php

PHP

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Student;

class ApiController extends Controller
{
    public function getAllStudents() {
      $students = Student::get()->toJson(JSON_PRETTY_PRINT);
      return response($students, 200);
    }

    public function createStudent(Request $request) {
      $student = new Student;
      $student->name = $request->name;
      $student->course = $request->course;
      $student->save();

      return response()->json([
        "message" => "student record created"
      ], 201);
    }

    public function getStudent($id) {
      if (Student::where('id', $id)->exists()) {
        $student = Student::where('id', $id)->get()->toJson(JSO
        return response($student, 200);
      } else {
        return response()->json([
          "message" => "Student not found"
        ], 404);
      }
    }

    public function updateStudent(Request $request, $id) {
      if (Student::where('id', $id)->exists()) {
        $student = Student::find($id);

        $student->name = is_null($request->name) ? $student->na
        $student->course = is_null($request->course) ? $student
        $student->save();

        return response()->json([
          "message" => "records updated successfully"
        ], 200);
      } else {
        return response()->json([
          "message" => "Student not found"
        ], 404);
      }
    }

    public function deleteStudent ($id) {
      if(Student::where('id', $id)->exists()) {
        $student = Student::find($id);
        $student->delete();

        return response()->json([
          "message" => "records deleted"
        ], 202);
      } else {
        return response()->json([
          "message" => "Student not found"
        ], 404);
      }
    }
}
```

routes\web.php

PHP

```
<?php

use Illuminate\Http\Request;

/*
|-------------------------------------------------------------
| API Routes
|-------------------------------------------------------------
|
| Here is where you can register API routes for your applicatio
| routes are loaded by the RouteServiceProvider within a group
| is assigned the "api" middleware group. Enjoy building your A
|

*/

Route::middleware('auth:api')->get('/user', function (Request $
    return $request->user();
});


Route::get('students', 'ApiController@getAllStudents');
Route::get('students/{id}', 'ApiController@getStudent');
Route::post('students, 'ApiController@createStudent');
Route::put('students/{id}', 'ApiController@updateStudent');
Route::delete('students/{id}', 'ApiController@deleteStudent');
```

app\Student.php

PHP

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Student extends Model
{
    protected $table = 'students';

    protected $fillable = ['name', 'course'];
}
```

We have been able to build a simple CRUD RESTful API using Laravel. This article covered the basics of the subject matter. I did not cover request validation and API security which would make a great next step for you to implement.

Twitter: @ichtrojan

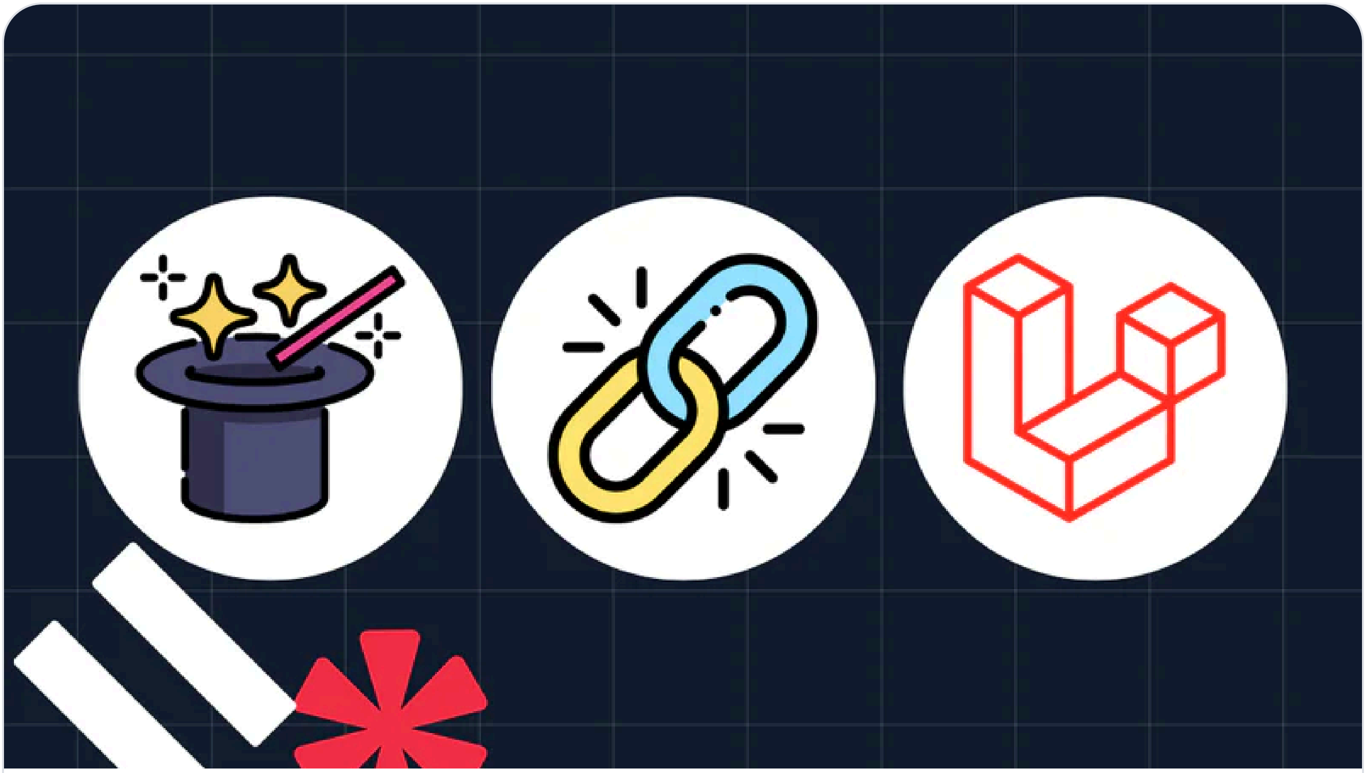
GitHub: @ichtrojan

E-mail: michael@okoh.co.uk

Related Posts

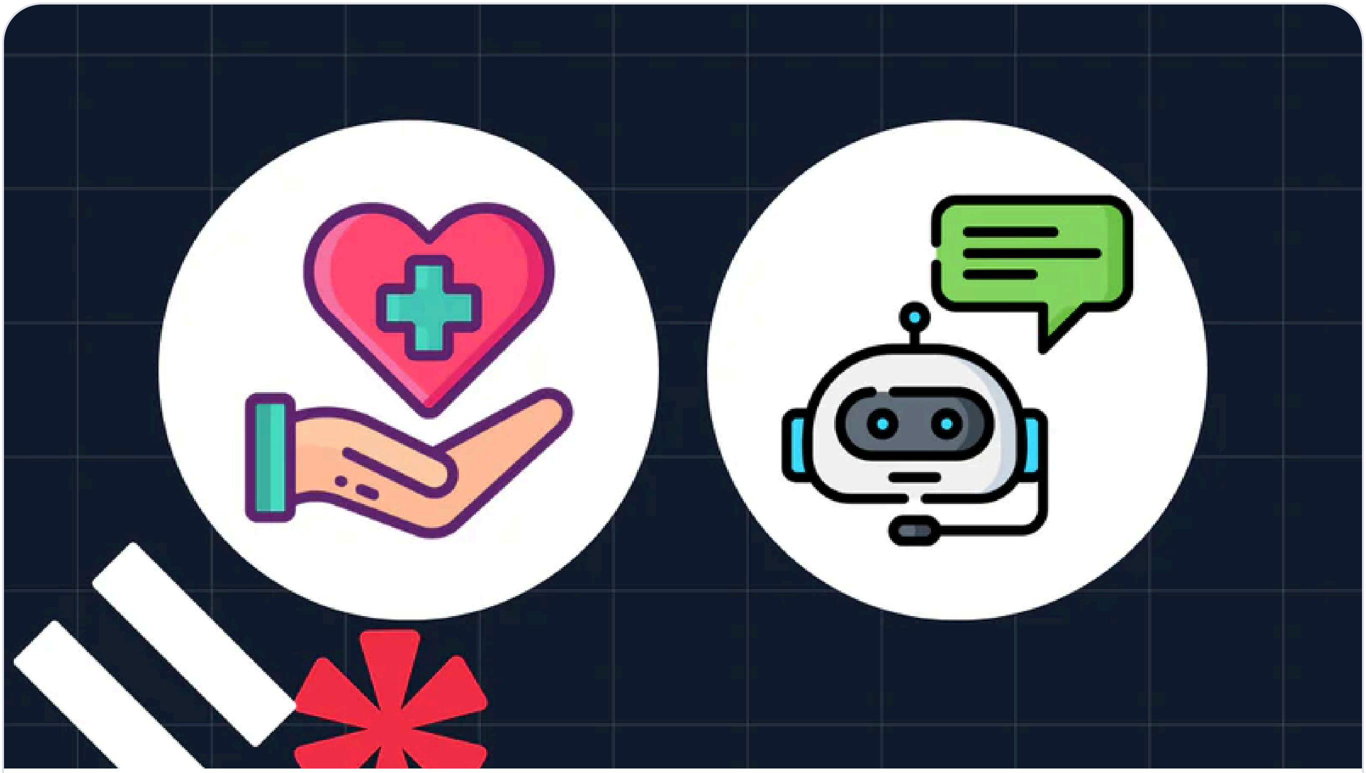
The Ins and Outs of DMARC Monitoring in 2026

Denis O'Sullivan



How to Create a Magic Link in Laravel with Twilio SMS

Popoola Temitope



Creating a Health Chatbot Using Twilio and Gemini: A Step-by-Step Guide

Oluseye Jeremiah

Related Resources

Twilio Docs

From APIs to SDKs to sample apps

API reference documentation, SDKs, helper libraries, quickstarts, and tutorials for your language and platform.

→

Resource Center

The latest ebooks, industry reports, and webinars

Learn from customer engagement experts to improve your own communication.

→

Ahoy

Twilio's developer community hub

Best practices, code samples, and inspiration to build communications and digital engagement experiences.

→