



Outlines of Discussion

- ❖ Accessors in Laravel
- ❖ Mutators in Laravel
- ❖ Relationship in Laravel
- ❖ API in laravel
- ❖ Types of APIs in Laravel
- ❖ API Methods is Laravel
- ❖ API Validation

Accessors in Laravel

- In Laravel, accessors are special methods used to transform attributes in your model whenever they are accessed. This feature is particularly useful when you want to customize the way certain data is presented or format it before it is retrieved.

How Accessors Work

How Accessors Work

Accessors follow a specific naming convention:

- The method name should be `get{Attribute}Attribute`.
- `{Attribute}` is the name of the attribute you want to modify or format.

For example, if you have a `name` column in your database, the accessor to modify it would be named `getNameAttribute`.

Defining Accessors

Defining Accessors

To define an accessor, add it to your Eloquent model. For instance, let's say you have a `User` model with `first_name` and `last_name` attributes, and you want to create a `full_name` attribute.

Example: Full Name Accessor

```
php

class User extends Model

{
    public function getFullNameAttribute()
    {
        return "{$this->first_name} {$this->last_name}";
    }
}
```

Now, you can access `full_name` as if it were a property:

php

```
$user = User::find(1);
echo $user->full_name; // Outputs: "John Doe" (if first_name is John and last_name is Doe)
```



Mutators in Laravel

- In Laravel, mutators are methods that allow you to modify the value of an attribute before it is saved to the database. They are the counterpart to accessors, which format data when retrieving it.

How Mutators Work

Mutators follow a specific naming convention:

- The method name should be `set{Attribute}Attribute`.
- `{Attribute}` is the name of the database column or model attribute you want to modify.

For example, if you want to modify a `name` attribute, the mutator method would be
`setNameAttribute`.

Defining Mutators

Defining Mutators

You define a mutator in your model class. The value you set in the mutator method will be automatically applied whenever the attribute is assigned.

Example: Storing Name in Lowercase

php

 Copy code

```
class User extends Model
{
    public function setNameAttribute($value)
    {
        $this->attributes['name'] = strtolower($value);
    }
}
```

Cont. ...

Now, whenever you assign a value to `name`, it will be converted to lowercase before being saved:

php

 Copy code

```
$user = new User();
$user->name = 'John DOE';
echo $user->name; // Outputs: "john doe"
```

Cont. ...

Key Points to Remember

1. **Naming Convention:** Mutators use `set{Attribute}Attribute`.
2. **Attribute Transformation:** Mutators modify values before they are stored in the database.
3. **No Impact on Retrieval:** Mutators do not affect how attributes are retrieved. For retrieval transformations, use **accessors**.
4. **Used with Input Validation:** Mutators are often paired with Laravel's validation rules to ensure data integrity.
5. **Automatic Application:** Mutators are automatically applied whenever you assign a value to the corresponding attribute.

Relationship in Laravel

- In Laravel, a **relationship** refers to the connection between database tables, enabling you to interact with related data efficiently.
- Relationships are defined in **Eloquent models**, which map your database tables to PHP classes. By defining relationships, you can fetch related data without writing complex SQL queries.

Types of Relationships in Laravel

1. One-to-One Relationship

- **Description:** One record in a table is associated with one record in another table.
- **Example:** A user has one profile.

Table Structure:

Users Table	Profiles Table
id (PK)	id (PK)
name	user_id (FK)
	bio

Example (One-to-One Relationship)

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with a dark theme. The left sidebar contains icons for file operations like Open, Save, Find, and Undo. The Explorer sidebar shows a project structure under 'LARAVEL-WORKSPACE' named 'laravel-demo'. Inside 'resources/views', several blade files are listed: 'search-employee.blade.php', 'SessionView1.blade.php', 'SessionView2.blade.php', 'Students.blade.php', 'UploadFile.blade.php', 'url1.blade.php', 'url2.blade.php', 'userform.blade.php', 'userform2.blade.php', and 'welcome.blade.php'. Under 'routes', there are 'console.php' and 'web.php'. The 'routes/web.php' file is currently selected and shown in the main editor area. The status bar at the bottom indicates the file path: 'laravel-demo > routes > web.php > ...'. The terminal tab is active, showing the command line output of Laravel artisan commands:

```
yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/laravel-demo (master)
$ php artisan make:controller RelationshipController
INFO Controller [C:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\app\Http\Controllers\RelationshipController.php] created successfully.

yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/laravel-demo (master)
$ php artisan make:model Product
INFO Model [C:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\app\Models\Product.php] created successfully.

yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/laravel-demo (master)
$ php artisan make:model Seller
INFO Model [C:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\app\Models\Seller.php] created successfully.
```

Databases

Server: 127.0.0.1 » Database: laravel » Table: products

[Browse](#) [Structure](#) [SQL](#) [Search](#) [Insert](#) [Export](#) [Import](#)

Showing rows 0 - 2 (3 total, Query took 0.0002 seconds.)

```
SELECT * FROM `products`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 ▾ Filter rows: Search this table Sort by key:

Extra options

	id	name	price	seller_id	created_at	updated_at
<input type="checkbox"/>	1	iphone	1000	1	2024-11-20	2024-11-21
<input type="checkbox"/>	2	samsung	2000	2	2024-11-22	2024-11-23
<input type="checkbox"/>	3	OOP	3000	3	2024-11-24	2024-11-25

Server: 127.0.0.1 » Database: laravel » Table: sellers

[Browse](#) [Structure](#) [SQL](#) [Search](#) [Insert](#) [E](#)

Showing rows 0 - 1 (2 total, Query took 0.0003 seconds.)

```
SELECT * FROM `sellers`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 ▾ Filter rows: Search this table

Extra options

	id	name	created_at	updated_at
<input type="checkbox"/>	2	yitayew	2024-11-20	2024-11-21
<input type="checkbox"/>	3	solomon	2024-11-22	2024-11-24

Controller (RelationshipController)

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** Laravel-WorkSpace.
- Explorer:** Shows the file structure under LARAVEL-WORKSPACE/laravel-demo/app/Http/Controllers. The RelationshipController.php file is currently selected.
- Code Editor:** Displays the code for RelationshipController.php. The code defines a class RelationshipController that extends Controller and contains a function product_list() which returns a Seller model instance.

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\Seller;
7
8 class RelationshipController extends Controller
9 {
10     //
11     function product_list(){
12         return Seller::find(2)->ProductData;
13     }
14 }
```

Model (Seller)

The screenshot shows a code editor interface with a dark theme. On the left is a vertical toolbar with icons for file operations like New, Open, Save, and Find. Next to it is the Explorer sidebar titled 'EXPLORER' which shows the project structure under 'LARAVEL-WORKSPACE'. The 'app' folder contains several files: Http, Controllers, AddNewUser.php, Controller.php, DatabaseController.php, DbQueryBuilder.php, EmodelQueryBuilder.php, EmployeeController.php, FileUpload.php, FormValidation.php, GroupPrefix.php, HomeController.php, NamedRoute.php, PostController.php, and RelationshipContr... . The main editor area has tabs for 'web.php', 'RelationshipController.php', and 'Seller.php'. The 'Seller.php' tab is active, showing the following PHP code:

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Model;
6
7 class Seller extends Model
8 {
9     //
10    function ProductData(){
11        return $this->hasOne('App\Models\Product');
12    }
13 }
14
```

The status bar at the bottom indicates the file is 14 lines long.

Output

The screenshot shows a dark-themed browser interface for viewing JSON data. At the top, there are navigation icons (back, forward, refresh) and a URL bar displaying "127.0.0.1:8000/relationship". Below the URL bar, there are three tabs: "JSON" (which is selected), "Raw Data", and "Headers". A toolbar below the tabs includes buttons for "Save", "Copy", "Collapse All", "Expand All", and a "Filter JSON" input field.

Field	Value
id	2
name	"samsung"
price	2000
seller_id	2
created_at	"2024-11-22T00:00:00.000000Z"
updated_at	"2024-11-23T00:00:00.000000Z"

One to Many Relationship

2. One-to-Many Relationship

- **Description:** One record in a table can have multiple related records in another table.
- **Example:** A post has many comments.

Table Structure:

Posts Table	Comments Table
<code>id</code> (PK)	<code>id</code> (PK)
<code>title</code>	<code>post_id</code> (FK)
	<code>comment</code>

Example (Databases)

Server: 127.0.0.1 » Database: laravel » Table: sellers

[Browse](#) [Structure](#) [SQL](#) [Search](#) [Insert](#)

Showing rows 0 - 1 (2 total, Query took 0.0002 seconds.)

```
SELECT * FROM `sellers`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

	id	name	created_at	updated_at
<input type="checkbox"/>	1	iphone	1000	1 2024-11-20 2024-11-21
<input type="checkbox"/>	2	samsung	2000	2 2024-11-22 2024-11-23
<input type="checkbox"/>	3	OOP	3000	3 2024-11-24 2024-11-25
<input type="checkbox"/>	4	micromax	4000	2 2024-11-27 0000-00-00
<input type="checkbox"/>	5	Infinix	4000	2 2024-11-27 2024-11-28

Server: 127.0.0.1 » Database: laravel » Table: products

[Browse](#) [Structure](#) [SQL](#) [Search](#) [Insert](#) [Export](#) [Import](#)

```
SELECT * FROM `products`
```

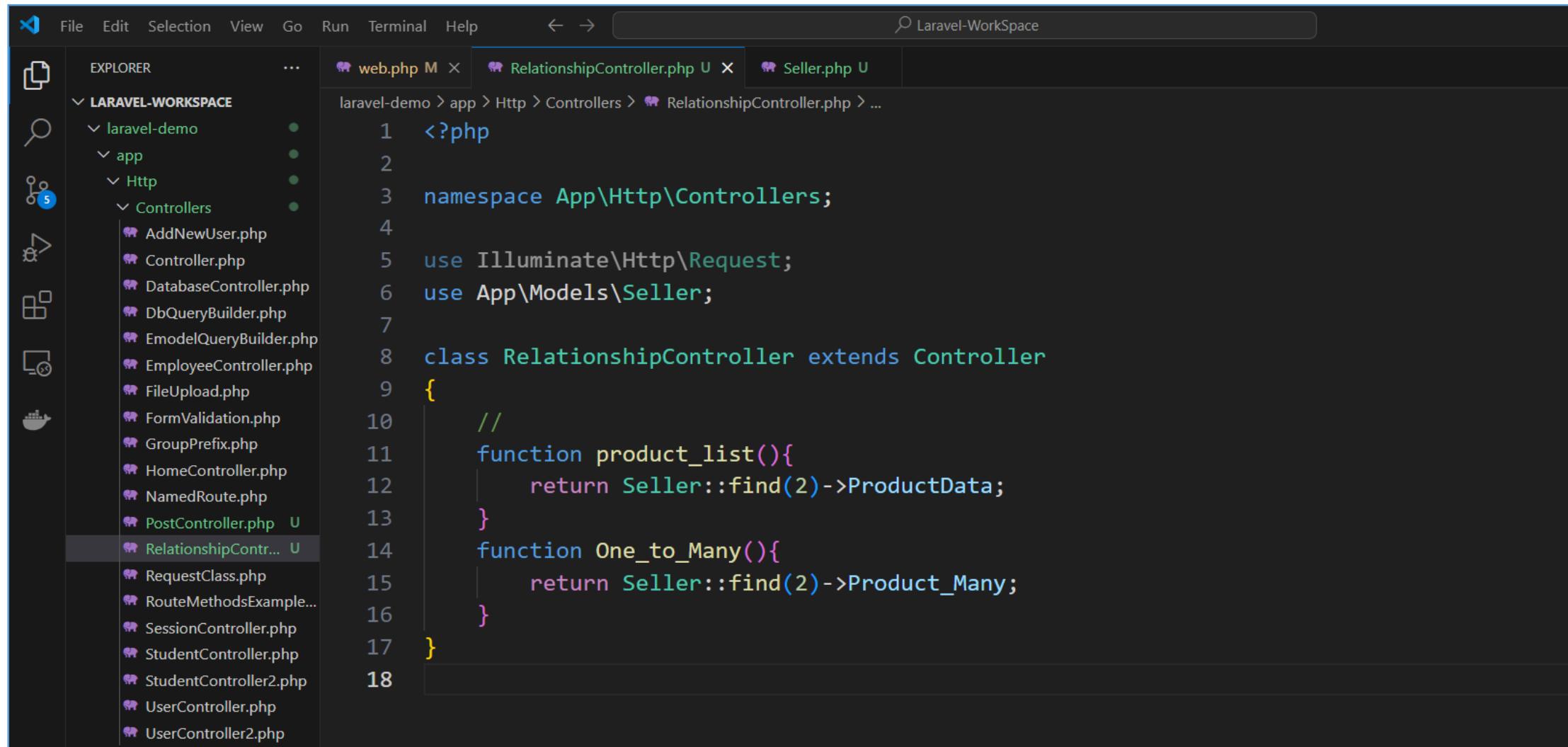
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key:

Extra options

	id	name	price	seller_id	created_at	updated_at
<input type="checkbox"/>	1	iphone	1000	1	2024-11-20	2024-11-21
<input type="checkbox"/>	2	samsung	2000	2	2024-11-22	2024-11-23
<input type="checkbox"/>	3	OOP	3000	3	2024-11-24	2024-11-25
<input type="checkbox"/>	4	micromax	4000	2	2024-11-27	0000-00-00
<input type="checkbox"/>	5	Infinix	4000	2	2024-11-27	2024-11-28

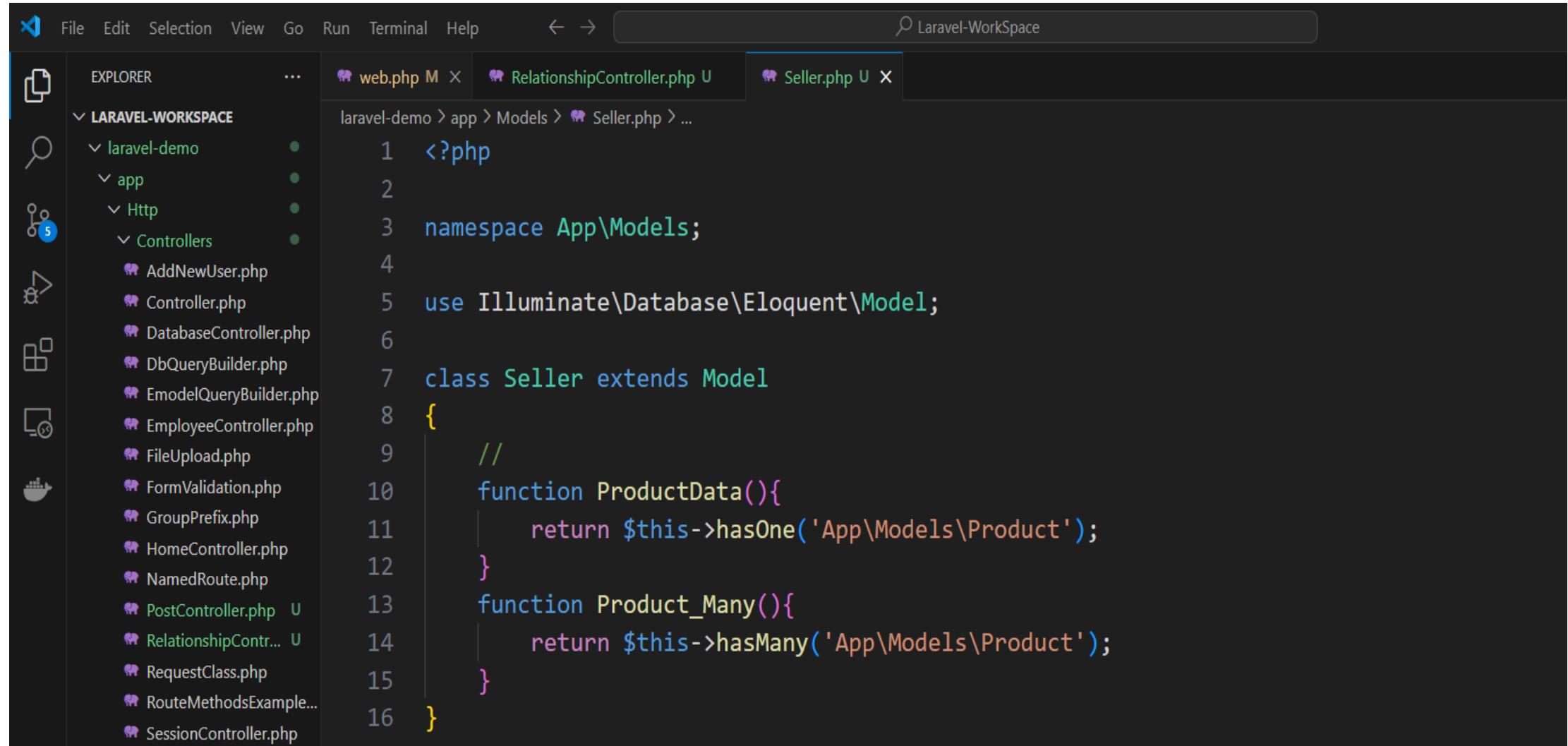
Controller (Relationship Controller)



The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists the project structure under 'LARAVEL-WORKSPACE' and 'laravel-demo'. The 'Controllers' folder contains several files, with 'RelationshipController.php' currently selected and highlighted in grey. The main editor area displays the code for 'RelationshipController.php':

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\Seller;
7
8 class RelationshipController extends Controller
9 {
10     //
11     function product_list(){
12         return Seller::find(2)->ProductData;
13     }
14     function One_to_Many(){
15         return Seller::find(2)->Product_Many;
16     }
17 }
18
```

Model (Seller)

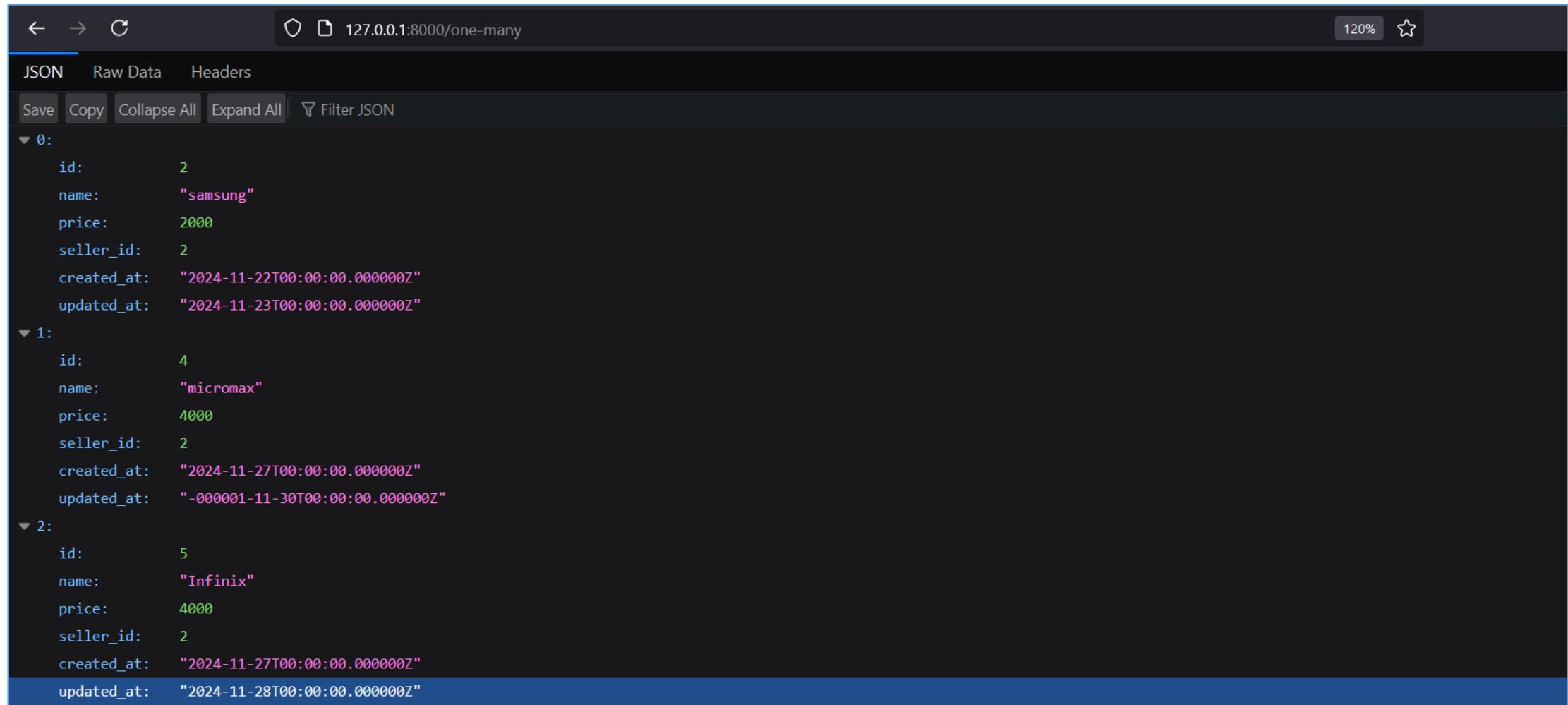


The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** Laravel-WorkSpace
- Explorer:** Shows the project structure under LARAVEL-WORKSPACE / laravel-demo / app / Models. The Seller.php file is selected.
- Code Editor:** Displays the contents of Seller.php:

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Model;
6
7 class Seller extends Model
8 {
9     //
10    function ProductData(){
11        return $this->hasOne('App\Models\Product');
12    }
13    function Product_Many(){
14        return $this->hasMany('App\Models\Product');
15    }
16 }
```

Output



The screenshot shows a browser interface for viewing JSON data. The address bar indicates the URL is 127.0.0.1:8000/one-many. The top navigation bar includes back, forward, and refresh buttons, along with zoom controls (120%) and a star icon. Below the address bar, there are tabs for "JSON", "Raw Data", and "Headers", with "JSON" being the active tab. A toolbar below the tabs contains buttons for "Save", "Copy", "Collapse All", "Expand All", and "Filter JSON". The main content area displays a JSON array with three elements, indexed 0, 1, and 2. Each element is an object with properties: id, name, price, seller_id, created_at, and updated_at.

```
[{"id": 2, "name": "samsung", "price": 2000, "seller_id": 2, "created_at": "2024-11-22T00:00:00.000000Z", "updated_at": "2024-11-23T00:00:00.000000Z"}, {"id": 4, "name": "micromax", "price": 4000, "seller_id": 2, "created_at": "2024-11-27T00:00:00.000000Z", "updated_at": "-000001-11-30T00:00:00.000000Z"}, {"id": 5, "name": "Infinix", "price": 4000, "seller_id": 2, "created_at": "2024-11-27T00:00:00.000000Z", "updated_at": "2024-11-28T00:00:00.000000Z"}]
```

Many to One

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists the project structure under 'LARAVEL-WORKSPACE'. The 'app' folder contains 'Http' and 'Controllers'. Under 'Controllers', 'RelationshipController.php' is selected and highlighted in blue. Other files listed include PostController.php, RequestClass.php, RouteMethodsExample..., SessionController.php, StudentController.php, StudentController2.php, UserController.php, UserController2.php, Middleware, Models (with employee.php, Product.php, Seller.php, Student.php, user.php), Providers, Rules, View, bootstrap, config, database, and lang. The main editor area shows the code for 'RelationshipController.php'. The code defines a class 'RelationshipController' that extends 'Controller'. It includes three methods: 'product_list()', 'One_to_Many()', and 'Many_to_One()'. The 'product_list()' method returns a seller with their products. The 'One_to_Many()' method returns all products for a specific seller. The 'Many_to_One()' method returns a single product with its seller information.

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\Seller;
7 use App\Models\Product;
8
9 class RelationshipController extends Controller
10 {
11     //
12     function product_list(){
13         return Seller::find(2)->ProductData;
14     }
15     function One_to_Many(){
16         return Seller::find(2)->Product_Many;
17     }
18     function Many_to_One(){
19         return Product::with('seller')->get();
20     }
21 }
22 }
```

Model Definition (Product)

The screenshot shows a code editor interface with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. A search bar on the right contains the text "Laravel-WorkSpace". The left sidebar has icons for Explorer, Search, Task List (with 5 items), and others. The main area displays the file structure under "LARAVEL-WORKSPACE/laravel-demo/app/Models":

```
laravel-demo > app > Models > Product.php > ...
```

The code itself is:

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Model;
6
7 class Product extends Model
8 {
9     //
10    function seller(){
11        return $this->belongsTo('App\Models\Seller');
12    }
13
14 }
```

The "Product.php" file is currently selected in the sidebar.

Route

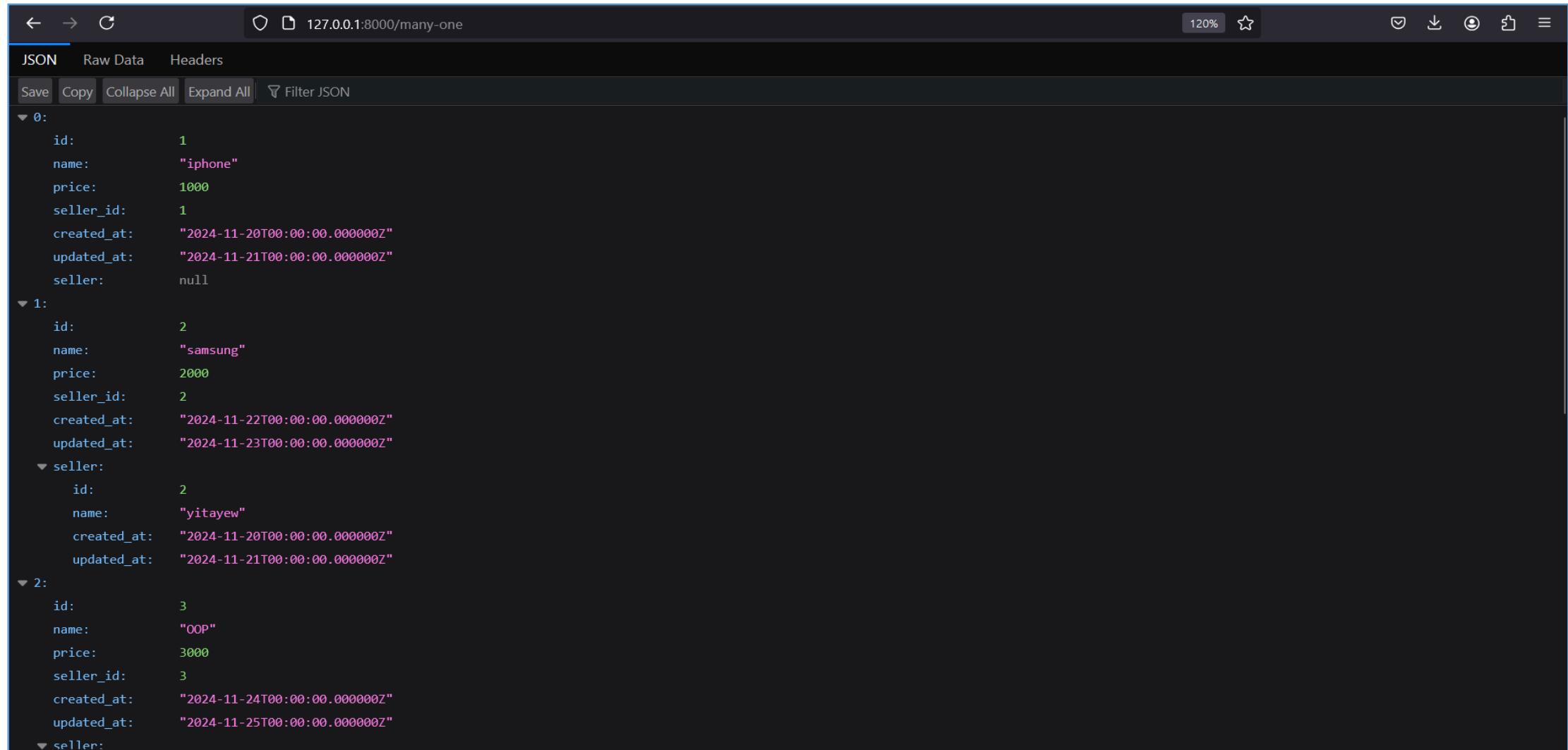
The screenshot shows a dark-themed interface of the Visual Studio Code (VS Code) code editor. At the top, there's a navigation bar with icons for File, Edit, Selection, View, Go, Run, Terminal, and Help. To the right of the bar is a search field containing "Laravel-WorkSpace". Below the navigation bar is a toolbar with several icons: a file icon, a magnifying glass, a circular progress bar with the number 5, a play/pause icon, a grid icon, a monitor icon, and a gear icon.

The main area is divided into three panes. On the left is the Explorer pane, which displays the project structure under "LARAVEL-WORKSPACE". It shows a folder named "laravel-demo" containing a "resources" folder with "views" and "blade" files like "search-employee.blade.php", "SessionView1.blade.php", "SessionView2.blade.php", "Students.blade.php", "UploadFile.blade.php", "url1.blade.php", "url2.blade.php", "userform.blade.php", "userform2.blade.php", and "welcome.blade.php". Below "views" is a "routes" folder containing "console.php" and "web.php". The status bar at the bottom indicates "M" next to "web.php".

The center and right panes show the content of the "web.php" file. The file path is listed as "laravel-demo > routes > web.php > ...". The code itself is:

```
190 // Relationship in Database
191 use App\Http\Controllers\RelationshipController;
192 // Route For One-to-One Relationship
193 Route::get('relationship',[RelationshipController::class,'product_list']);
194 // Route For One-to-Many Relationship
195 Route::get('one-many',[RelationshipController::class,'One_to_Many']);
196 // Route for Many-to-One Relationship
197 Route::get('many-one',[RelationshipController::class,'Many_to_One']);
198 
```

Output



A screenshot of a web browser displaying a JSON response from the URL `127.0.0.1:8000/many-one`. The browser interface includes a header with back, forward, and refresh buttons, a search bar, and a zoom level of 120%. Below the header is a toolbar with `Save`, `Copy`, `Collapse All`, `Expand All`, and `Filter JSON` buttons. The main content area shows a nested JSON structure representing three items:

```
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
▼ 0:
  id: 1
  name: "iphone"
  price: 1000
  seller_id: 1
  created_at: "2024-11-20T00:00:00.000000Z"
  updated_at: "2024-11-21T00:00:00.000000Z"
  seller: null
▼ 1:
  id: 2
  name: "samsung"
  price: 2000
  seller_id: 2
  created_at: "2024-11-22T00:00:00.000000Z"
  updated_at: "2024-11-23T00:00:00.000000Z"
▼ seller:
  id: 2
  name: "yitayew"
  created_at: "2024-11-20T00:00:00.000000Z"
  updated_at: "2024-11-21T00:00:00.000000Z"
▼ 2:
  id: 3
  name: "OOP"
  price: 3000
  seller_id: 3
  created_at: "2024-11-24T00:00:00.000000Z"
  updated_at: "2024-11-25T00:00:00.000000Z"
▼ seller:
```

Many-to-Many Relationship

3. Many-to-Many Relationship

- **Description:** Multiple records in one table are associated with multiple records in another table.
- **Example:** Users can have many roles, and roles can belong to many users.

Table Structure:

Users Table	Roles Table	role_user Table (Pivot)
<code>id</code> (PK)	<code>id</code> (PK)	<code>user_id</code> (FK)
<code>name</code>	<code>role_name</code>	<code>role_id</code> (FK)

Example (Model Definition)

Model Definition:

php

 Copy code

```
class User extends Model {  
    public function roles() {  
        return $this->belongsToMany(Role::class);  
    }  
}  
  
class Role extends Model {  
    public function users() {  
        return $this->belongsToMany(User::class);  
    }  
}
```

API in laravel

- An API (Application Programming Interface) in Laravel is a set of rules and endpoints that allows different software systems or applications to communicate with each other.
- Laravel provides robust tools for building RESTful APIs, making it easier to create, read, update, and delete resources over HTTP.

Core Features of Laravel APIs

Core Features of Laravel APIs

1. **Route Management:** Laravel allows defining API routes using `api.php`.
2. **Authentication:** Provides API token-based authentication through tools like Passport, Sanctum, or custom implementations.
3. **Validation:** Validates incoming requests using Laravel's built-in request validation.
4. **Eloquent Models:** Simplifies database interactions for API resources.
5. **Middleware:** Adds layers of security and processing to API routes.
6. **Resource Responses:** Formats API responses consistently using `Resources`.

Types of APIs in Laravel

Types of APIs in Laravel

1. RESTful APIs:

- Used for resource-based CRUD operations (Create, Read, Update, Delete).
- Example: Managing user data, products, etc.

2. Authentication APIs:

- Used for user login, logout, and token management.
- Example: Login APIs for mobile apps.

3. Third-Party APIs:

- Laravel applications can consume external APIs using HTTP clients like Guzzle or Laravel's HTTP client.



API Installation

The screenshot shows a terminal window within a code editor interface. The terminal tab is selected at the top. The command `$ php artisan install:api` is run, followed by the output of the composer update process. The right side of the terminal shows a sidebar with multiple terminal sessions, all labeled "bash" or "powershell".

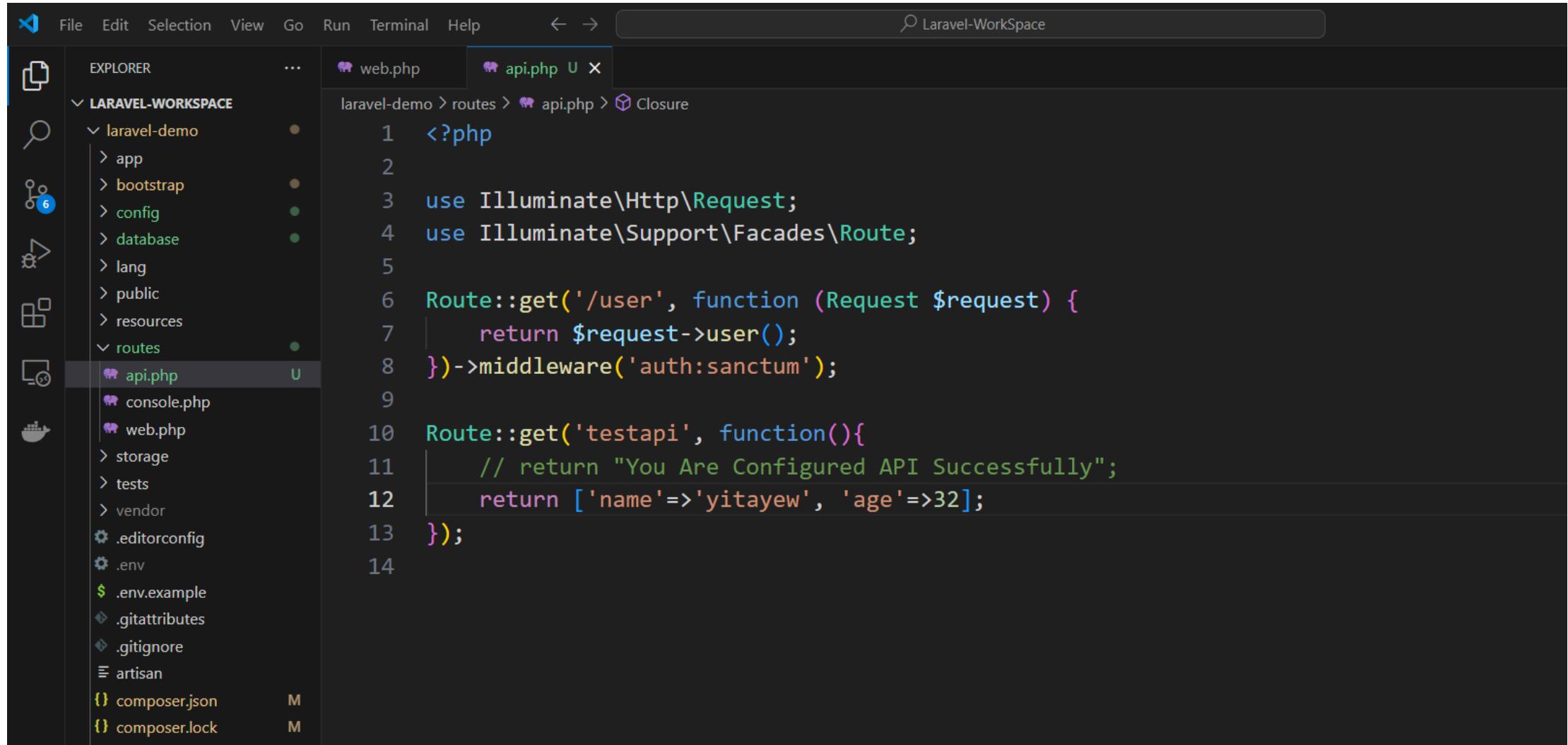
```
yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/laravel-demo (master)
$ php artisan install:api
./composer.json has been updated
Running composer update laravel/sanctum
Loading composer repositories with package information
Updating dependencies
Nothing to modify in lock file
Writing lock file
Installing dependencies from lock file (including require-dev)
Nothing to install, update or remove
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

INFO Discovering packages.

laravel/pail ...
laravel/sail ...
laravel/sanctum ...
laravel/tinker ...
nesbot/carbon ...
nunomaduro/collision ...
nunomaduro/termwind ...
pestphp/pest-plugin-laravel ...

85 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force
```

API (route/api.php)



The screenshot shows a dark-themed interface of the Visual Studio Code code editor. On the left is the Explorer sidebar, which lists the project structure of a Laravel application named "laravel-demo". The "routes" folder contains two files: "api.php" (which is currently selected) and "web.php". Other files visible include "app", "bootstrap", "config", "database", "lang", "public", "resources", ".editorconfig", ".env", ".env.example", ".gitattributes", ".gitignore", "artisan", "composer.json", and "composer.lock". The status bar at the bottom indicates "M" for modified status.

The main editor area displays the contents of the "api.php" file:

```
1 <?php
2
3 use Illuminate\Http\Request;
4 use Illuminate\Support\Facades\Route;
5
6 Route::get('/user', function (Request $request) {
7     return $request->user();
8 })->middleware('auth:sanctum');
9
10 Route::get('testapi', function(){
11     // return "You Are Configured API Successfully";
12     return ['name'=>'yitayew', 'age'=>32];
13 });
14
```

Output (<http://127.0.0.1:8000/api/testapi>)

The screenshot shows a browser window with two main sections. The top section displays a simple text message: "You Are Configured API Successfully". The bottom section is a JSON viewer for the same endpoint, showing the following data:

127.0.0.1:8000/api/testapi	
	Value
name:	"yitayew"
age:	32

The JSON viewer interface includes tabs for "JSON", "Raw Data", and "Headers", and buttons for "Save", "Copy", "Collapse All", "Expand All", and "Filter JSON".

Test API with VS Code Thunder Client

The screenshot shows the VS Code Marketplace interface. The search bar at the top contains the text "Laravel-WorkSpace". The left sidebar has a "thunder" filter applied, showing a list of extensions. The main panel displays the "Thunder Client" extension by Ranga Vadhineni, version v2.30.0. The extension is described as a "Lightweight Rest API Client for VS Code". It has 4,763,548 installs and a rating of 4.5 stars from 492 reviews. Buttons for "Disable", "Uninstall", and "Auto Update" are visible. Below the extension details, there's a section titled "Thunder Client" which describes it as a lightweight Rest API Client Extension for VS Code, hand-crafted by Ranga Vadhineni with a focus on simplicity, clean design and local storage. A bulleted list provides more information: Featured on Product Hunt, Featured in the "20 Fan Favorite Extensions" for VS Code, Website - www.thunderclient.com, Documentation: docs.thunderclient.com, and Support: github.com/rangav/thunder-client-support. At the bottom, there's a "Story behind Thunder Client" section and a link to "Read Launch Blog Post on Medium". The right sidebar lists "Categories" such as Programming Languages, Snippets, and Testing, and "Resources" including Marketplace, Issues, Repository, License, and Thunder Client. The status bar at the bottom shows "1 file 100%".

Output(<http://127.0.0.1:8000/api/testapi>)

The screenshot shows the Thunder Client application interface. The top navigation bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar labeled "Laravel-WorkSpace". The left sidebar has icons for file operations like New Request, Activity (which is selected), Collections, Env, and a history section with 6 items. The main workspace shows a request configuration for a GET request to "http://127.0.0.1:8000/api/testapi". The "Send" button is highlighted in blue. To the right, the response details are displayed: Status: 200 OK, Size: 27 Bytes, Time: 126 ms. The "Response" tab is selected, showing the JSON output:

```
1 {
2   "name": "yitayew",
3   "age": 32
4 }
```

Output (<http://127.0.0.1:8000/api/testapi2>)

The screenshot shows the Thunder Client application interface. On the left, there's a sidebar with various icons and a list of recent requests. The main area has tabs for 'web.php' and 'api.php', with 'api.php' currently selected. A search bar at the top right says 'Laravel-WorkSpace'. Below the tabs, a request configuration panel shows a 'GET' method and the URL 'http://127.0.0.1:8000/api/testapi2'. A large blue 'Send' button is to the right. Underneath, there are tabs for 'Query', 'Headers 2', 'Auth', 'Body', 'Tests', and 'Pre Run'. The 'Query' tab is active. To its right, the response details are displayed: 'Status: 404 Not Found', 'Size: 6.45 KB', and 'Time: 1.98 s'. The 'Response' tab is selected, showing the raw HTML code of the 404 error page. At the bottom, there are buttons for 'Response', 'Preview', and 'Chart'.

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6
7   <title>Not Found</title>
8
9   <style>
10    /*! normalize.css v8.0.1 | MIT License | github.com/necolas
11      /normalize.css */html{line-height:1.15;-webkit-text-size
12      -adjust:100%}body{margin:0}a{background-color
13      :transparent}code{font-family:monospace,monospace;font
14      -size:1em}[hidden]{display:none}html{font-family:system
15      -ui,-apple-system,BlinkMacSystemFont,Segoe UI,Roboto
16      ,Helvetica Neue,Arial,Noto Sans,sans-serif,Apple Color
17      Emoji,Segoe UI Emoji,Segoe UI Symbol,Noto Color Emoji
18      ;line-height:1.5}*,:after,:before{box-sizing:border-box
19      ;border:0 solid #e2e8f0}a{color:inherit;text-decoration
20      :inherit}code{font-family:Menlo,Monaco,Consolas
21      ,Liberation Mono,Courier New,monospace}svg,video{display
```

Make First GET API with Database

The screenshot shows a dark-themed instance of Visual Studio Code (VS Code) with the title bar "Laravel-WorkSpace". The left sidebar (Explorer) displays the project structure under "LARAVEL-WORKSPACE". The "routes" folder contains "api.php", "console.php", "web.php", and other files like "storage", "tests", "vendor", ".editorconfig", ".env", and ".env.example". The "api.php" file is open in the main editor, showing PHP code for defining routes:

```
<?php  
use Illuminate\Http\Request;  
use Illuminate\Support\Facades\Route;  
  
Route::get('/user', function (Request $request) {  
    return $request->user();  
})->middleware('auth:sanctum');  
  
Route::get('testapi', function(){  
    // return "You Are Configured API Successfully";  
    return [ 'name'=>'yitayew', 'age'=>32];  
});
```

The bottom right sidebar shows multiple terminal windows, all labeled "bash", indicating they are currently inactive.

The terminal pane at the bottom shows command-line output from artisan commands:

```
yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/laravel-demo (master)  
$ php artisan make:controller ApiController  
INFO Controller [c:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\app\Http\Controllers\ApiController.php] created successfully.  
  
yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/laravel-demo (master)  
$ php artisan make:model EmployeeAPI  
INFO Model [c:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\app\Models\EmployeeAPI.php] created successfully.
```

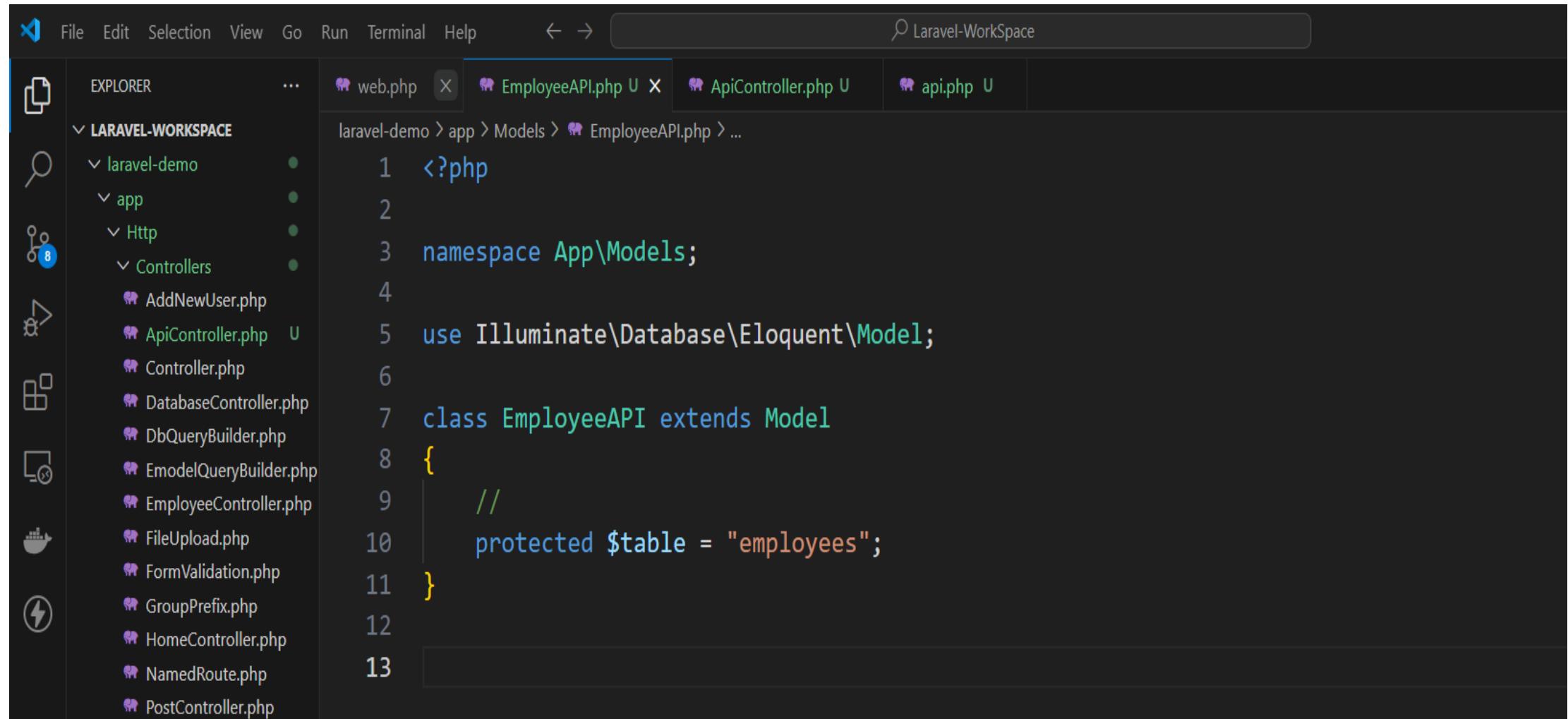
Controller (ApiController)

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** Laravel-WorkSpace
- Explorer:** Shows the project structure under LARAVEL-WORKSPACE / laravel-demo / app / Http / Controllers. The ApiController.php file is selected and highlighted in the list.
- Code Editor:** Displays the code for ApiController.php. The code defines a class ApiController that extends Controller. It contains a single function APIEmployee() which returns "Employee API is Called".

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\EmployeeAPI;
7
8 class ApiController extends Controller
9 {
10     //
11     function APIEmployee(){
12         // return "Employee API is Called";
13
14         return EmployeeAPI::all();
15     }
16 }
17
```

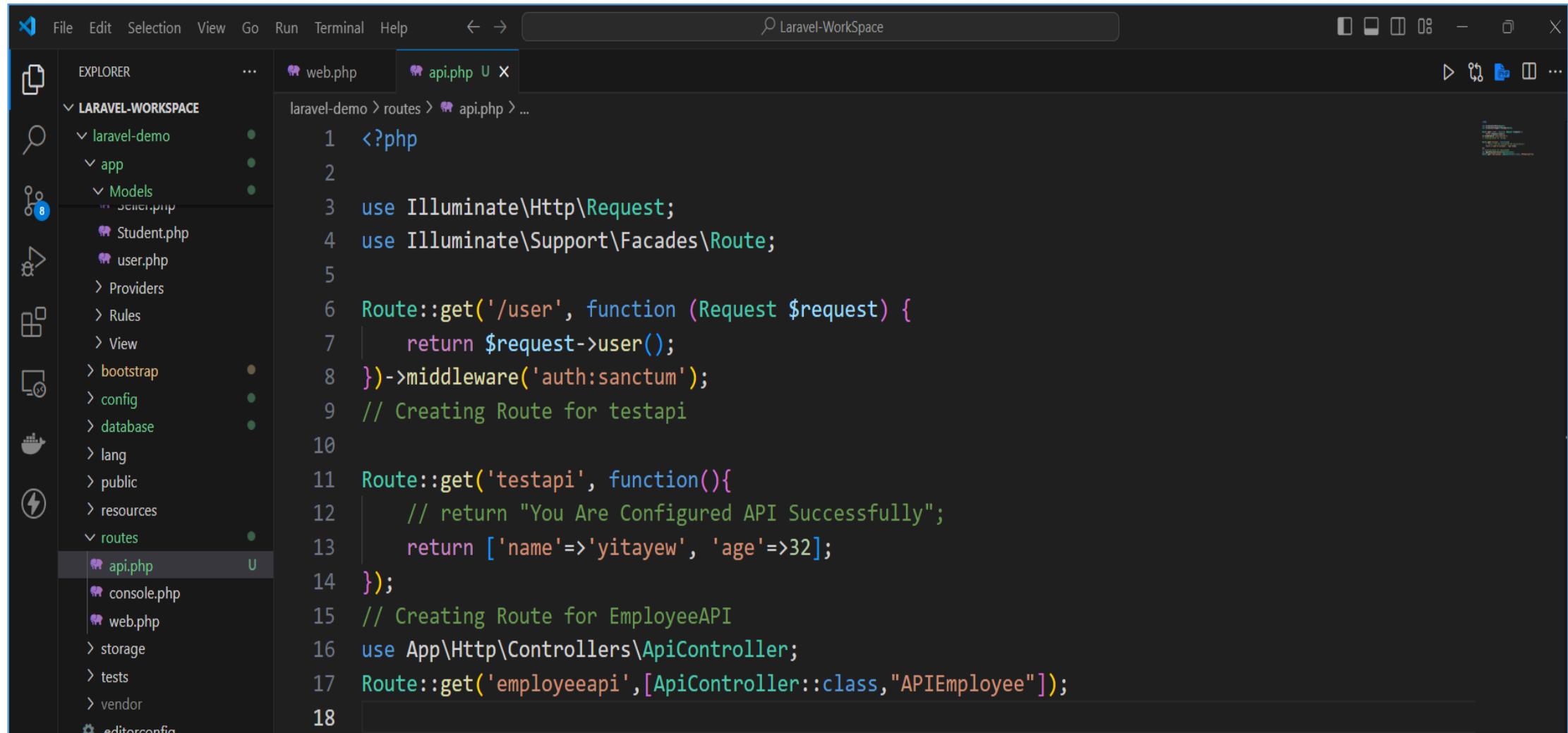
Model (EmployeeAPI)



The screenshot shows a code editor interface with a dark theme. The top navigation bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. A search bar on the right contains the text "Laravel-WorkSpace". The left sidebar has icons for Explorer, Search, Problems (with 8 items), and other development tools. The main area displays the EmployeeAPI.php file under the "laravel-demo > app > Models" directory. The code is as follows:

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Model;
6
7 class EmployeeAPI extends Model
8 {
9     //
10    protected $table = "employees";
11 }
12
13
```

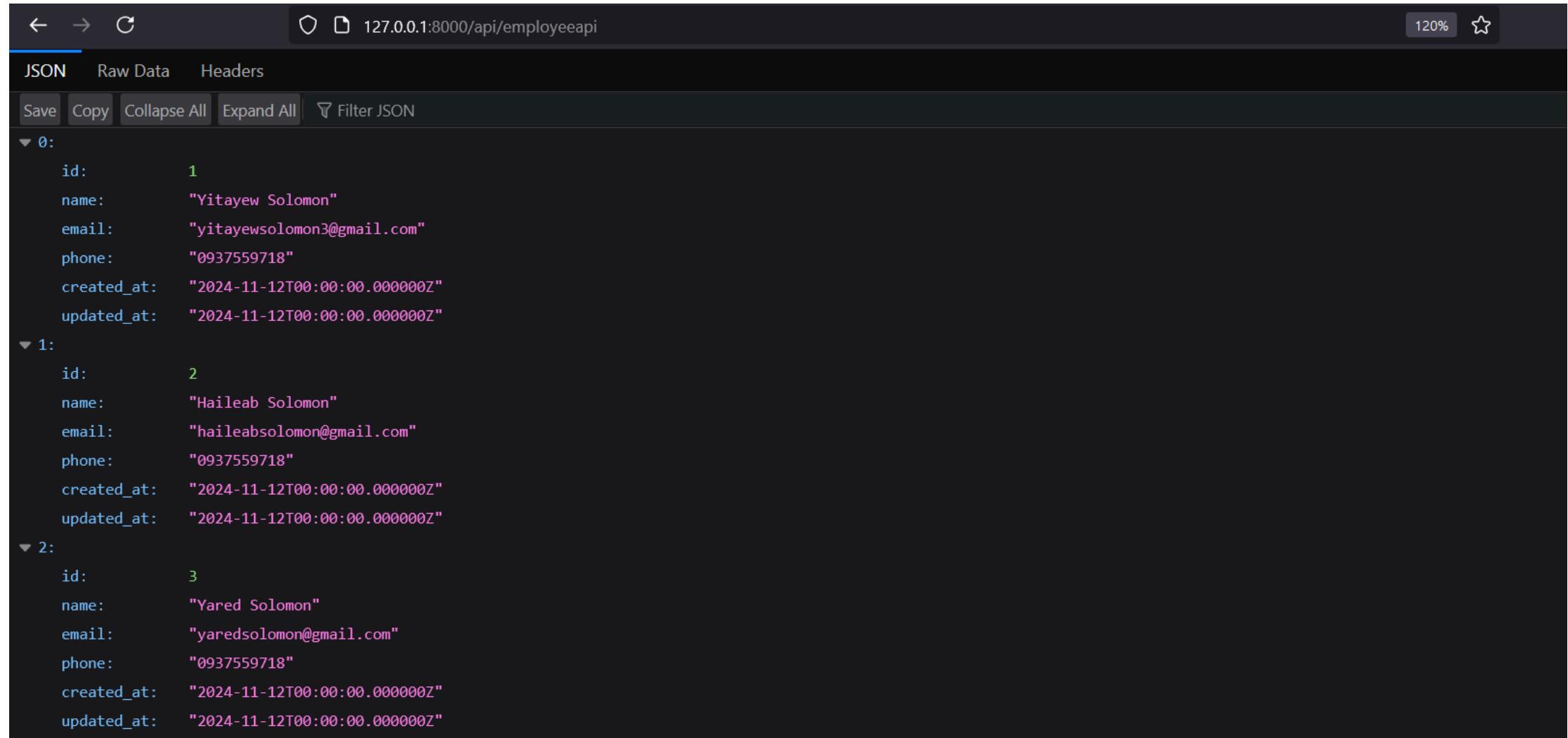
API Route



The screenshot shows a code editor interface with a dark theme. The title bar reads "Laravel-WorkSpace". The left sidebar is the "EXPLORER" panel, showing the project structure of "laravel-demo" with its subfolders like "app", "Models", "Providers", etc., and files like "Seller.php", "Student.php", "user.php", "api.php", "console.php", "web.php", and "editorconfig". The "api.php" file is currently selected and open in the main editor area. The code in "api.php" is as follows:

```
<?php  
use Illuminate\Http\Request;  
use Illuminate\Support\Facades\Route;  
  
Route::get('/user', function (Request $request) {  
    return $request->user();  
})->middleware('auth:sanctum');  
  
// Creating Route for testapi  
  
Route::get('testapi', function(){  
    // return "You Are Configured API Successfully";  
    return ['name'=>'yitayew', 'age'=>32];  
});  
  
// Creating Route for EmployeeAPI  
use App\Http\Controllers\ApiController;  
Route::get('employeeapi',[ApiController::class,"APIEmployee"]);
```

Output (<http://127.0.0.1:8000/api/employeeapi>)



A screenshot of a web browser displaying the JSON output of an API endpoint. The URL in the address bar is `127.0.0.1:8000/api/employeeapi`. The browser interface includes standard navigation buttons (back, forward, search) and a zoom level of 120%. Below the address bar, there are tabs for "JSON", "Raw Data", and "Headers", with "JSON" being the active tab. A toolbar below the tabs includes "Save", "Copy", "Collapse All", "Expand All", and a "Filter JSON" button. The main content area shows a JSON array with three elements (0, 1, 2), each representing an employee with fields: id, name, email, phone, created_at, and updated_at.

```
[{"id": 1, "name": "Yitayew Solomon", "email": "yitayewsolomon@gmail.com", "phone": "0937559718", "created_at": "2024-11-12T00:00:00.000000Z", "updated_at": "2024-11-12T00:00:00.000000Z"}, {"id": 2, "name": "Haileab Solomon", "email": "haileabsolomon@gmail.com", "phone": "0937559718", "created_at": "2024-11-12T00:00:00.000000Z", "updated_at": "2024-11-12T00:00:00.000000Z"}, {"id": 3, "name": "Yared Solomon", "email": "yaredsolomon@gmail.com", "phone": "0937559718", "created_at": "2024-11-12T00:00:00.000000Z", "updated_at": "2024-11-12T00:00:00.000000Z"}]
```

API Methods in Laravel

- Laravel supports all major HTTP methods for building APIs. These methods allow developers to perform CRUD (Create, Read, Update, Delete) operations and more. Here's a breakdown of the methods Laravel supports, their purposes, and descriptions:

GET Method

1. GET

- **Purpose:** Retrieve data from the server.
- **Example:** Fetching a list of users or a single user by ID.

php

 Copy code

```
Route::get('/users', [UserController::class, 'index']); // Fetch all users  
Route::get('/users/{id}', [UserController::class, 'show']); // Fetch a user by ID
```

Post Method

2. POST

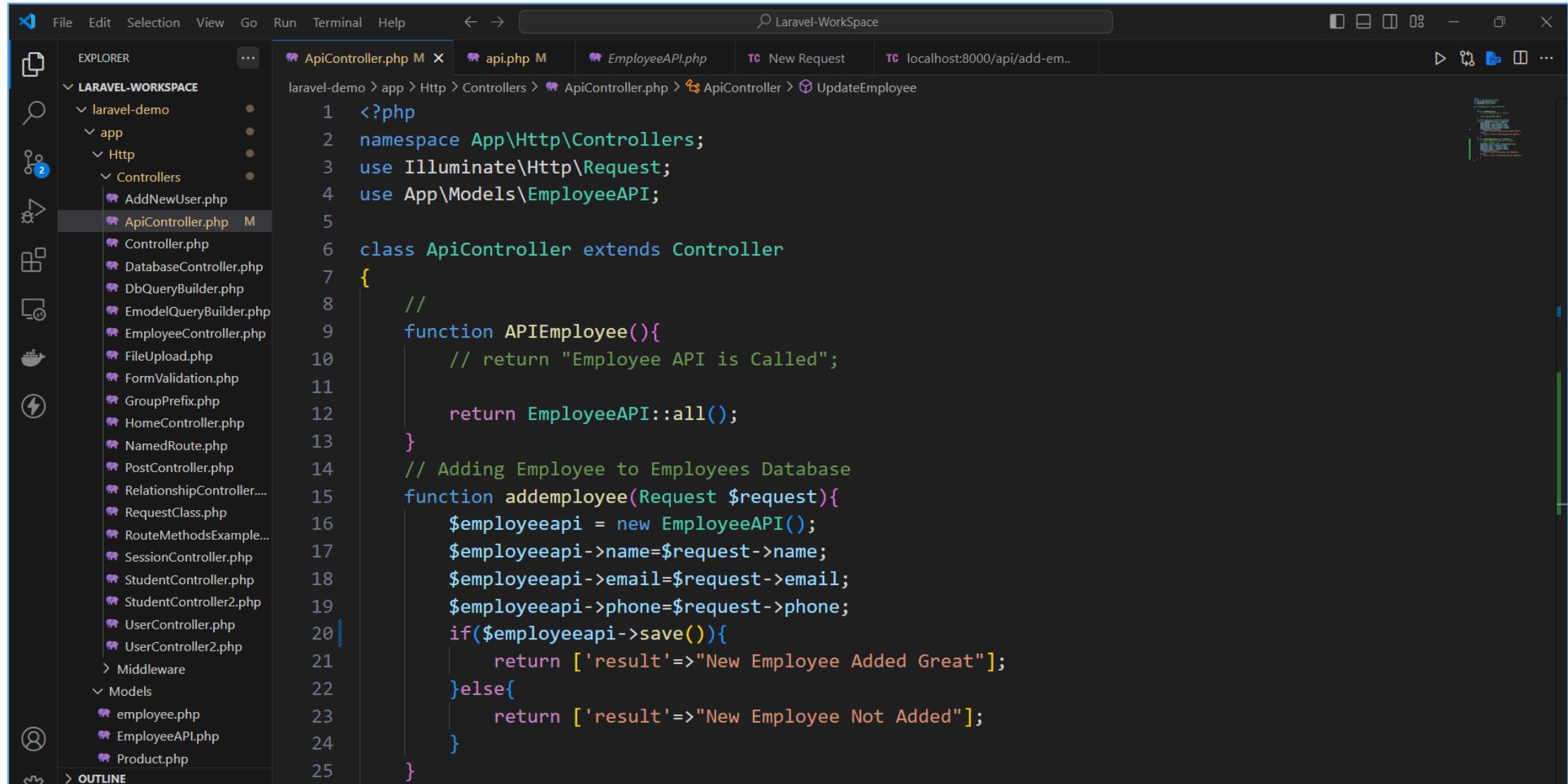
- **Purpose:** Create new resources on the server.
- **Example:** Adding a new user to the database.

php

 Copy code

```
Route::post('/users', [UserController::class, 'store']); // Create a new user
```

Example (APIController)



The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with an orange border. The title bar says "Laravel-WorkSpace". The left sidebar has icons for Explorer, Search, Problems, and Activity. The Explorer view shows a file tree under "LARAVEL-WORKSPACE/laravel-demo/app/Http/Controllers". The file "ApiController.php" is selected and highlighted in blue. The main editor area displays the following PHP code:

```
<?php  
namespace App\Http\Controllers;  
use Illuminate\Http\Request;  
use App\Models\EmployeeAPI;  
  
class ApiController extends Controller  
{  
    //  
    function APIEmployee(){  
        // return "Employee API is Called";  
  
        return EmployeeAPI::all();  
    }  
    // Adding Employee to Employees Database  
    function addemployee(Request $request){  
        $employeeapi = new EmployeeAPI();  
        $employeeapi->name=$request->name;  
        $employeeapi->email=$request->email;  
        $employeeapi->phone=$request->phone;  
        if($employeeapi->save()){  
            return ['result'=>"New Employee Added Great"];  
        }else{  
            return ['result'=>"New Employee Not Added"];  
        }  
    }  
}
```

Model (EmployeeAPI)

The screenshot shows the Visual Studio Code interface with the title bar "Laravel-WorkSpace". The left sidebar is the Explorer view, showing the project structure under "LARAVEL-WORKSPACE" for the "laravel-demo" application. The "Models" folder contains several files: employee.php, EmployeeAPI.php (which is currently selected and has a green background), Product.php, Seller.php, Student.php, and user.php. The main editor area displays the code for the EmployeeAPI.php model:

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Model;
6
7 class EmployeeAPI extends Model
8 {
9     //
10    protected $table = "employees";
11 }
12
13 
```

Route (API)

The screenshot shows a dark-themed interface of the Visual Studio Code (VS Code) code editor. The title bar reads "Laravel-WorkSpace". The left sidebar is the "EXPLORER" view, showing the project structure of a Laravel application named "laravel-demo". The "routes" folder contains three files: "api.php" (which is currently selected and has a green status bar indicator), "console.php", and "web.php". Other files like ".env", ".editorconfig", and ".gitignore" are also listed. The main editor area displays the content of "api.php".

```
1 <?php
2
3 use Illuminate\Http\Request;
4 use Illuminate\Support\Facades\Route;
5
6 Route::get('/user', function (Request $request) {
7     return $request->user();
8 })->middleware('auth:sanctum');
9 // Creating Route for testapi
10
11 Route::get('testapi', function(){
12     // return "You Are Configured API Successfully";
13     return [ 'name'=>'yitayew', 'age'=>32];
14 });
15 // Creating Route for EmployeeAPI
16 use App\Http\Controllers\ApiController;
17 Route::get('employeeapi',[ApiController::class,"APIEmployee"]);
18 Route::post('add-employeeapi',[ApiController::class,'adddemployee']);
```

Sending Add Request

The screenshot shows the Thunder Client application interface. The top navigation bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar says "Laravel-WorkSpace". The left sidebar has icons for THUNDER CLIENT, Activity (which is selected), Collections, Env, filter activity, POST requests, GET requests, and other tools. The main area shows a "New Request" dialog with a POST method, URL "localhost:8000/api/add-employeeapi", and a "Send" button. Below it, tabs for Query, Headers (2), Auth, Body (1), Tests, and Pre Run are visible, with "Body" being the active tab. The "Body" tab shows JSON content with the following payload:

```
1 {
2   "name": "yitayew",
3   "email": "yitayew@gmail.com",
4   "phone": "1122334455"
5 }
```

The response panel on the right displays the results of the request. It shows Status: 200 OK, Size: 37 Bytes, and Time: 141 ms. The Response tab shows the JSON output:

```
1 {
2   "result": "New Employee Added Great"
3 }
```

Output (Database Table)

			id	name	email	phone	created_at	updated_at
← T →		▼						
<input type="checkbox"/>	 Edit	 Copy	 Delete	1	Yitayew Solomon	yitayewsolomon3@gmail.com	0937559718	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	2	Haileab Solomon	haileabsolomon@gmail.com	0937559718	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	3	Yared Solomon	yaredsolomon@gmail.com	0937559718	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	6	Natanim Yitayew	natan@gmail.com	0937559720	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	7	Dureti Guye	dure@gmail.com	0910010203	2024-11-13
<input type="checkbox"/>	 Edit	 Copy	 Delete	12	yitayew	yitayew@gmail.com	1122334455	2024-11-25

PUT Method

3. PUT

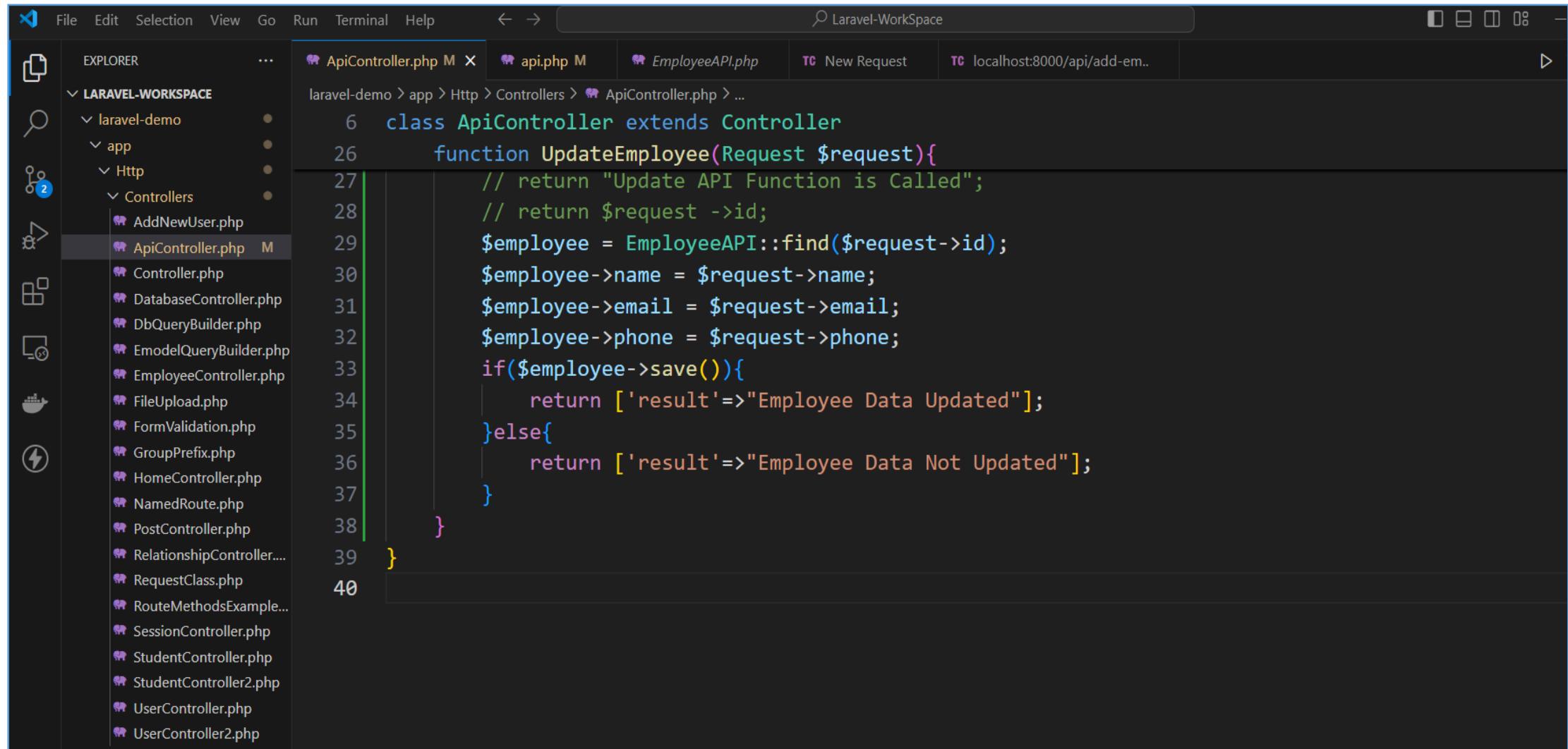
- **Purpose:** Update an existing resource completely.
- **Example:** Updating all fields of a user.

php

 Copy code

```
Route::put('/users/{id}', [UserController::class, 'update']); // Update a user
```

Example (APIController)



The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists the project structure under 'LARAVEL-WORKSPACE'. The 'laravel-demo' folder contains an 'app' folder, which has 'Http' and 'Controllers' subfolders. Inside 'Controllers', several files are listed: 'AddNewUser.php', 'ApiController.php' (which is currently selected), 'Controller.php', 'DatabaseController.php', 'DbQueryBuilder.php', 'EmodelQueryBuilder.php', 'EmployeeController.php', 'FileUpload.php', 'FormValidation.php', 'GroupPrefix.php', 'HomeController.php', 'NamedRoute.php', 'PostController.php', 'RelationshipController....', 'RequestClass.php', 'RouteMethodsExample...', 'SessionController.php', 'StudentController.php', 'StudentController2.php', 'UserController.php', and 'UserController2.php'. The main editor area displays the 'ApiController.php' file. The code is as follows:

```
6 class ApiController extends Controller
26     function UpdateEmployee(Request $request){
27         // return "Update API Function is Called";
28         // return $request->id;
29         $employee = EmployeeAPI::find($request->id);
30         $employee->name = $request->name;
31         $employee->email = $request->email;
32         $employee->phone = $request->phone;
33         if($employee->save()){
34             return ['result'=>"Employee Data Updated"];
35         }else{
36             return ['result'=>"Employee Data Not Updated"];
37         }
38     }
39 }
40 }
```

Model (EmployeeAPI)

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** Laravel-WorkSpace.
- Left Sidebar (Explorer):** Shows the project structure under "LARAVEL-WORKSPACE".
 - laravel-demo (selected):
 - app
 - Http
 - Controllers
 - PostController.php
 - RelationshipController....
 - RequestClass.php
 - RouteMethodsExample...
 - SessionController.php
 - StudentController.php
 - StudentController2.php
 - UserController.php
 - UserController2.php
 - Middleware
 - Models
 - employee.php
 - EmployeeAPI.php (selected)
 - Product.php
 - Seller.php
 - Student.php
 - user.php
 - Central Area:** The "EmployeeAPI.php" file is open in the editor.

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Model;
6
7 class EmployeeAPI extends Model
8 {
9     //
10    protected $table = "employees";
11 }
12
13 
```
 - Top Status Bar:** Shows tabs for api.php, ApiController.php, New Request, and EmployeeAPI.php.

Route

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists the project structure of a Laravel workspace named 'laravel-demo'. The 'routes' folder contains three files: 'api.php' (selected), 'console.php', and 'web.php'. The main editor area displays the contents of 'api.php'.

```
1 <?php
2
3 use Illuminate\Http\Request;
4 use Illuminate\Support\Facades\Route;
5
6 Route::get('/user', function (Request $request) {
7     return $request->user();
8 })->middleware('auth:sanctum');
9 // Creating Route for testapi
10
11 Route::get('testapi', function(){
12     // return "You Are Configured API Successfully";
13     return ['name'=>'yitayew', 'age'=>32];
14 });
15 // Creating Route for EmployeeAPI
16 use App\Http\Controllers\ApiController;
17 Route::get('employeeapi',[ApiController::class,"APIEmployee"]);
18 Route::post('add-employeeapi',[ApiController::class,'adddemployee']);
19 Route::put('update-employee',[ApiController::class,'UpdateEmployee']);
```

Sending Update Request

The screenshot shows a Laravel workspace in a code editor. The left sidebar displays the project structure under 'LARAVEL-WORKSPACE' with 'laravel-demo' selected. The 'Controllers' folder contains several files: PostController.php, RelationshipController..., RequestClass.php, RouteMethodsExample..., SessionController.php, StudentController.php, StudentController2.php, UserController.php, and UserController2.php. The 'Models' folder contains employee.php, EmployeeAPI.php (which is currently selected), Product.php, Seller.php, Student.php, and user.php.

In the center, a 'New Request' tab is open, showing a PUT request to `http://127.0.0.1:8000/api/update-employee`. The 'Body' tab is selected, showing JSON content:

```
1  {
2    "id" : 12,
3    "name": "yitayew",
4    "email" : "yitayewsolomon3@gmail.com",
5    "phone": "000000"
6  }
```

The response on the right shows a successful 200 OK status with a size of 34 bytes and a time of 150 ms. The response body is:

```
1  {
2    "result": "Employee Data Updated"
3  }
```

Output (Database Table)

			id	name	email	phone	created_at	updated_at
<input type="checkbox"/>	 Edit	 Copy	 Delete	1 Yitayew Solomon	yitayewsolomon3@gmail.com	0937559718	2024-11-12	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	2 Haileab Solomon	haileabsolomon@gmail.com	0937559718	2024-11-12	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	3 Yared Solomon	yaredsolomon@gmail.com	0937559718	2024-11-12	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	6 Natanim Yitayew	natan@gmail.com	0937559720	2024-11-12	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	7 Dureti Guye	dure@gmail.com	0910010203	2024-11-13	2024-11-13
<input type="checkbox"/>	 Edit	 Copy	 Delete	12 yitayew	yitayewsolomon3@gmail.com	000000	2024-11-25	2024-11-25
	<input type="checkbox"/> Check all	With selected:	 Edit	 Copy	 Delete	 Export		

PATCH Method

4. PATCH

- **Purpose:** Update specific fields of an existing resource.
- **Example:** Updating only the email of a user.

php

 Copy code

```
Route::patch('/users/{id}', [UserController::class, 'update']); // Partial update
```

DELETE Method

5. DELETE

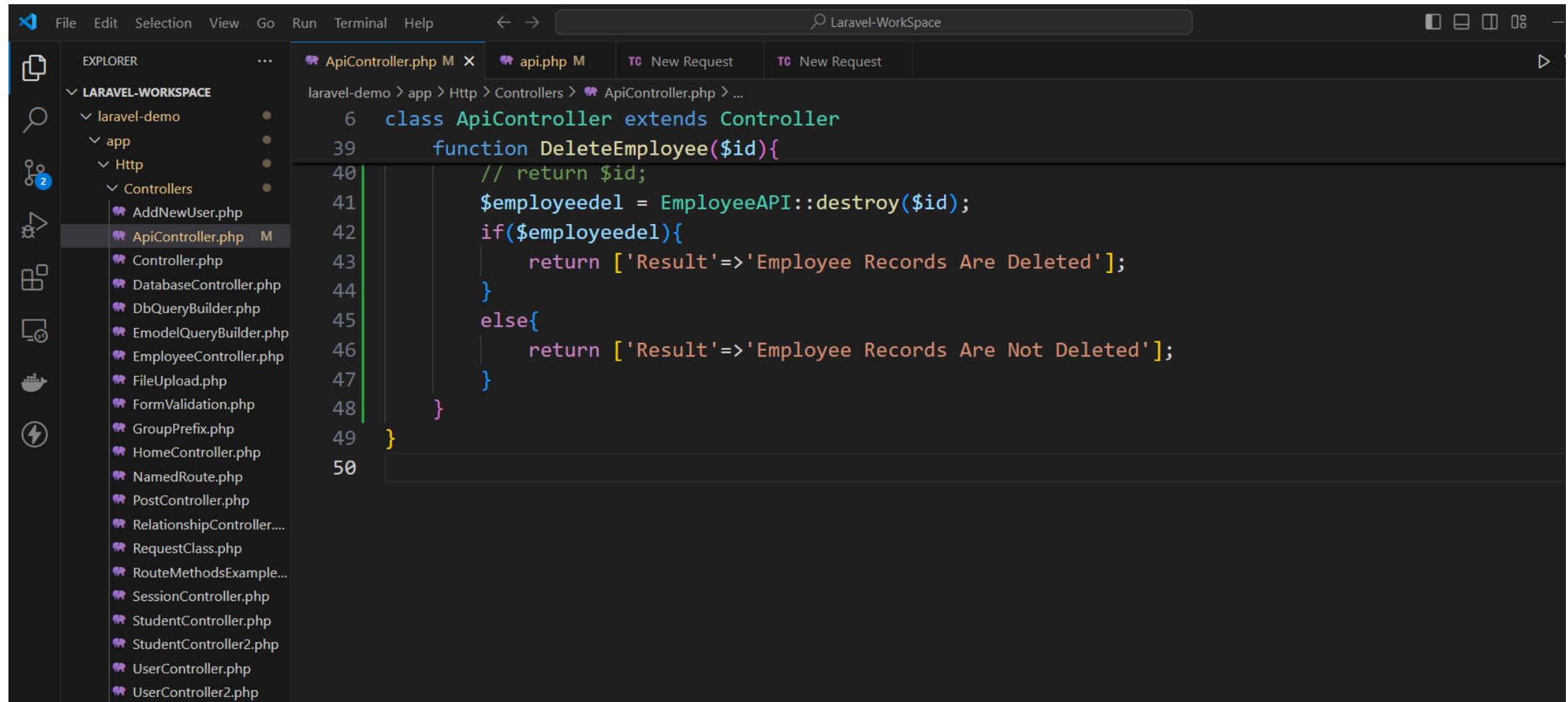
- **Purpose:** Delete a resource from the server.
- **Example:** Removing a user from the database.

php

 Copy code

```
Route::delete('/users/{id}', [UserController::class, 'destroy']); // Delete a user
```

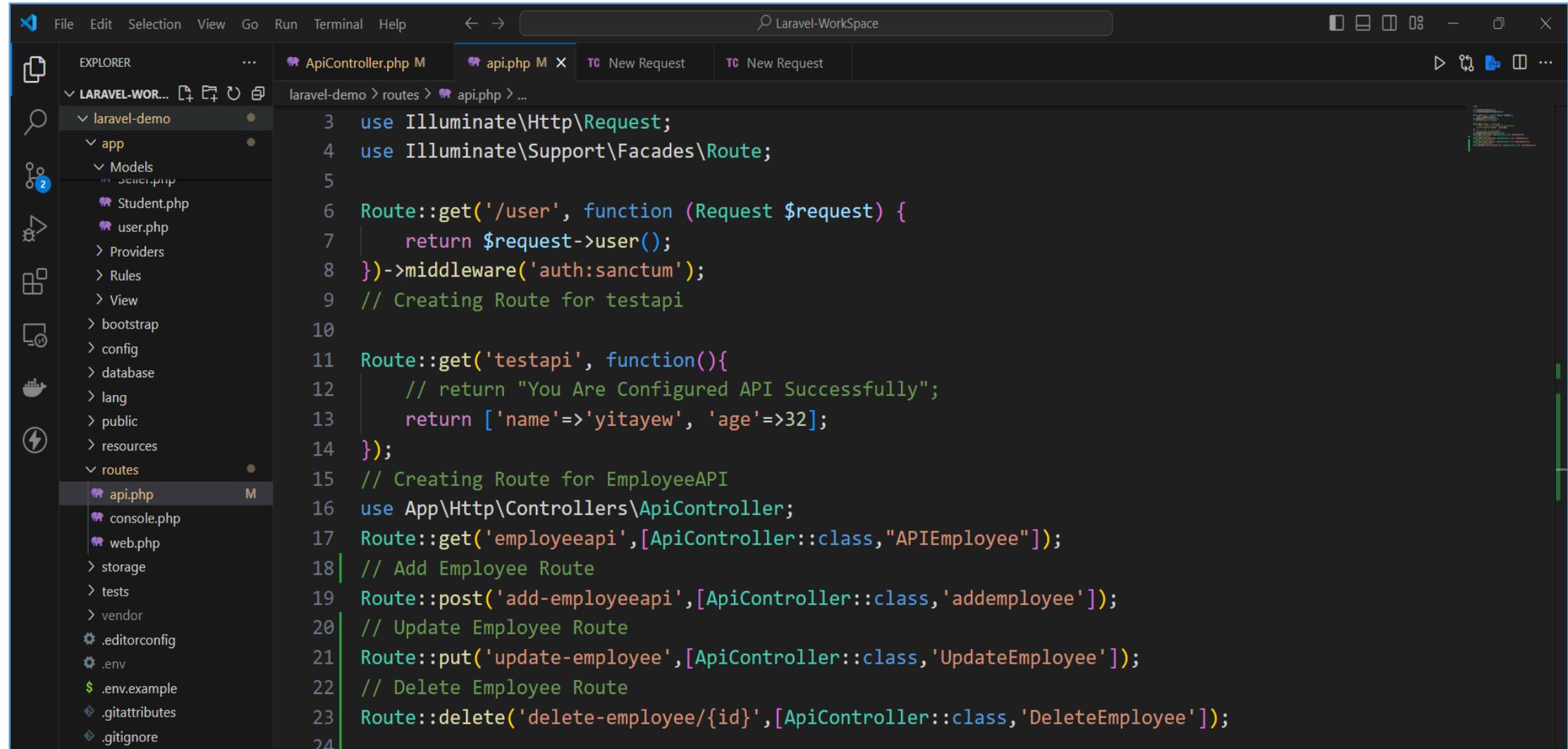
Example



The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists the project structure under 'LARAVEL-WORKSPACE'. The 'laravel-demo' project contains an 'app' folder with 'Http' and 'Controllers' subfolders. Inside 'Controllers', several files are listed, including 'AddNewUser.php', 'ApiController.php' (which is currently selected), 'Controller.php', 'DatabaseController.php', 'DbQueryBuilder.php', 'EmodelQueryBuilder.php', 'EmployeeController.php', 'FileUpload.php', 'FormValidation.php', 'GroupPrefix.php', 'HomeController.php', 'NamedRoute.php', 'PostController.php', 'RelationshipController...', 'RequestClass.php', 'RouteMethodsExample...', 'SessionController.php', 'StudentController.php', 'StudentController2.php', 'UserController.php', and 'UserController2.php'. The main editor area displays the 'ApiController.php' file with the following code:

```
6 class ApiController extends Controller
39     function DeleteEmployee($id){
40         // return $id;
41         $employeedel = EmployeeAPI::destroy($id);
42         if($employeedel){
43             return ['Result'=>'Employee Records Are Deleted'];
44         }
45         else{
46             return ['Result'=>'Employee Records Are Not Deleted'];
47         }
48     }
49 }
50 }
```

Route



The screenshot shows a code editor interface with a dark theme. The left sidebar contains a file tree for a Laravel project named 'laravel-demo'. The 'routes' directory is expanded, showing files like 'api.php', 'console.php', and 'web.php'. The 'api.php' file is currently selected and open in the main editor area.

```
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

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

// Creating Route for testapi

Route::get('testapi', function(){
    // return "You Are Configured API Successfully";
    return ['name'=>'yitayew', 'age'=>32];
});

// Creating Route for EmployeeAPI
use App\Http\Controllers\ApiController;
Route::get('employeeapi',[ApiController::class,"APIEmployee"]);
// Add Employee Route
Route::post('add-employeeapi',[ApiController::class,'addemployee']);
// Update Employee Route
Route::put('update-employee',[ApiController::class,'UpdateEmployee']);
// Delete Employee Route
Route::delete('delete-employee/{id}',[ApiController::class,'DeleteEmployee']);
```

Sending Delete Request

The screenshot shows the Thunder Client interface. In the top navigation bar, there are tabs for 'ApiController.php M', 'api.php M', 'New Request', and another 'New Request' tab which is currently active. The main area displays a 'Query' tab for a DELETE request to 'http://127.0.0.1:8000/api/delete-employee/12'. The response status is '200 OK', size is '41 Bytes', and time is '203 ms'. The response body is a JSON object: { "Result": "Employee Records Are Deleted" }. On the left sidebar, under the 'Activity' section, there is a log entry for a DELETE request to '127.0.0.1:8000/api/delete-employee/12' made 11 mins ago.

This screenshot shows the same Thunder Client interface as the previous one, but with a different response. The DELETE request to 'http://127.0.0.1:8000/api/delete-employee/12' resulted in a '200 OK' status, '45 Bytes' size, and '203 ms' time. The response body is a JSON object: { "Result": "Employee Records Are Not Deleted" }. The left sidebar's activity log shows the same DELETE request from 12 mins ago, but the most recent log entry is now a PUT request to '127.0.0.1:8000/api/upda...' made 12 mins ago.

Output (Database Table)

			id	name	email	phone	created_at	updated_at
<input type="checkbox"/>	 Edit	 Copy	 Delete	1 Yitayew Solomon	yitayewsolomon3@gmail.com	0937559718	2024-11-12	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	2 Haileab Solomon	haileabsolomon@gmail.com	0937559718	2024-11-12	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	3 Yared Solomon	yaredsolomon@gmail.com	0937559718	2024-11-12	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	6 Natanim Yitayew	natan@gmail.com	0937559720	2024-11-12	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	7 Dureti Guye	dure@gmail.com	0910010203	2024-11-13	2024-11-13
	<input type="checkbox"/> Check all	With selected:	 Edit	 Copy	 Delete	 Export		

OPTION Method

6. OPTIONS

- **Purpose:** Retrieve communication options available for the target resource.
- **Example:** Typically used in CORS (Cross-Origin Resource Sharing) preflight requests.

php

 Copy code

```
Route::options('/users', function () {
    return response()->json(['GET', 'POST', 'OPTIONS']);
});
```

HEAD Method

7. HEAD

- **Purpose:** Retrieve metadata about a resource without the actual content.
- **Example:** Checking headers to verify if a resource exists.

php

Copy code

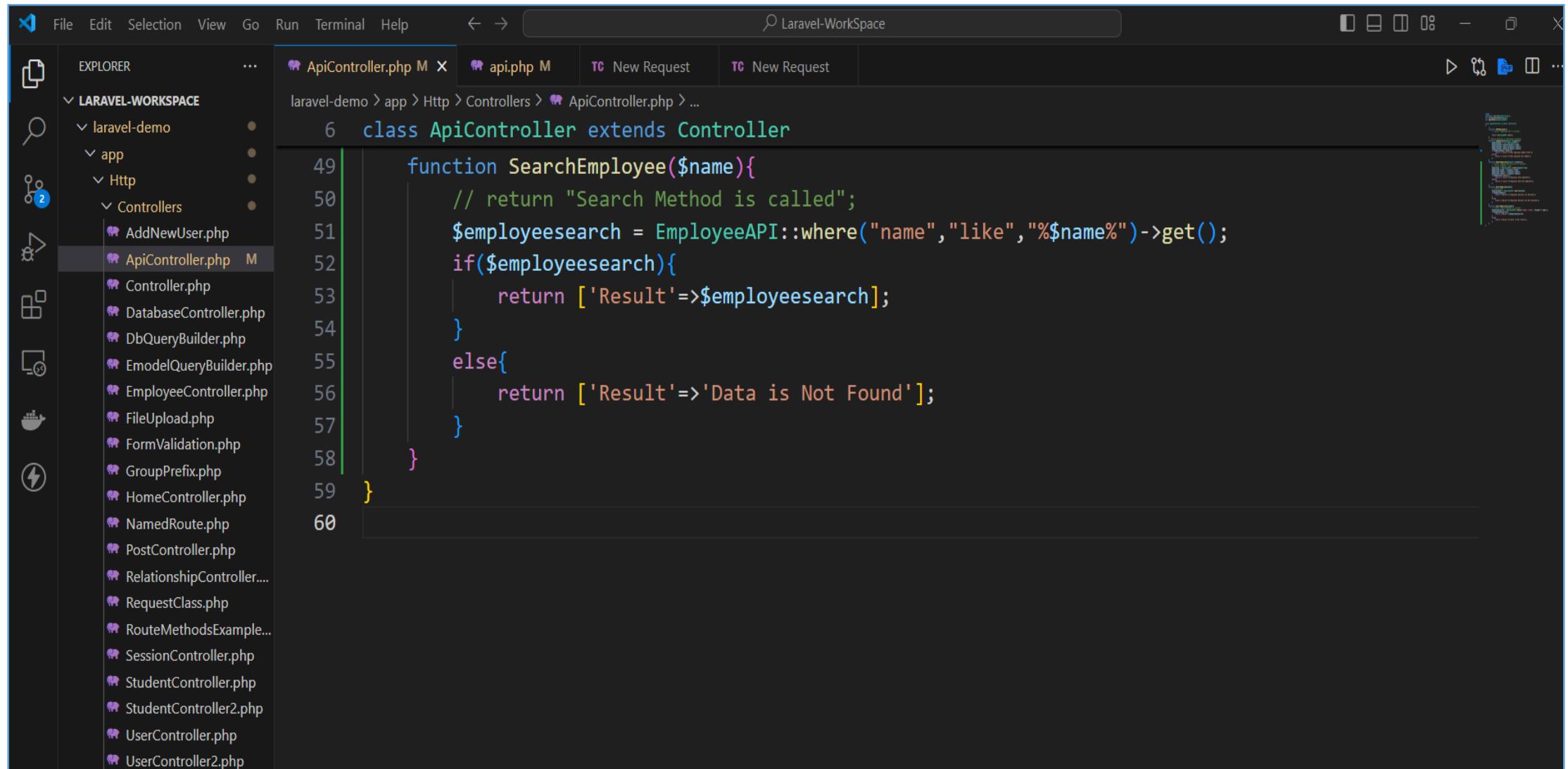
```
Route::get('/users', [UserController::class, 'index'])->middleware('head');
```

How Laravel Handles These Methods

Laravel uses **Route definitions** to handle these methods. The `Route` facade supports methods for each HTTP verb, as shown above.



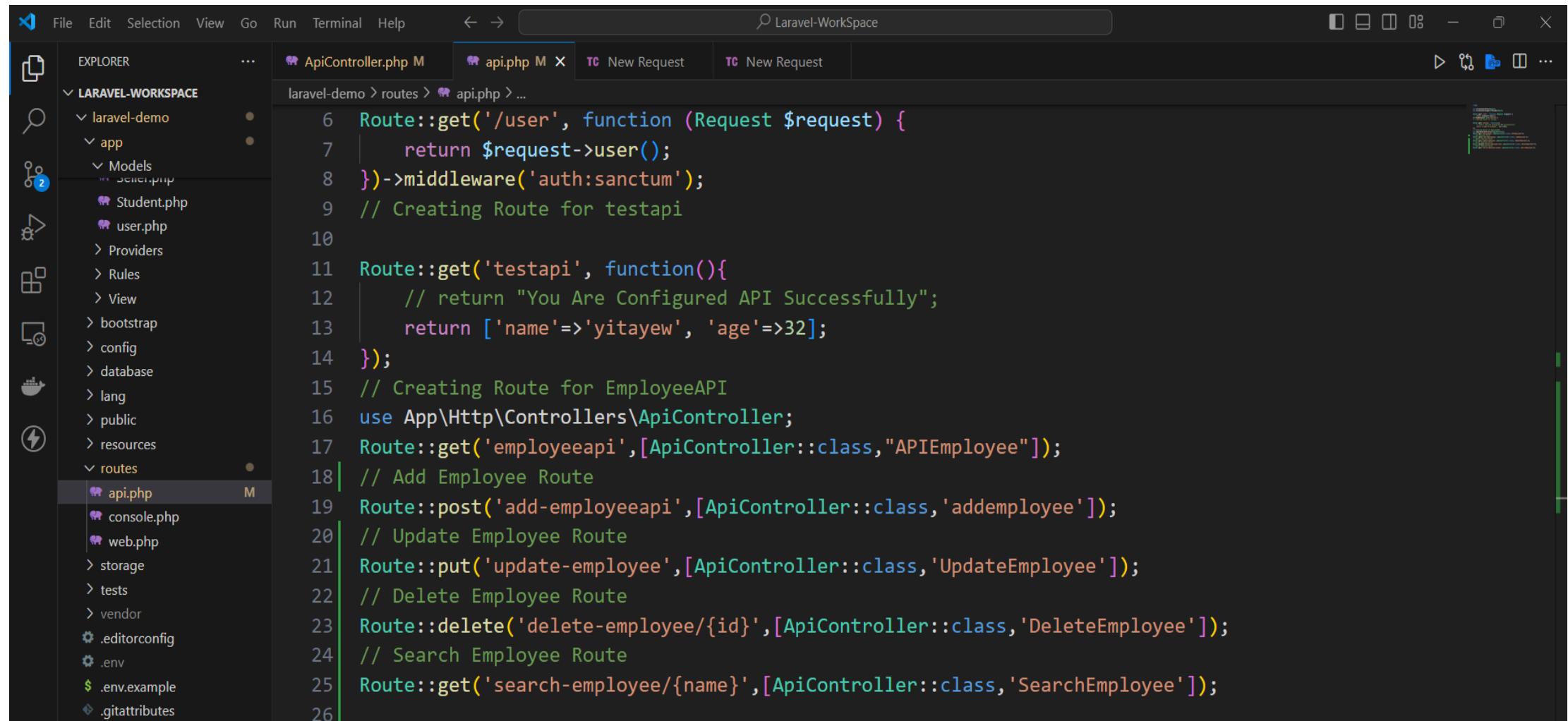
Search API



The screenshot shows a code editor interface with the title "Laravel-WorkSpace". The left sidebar contains a file tree under "EXPLORER" labeled "LARAVEL-WORKSPACE". The "laravel-demo" project is expanded, showing the "app" directory, which further contains "Http" and "Controllers". Inside "Controllers", several files are listed, including "AddNewUser.php", "ApiController.php" (which is currently selected and highlighted in blue), "Controller.php", "DatabaseController.php", "DbQueryBuilder.php", "EmodelQueryBuilder.php", "EmployeeController.php", "FileUpload.php", "FormValidation.php", "GroupPrefix.php", "HomeController.php", "NamedRoute.php", "PostController.php", "RelationshipController....", "RequestClass.php", "RouteMethodsExample...", "SessionController.php", "StudentController.php", "StudentController2.php", "UserController.php", and "UserController2.php". The main editor area displays the "ApiController.php" code:

```
6 class ApiController extends Controller
49
50     function SearchEmployee($name){
51         // return "Search Method is called";
52         $employeesearch = EmployeeAPI::where("name","like","%$name%")->get();
53         if($employeesearch){
54             return ['Result'=>$employeesearch];
55         }
56         else{
57             return ['Result'=>'Data is Not Found'];
58         }
59     }
60 }
```

Route



The screenshot shows a code editor interface with a dark theme. The title bar reads "Laravel-WorkSpace". The left sidebar is titled "EXPLORER" and shows the project structure under "LARAVEL-WORKSPACE". The "routes" folder contains three files: "api.php" (selected), "console.php", and "web.php". The "api.php" file is open in the main editor area, displaying the following PHP code:

```
Route::get('/user', function (Request $request) {
    return $request->user();
})->middleware('auth:sanctum');
// Creating Route for testapi
Route::get('testapi', function(){
    // return "You Are Configured API Successfully";
    return [ 'name'=>'yitayew', 'age'=>32];
});
// Creating Route for EmployeeAPI
use App\Http\Controllers\ApiController;
Route::get('employeeapi',[ApiController::class,"APIEmployee"]);
// Add Employee Route
Route::post('add-employeeapi',[ApiController::class,'addemployee']);
// Update Employee Route
Route::put('update-employee',[ApiController::class,'UpdateEmployee']);
// Delete Employee Route
Route::delete('delete-employee/{id}',[ApiController::class,'DeleteEmployee']);
// Search Employee Route
Route::get('search-employee/{name}',[ApiController::class,'SearchEmployee']);
```

Sending Search Request

The screenshot shows a Laravel development environment in a dark-themed IDE. The top navigation bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar says "Laravel-WorkSpace". The left sidebar has an Explorer section with a tree view of a Laravel project named "laravel-demo". The "routes" folder contains "api.php", "console.php", and "web.php". The main workspace shows a "New Request" tab with a GET request to "http://127.0.0.1:8000/api/search-employee/yitayew". The "Query" tab is selected, showing a "Query Parameters" table with a single row: "parameter" (checkbox) and "value" (text input). The "Send" button is highlighted in blue. To the right, the response panel shows the status: "Status: 200 OK", "Size: 356 Bytes", and "Time: 223 ms". The "Response" tab displays the JSON output:

```
1  {
2      "Result": [
3          {
4              "id": 1,
5              "name": "Yitayew Solomon",
6              "email": "yitayewsolomon3@gmail.com",
7              "phone": "0937559718",
8              "created_at": "2024-11-12T00:00:00.000000Z",
9              "updated_at": "2024-11-12T00:00:00.000000Z"
10         },
11         {
12             "id": 6,
13             "name": "Natanim Yitayew",
14             "email": "natan@gmail.com",
15             "phone": "0937559720",
16             "created_at": "2024-11-12T00:00:00.000000Z",
17             "updated_at": "2024-11-12T00:00:00.000000Z"
18         }
19     ]
20 }
```

Output (Database Table)

			id	name	email	phone	created_at	updated_at
<input type="checkbox"/>	 Edit	 Copy	 Delete	1	Yitayew Solomon	yitayewsolomon3@gmail.com	0937559718	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	2	Haileab Solomon	haileabsolomon@gmail.com	0937559718	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	3	Yared Solomon	yaredsolomon@gmail.com	0937559718	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	6	Natanim Yitayew	natan@gmail.com	0937559720	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	7	Dureti Guye	dure@gmail.com	0910010203	2024-11-13
 <input type="checkbox"/> Check all	<input type="checkbox"/> With selected:	 Edit	 Copy	 Delete	 Export			
<input type="checkbox"/> Show all	Number of rows:	25	Filter rows:	Search this table	Sort by key:	None		

Summary

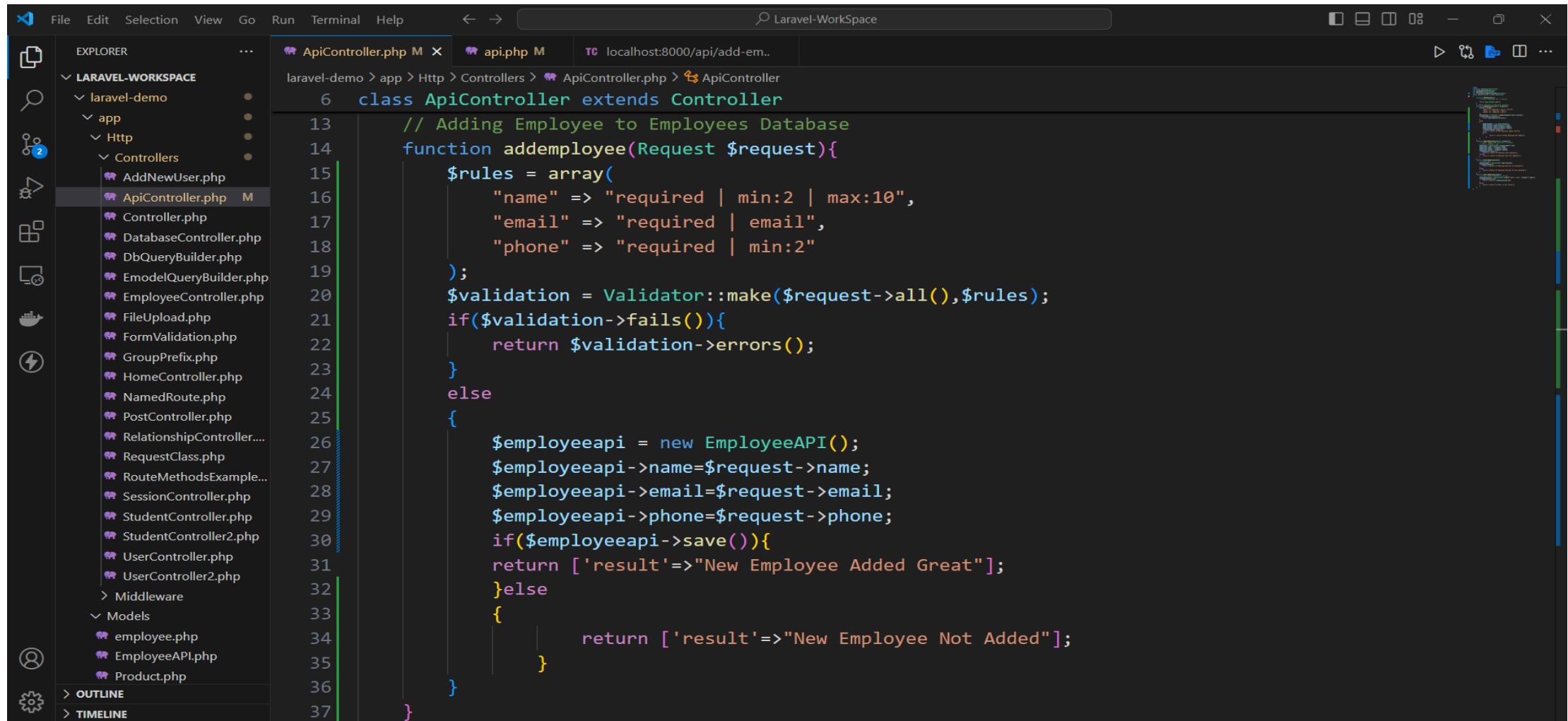
Summary Table of HTTP Methods

HTTP Method	Purpose	Laravel Route Example	Controller Method
GET	Retrieve data	<code>Route::get('/users')</code>	<code>index()</code> / <code>show()</code>
POST	Create a new resource	<code>Route::post('/users')</code>	<code>store()</code>
PUT	Update a resource fully	<code>Route::put('/users/{id}')</code>	<code>update()</code>
PATCH	Update specific fields	<code>Route::patch('/users/{id}')</code>	<code>update()</code>
DELETE	Remove a resource	<code>Route::delete('/users/{id}')</code>	<code>destroy()</code>
OPTIONS	Fetch communication options	<code>Route::options('/users')</code>	N/A
HEAD	Fetch resource metadata	<code>Route::get('/users')->middleware('head')</code>	N/A

API Validation

- API validation in Laravel is used to ensure that the data provided in API requests meets specific rules and constraints before processing the request.
- Laravel provides a clean and robust way to validate API data using the validate method or form request classes.

Example Applying Validation on add Employee

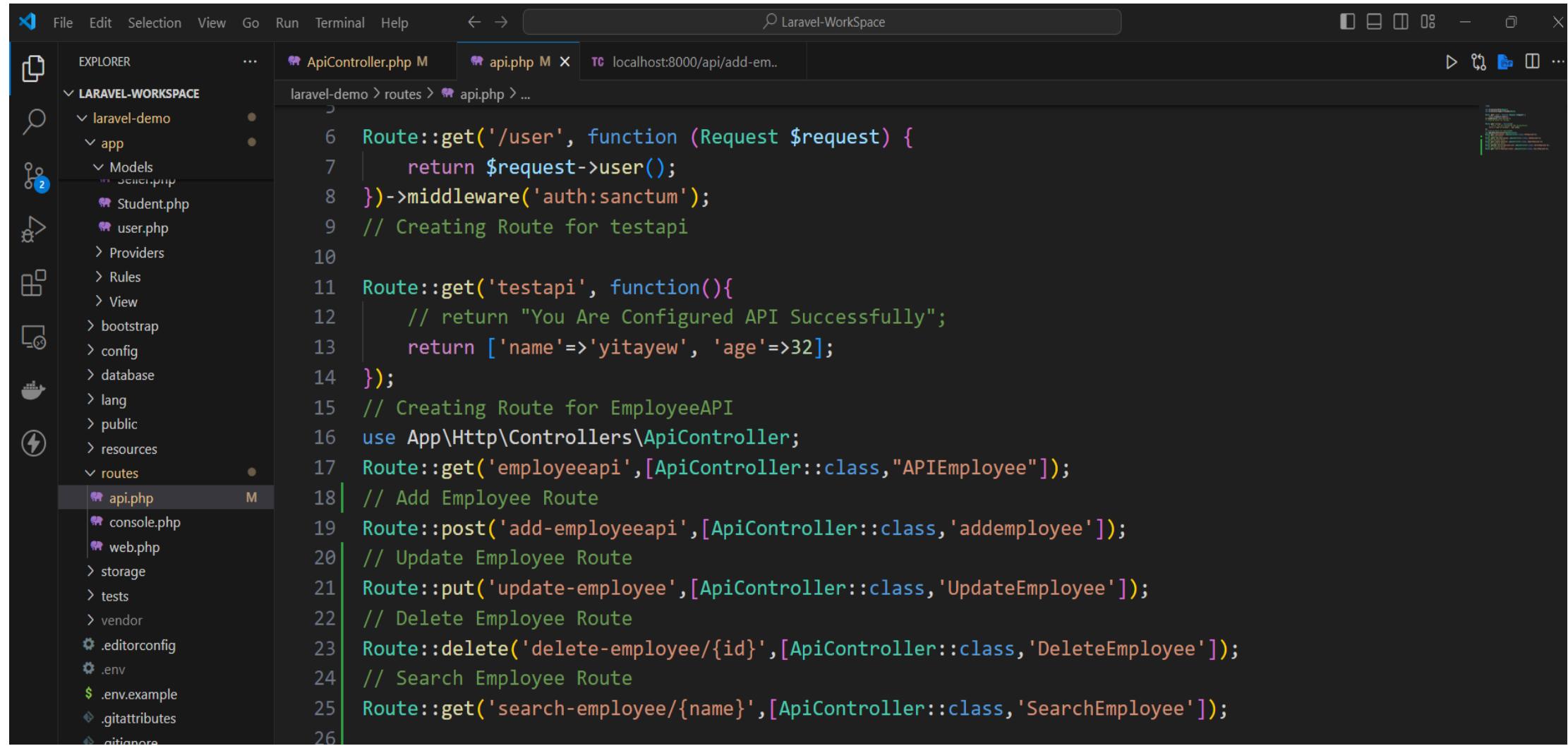


The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists the project structure under 'LARAVEL-WORKSPACE'. The current file is 'ApiController.php' located in the 'Http\Controllers' directory. The main editor area displays the following PHP code:

```
6 class ApiController extends Controller
13     // Adding Employee to Employees Database
14     function addemployee(Request $request){
15         $rules = array(
16             "name" => "required | min:2 | max:10",
17             "email" => "required | email",
18             "phone" => "required | min:2"
19         );
20         $validation = Validator::make($request->all(),$rules);
21         if($validation->fails()){
22             return $validation->errors();
23         }
24         else
25         {
26             $employeeapi = new EmployeeAPI();
27             $employeeapi->name=$request->name;
28             $employeeapi->email=$request->email;
29             $employeeapi->phone=$request->phone;
30             if($employeeapi->save()){
31                 return [ 'result'=>"New Employee Added Great"];
32             }
33             else
34             {
35                 return [ 'result'=>"New Employee Not Added"];
36             }
37         }
38     }
39 }
```

The code implements a validation rule for adding an employee. It checks for required fields ('name', 'email', 'phone') and their lengths. If validation fails, it returns the errors. If successful, it creates a new instance of the 'EmployeeAPI' class and saves the employee. If saving fails, it returns an error message.

Route



The screenshot shows a code editor interface with an orange header bar. The title bar says "Laravel-WorkSpace". The left sidebar has icons for File, Edit, Selection, View, Go, Run, Terminal, and Help. The Explorer panel shows a file tree for a "laravel-demo" workspace, including "app", "routes", and various PHP files like "Seller.php", "Student.php", "user.php", and "ApiController.php". The main editor area displays the "api.php" file with the following code:

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

// Creating Route for testapi

Route::get('testapi', function(){
    // return "You Are Configured API Successfully";
    return ['name'=>'yitayew', 'age'=>32];
});

// Creating Route for EmployeeAPI
use App\Http\Controllers\ApiController;
Route::get('employeeapi',[ApiController::class,"APIEmployee"]);

// Add Employee Route
Route::post('add-employeeapi',[ApiController::class,'addemployee']);

// Update Employee Route
Route::put('update-employee',[ApiController::class,'UpdateEmployee']);

// Delete Employee Route
Route::delete('delete-employee/{id}',[ApiController::class,'DeleteEmployee']);

// Search Employee Route
Route::get('search-employee/{name}',[ApiController::class,'SearchEmployee']);
```

Sending Add Request

The screenshot shows the Postman application interface with two requests sent to the endpoint `localhost:8000/api/add-employeeapi`.

Request 1: Status: 200 OK | Size: 122 Bytes | Time: 128 ms

Body (JSON):

```
1 {
2   "name": [
3     "The name field is required."
4   ],
5   "email": [
6     "The email field is required."
7   ],
8   "phone": [
9     "The phone field is required."
10 ]
11 }
```

Request 2: Status: 200 OK | Size: 37 Bytes | Time: 158 ms

Body (JSON):

```
1 {
2   "result": "New Employee Added Great"
3 }
```

The left sidebar shows the Laravel workspace structure, including the `laravel-demo` project and its `app`, `routes`, and `api.php` files.

Output (Database Table)

			id	name	email	phone	created_at	updated_at
<input type="checkbox"/>	 Edit	 Copy	 Delete	1 Yitayew Solomon	yitayewsolomon3@gmail.com	0937559718	2024-11-12	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	2 Haileab Solomon	haileabsolomon@gmail.com	0937559718	2024-11-12	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	3 Yared Solomon	yaredsolomon@gmail.com	0937559718	2024-11-12	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	6 Natanim Yitayew	natan@gmail.com	0937559720	2024-11-12	2024-11-12
<input type="checkbox"/>	 Edit	 Copy	 Delete	7 Dureti Guye	dure@gmail.com	0910010203	2024-11-13	2024-11-13
<input type="checkbox"/>	 Edit	 Copy	 Delete	13 Fassil	fasil@gmail.com	112233	2024-11-25	2024-11-25
<input type="checkbox"/>	 Edit	 Copy	 Delete	15 Fassil	fassil@gmail.com	0011223344	2024-11-25	2024-11-25
	<input type="checkbox"/> Check all	With selected:	 Edit	 Copy	 Delete	 Export		
<input type="checkbox"/> Show all	Number of rows:	25	Filter rows:	Search this table	Sort by key:	None		

API with Resource Controller

- Using a Resource Controller in Laravel for API development is an efficient way to handle standard CRUD operations. A Resource Controller provides predefined methods corresponding to common actions like creating, retrieving, updating, and deleting records.

Example

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with a dark theme. The left sidebar contains a file tree under 'LARAVEL-WORKSPACE' for a project named 'laravel-demo'. The 'Controllers' folder is expanded, showing various controller files like 'AddNewUser.php', 'ApiController.php', etc., and 'ResourceControllerAPI.php' is currently selected. The main editor area displays the code for 'ResourceControllerAPI.php':

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class ResourceControllerAPI extends Controller
8 {
9     /**
10      * Display a listing of the resource.
11      */
12     public function index()
13     {
14         //
15     }
16
17     /**
18      * Show the form for creating a new resource.
19     */
20 }
```

Below the editor, tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL' are visible, with 'TERMINAL' being active. The terminal window at the bottom shows the command '\$ php artisan make:controller ResourceControllerAPI --resource' being run, followed by the response 'INFO Controller [c:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\app\Http\Controllers\ResourceControllerAPI.php] created successfully.' To the right of the terminal, there is a sidebar with several 'bash' entries.

Sending Index Request

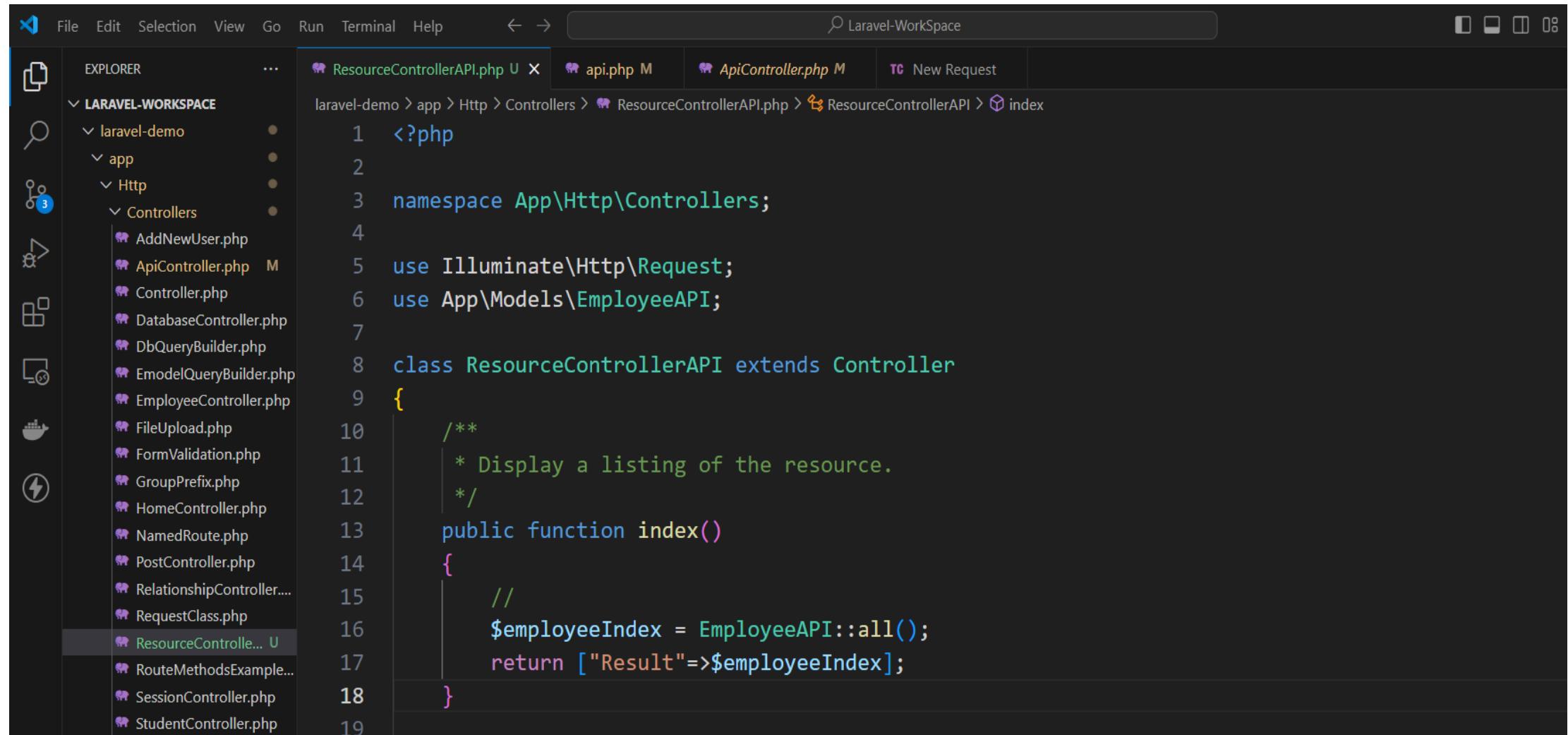
The screenshot shows the Thunder Client interface for sending an API request. The request URL is set to `localhost:8000/api/resourcecontrollerapi`. The response status is `200 OK`, size is `36 Bytes`, and time is `215 ms`. The response body is a JSON object with a single key-value pair: `"Result": "Database Index Retrieved"`.

Request URL: `localhost:8000/api/resourcecontrollerapi`

Response:

```
1 {
2   "Result": "Database Index Retrieved"
3 }
```

Sending Index Request



The screenshot shows a code editor interface with a dark theme. On the left is a sidebar with various icons: a file icon, a magnifying glass, a gear with a blue circle, a grid, a square with a circle, a ship, and a lightning bolt. The main area has a title bar with "File Edit Selection View Go Run Terminal Help" and a search bar "Laravel-WorkSpace". Below the title bar is a tab bar with "ResourceControllerAPI.php U X", "api.php M", "ApiController.php M", and "New Request". The main content area displays the following PHP code:

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\EmployeeAPI;
7
8 class ResourceControllerAPI extends Controller
9 {
10     /**
11      * Display a listing of the resource.
12      */
13     public function index()
14     {
15         //
16         $employeeIndex = EmployeeAPI::all();
17         return ["Result"=>$employeeIndex];
18     }
19 }
```

The code defines a controller named `ResourceControllerAPI` that extends the `Controller` class. It contains a single method `index()` which returns a JSON response containing all employees.

Sending Index Request

The screenshot shows the Postman application interface for testing a Laravel API. The left sidebar displays the 'LARAVEL-WORKSPACE' folder containing several PHP files under 'app/Http/Controllers'. The 'ResourceController.php' file is currently selected. The main workspace shows a request configuration for a GET method to the endpoint 'localhost:8000/api/resourcecontrollerapi'. The 'Query' tab is active, showing an empty 'Query Parameters' section. The 'Send' button is visible at the top right of the request form. To the right, the response panel shows the status '200 OK', size '1.15 KB', and time '282 ms'. The 'Response' tab is selected, displaying the JSON data returned from the API. The JSON output is as follows:

```
1  {
2    "Result": [
3      {
4        "id": 1,
5        "name": "Yitayew Solomon",
6        "email": "yitayewsolomon3@gmail.com",
7        "phone": "0937559718",
8        "created_at": "2024-11-12T00:00:00.000000Z",
9        "updated_at": "2024-11-12T00:00:00.000000Z"
10      },
11      {
12        "id": 2,
13        "name": "Haileab Solomon",
14        "email": "haileabsolomon@gmail.com",
15        "phone": "0937559718",
16        "created_at": "2024-11-12T00:00:00.000000Z",
17        "updated_at": "2024-11-12T00:00:00.000000Z"
18      },
19      {
20        "id": 3,
21        "name": "Yared Solomon",
22        "email": "yaredsolomon@gmail.com",
23        "phone": "0937559718",
24        "created_at": "2024-11-12T00:00:00.000000Z",
25        "updated_at": "2024-11-12T00:00:00.000000Z"
26      }
    ]
```

Thank you!

Appreciate your action.