



# Laravel



# Discussion Outline

- ❖ **Middleware in Laravel**
- ❖ **Types of Middleware**
- ❖ **Connect to MySQL Database**
- ❖ **Model in Laravel**

# Middleware in Laravel

- In Laravel, middleware is a layer between the **request** and the **application** that filters HTTP requests entering the application.
- Middleware can be used for tasks like **authentication**, **logging**, and **modifying** request data. It's a powerful tool for **controlling access** to specific parts of your application and for pre- or post-processing requests.

# Common Uses for Middleware


- ❖ **Authentication:** Ensuring only logged-in users can access certain routes.
- ❖ **Logging:** Logging each request or tracking user activity.
- ❖ **CORS:** Adding Cross-Origin Resource Sharing headers to the response.
- ❖ **Data Sanitization:** Modifying request data before passing it to the controller.
- ❖ **Maintenance Mode:** Redirecting all users to a maintenance page when the site is down.

# Creating Middleware

## Creating Middleware

You can create a new middleware in Laravel using the Artisan command:

bash

 Copy code

```
php artisan make:middleware CheckRole
```

This command will create a new `CheckRole` middleware file in the `app/Http/Middleware` directory.

# Types of Middleware



In Laravel, middleware can be classified into different types based on how they are applied and their purpose. Here's a breakdown of the main types:

## 1. Global Middleware


- **Purpose:** These middleware are applied to every HTTP request for the application.
- **Usage:** Typically used for tasks that need to be executed on every request, such as session management, CORS, or logging.
- **Example:** The `TrimStrings` middleware trims whitespace from request input data on all routes.

## Cont. ...

Registration: Global middleware are registered in the `$middleware` property of `app/Http/`

`Kernel.php` :

php

 Copy code

```
protected $middleware = [  
    \App\Http\Middleware\TrustHosts::class,  
    \App\Http\Middleware\TrustProxies::class,  
    \App\Http\Middleware\HandleCors::class,  
    \App\Http\Middleware\PreventRequestsDuringMaintenance::class,  
];
```

# Cont. ...

## 2. Route Middleware


- **Purpose:** These middleware are applied only to specific routes or route groups, allowing fine-grained control over which requests they affect.
- **Usage:** Useful for applying middleware to routes with particular requirements, like authentication or role-based access control.
- **Example:** `auth` middleware checks if the user is authenticated for specific routes.

**Registration:** Route middleware are registered in the `$routeMiddleware` array in `Kernel.php` and are then referenced by their alias:



# Cont. ...


php

 Copy code

```
protected $routeMiddleware = [  
    'auth' => \App\Http\Middleware\Authenticate::class,  
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,  
    'checkRole' => \App\Http\Middleware\CheckRole::class,  
];
```

## Usage in Routes:

php

 Copy code

```
Route::get('/admin', [AdminController::class, 'index'])->middleware('auth');
```

# Cont. ...

## 3. Middleware Groups


- **Purpose:** Middleware groups allow bundling multiple middleware together and applying them as a group to a set of routes. This simplifies route definitions and makes code more organized.
- **Usage:** Common for categorizing routes by context, such as `web` and `api` routes.
- **Example:** The `web` group includes middleware for session handling, CSRF protection, etc., while the `api` group often includes rate limiting.

**Registration:** Middleware groups are registered in the `$middlewareGroups` array in `Kernel.php`:

# Cont. ...

Registration: Middleware groups are registered in the `$middlewareGroups` array in `Kernel.php`:

php

 Copy code


```
protected $middlewareGroups = [
    'web' => [
        \App\Http\Middleware\EncryptCookies::class,
        \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
        \Illuminate\Session\Middleware\StartSession::class,
        \Illuminate\View\Middleware\ShareErrorsFromSession::class,
        \App\Http\Middleware\VerifyCsrfToken::class,
    ],

    'api' => [
        'throttle:api',
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],
];
```

# Cont. ...

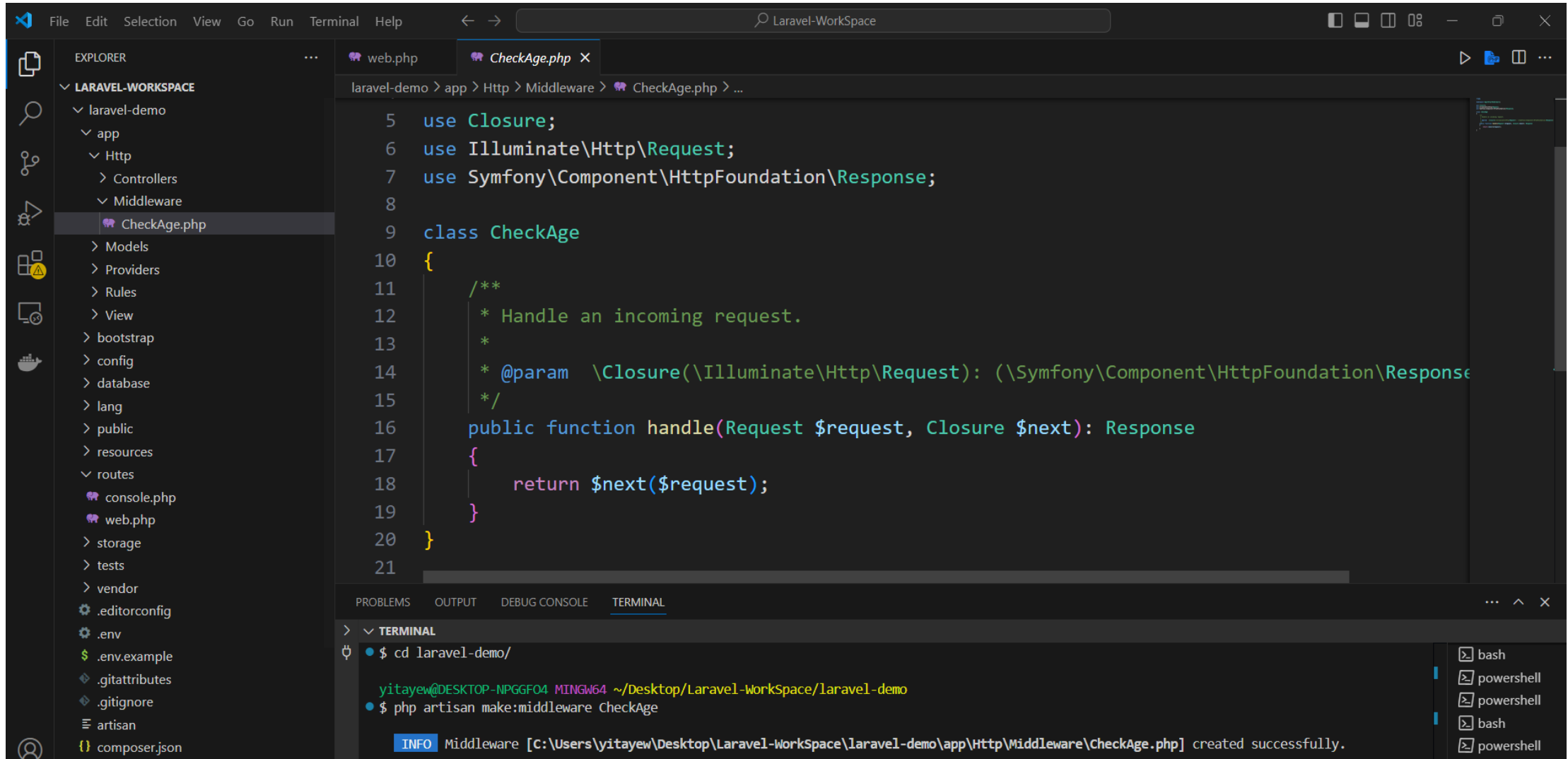
## Usage in Routes:

php

 Copy code

```
Route::middleware(['web'])->group(function () {  
    Route::get('/', function () {  
        return view('welcome');  
    });  
});
```

# Example (Global Route)



The screenshot displays the Visual Studio Code interface with a Laravel project. The Explorer sidebar on the left shows the project structure, with the `CheckAge.php` file selected under `app > Http > Middleware`. The main editor window shows the code for `CheckAge.php`, which is a Closure-based middleware class. The code includes the following:

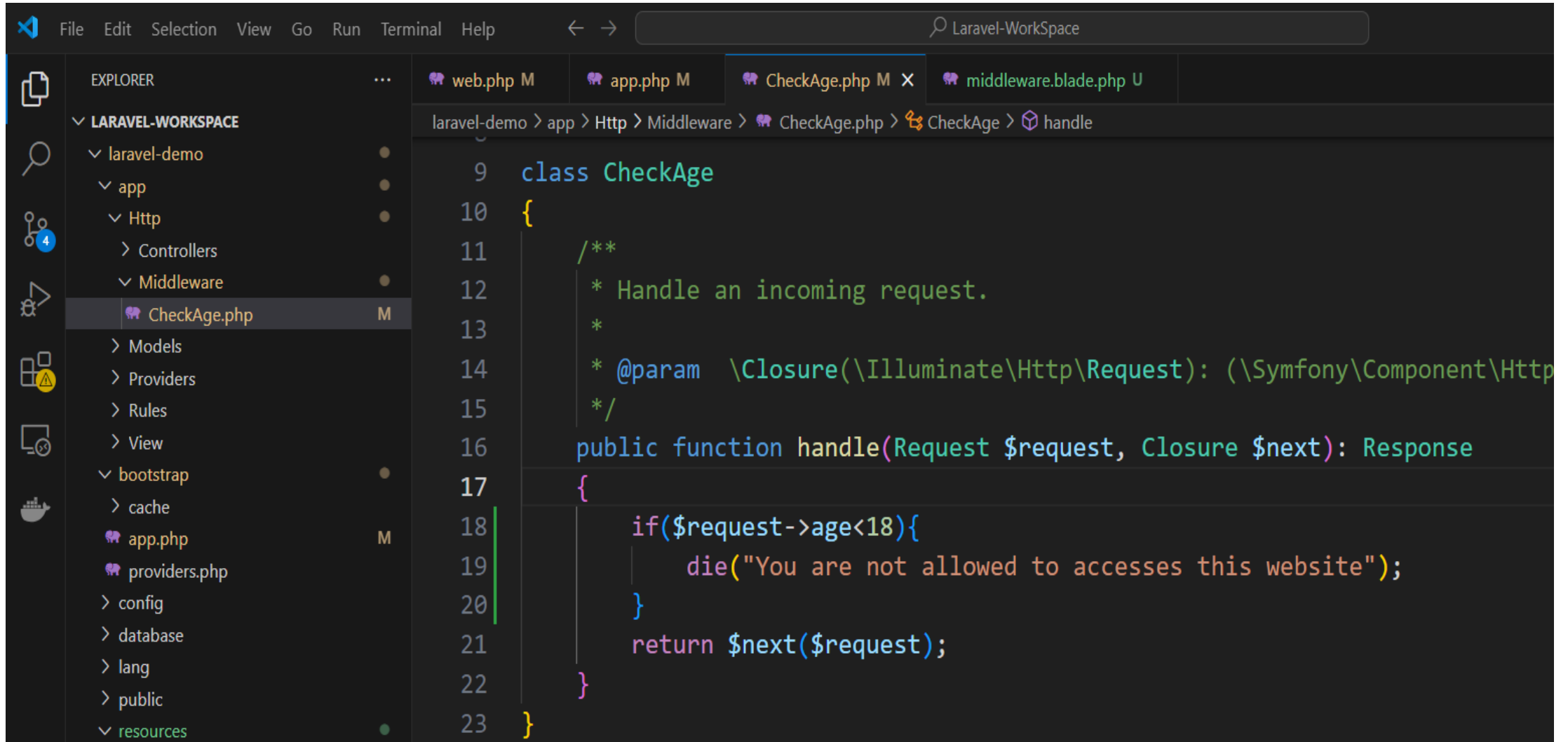
```
5 use Closure;
6 use Illuminate\Http\Request;
7 use Symfony\Component\HttpFoundation\Response;
8
9 class CheckAge
10 {
11     /**
12      * Handle an incoming request.
13      *
14      * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)
15      */
16     public function handle(Request $request, Closure $next): Response
17     {
18         return $next($request);
19     }
20 }
21
```

At the bottom, the Terminal panel shows the command execution:

```
> cd laravel-demo/
yitayew@DESKTOP-MPGGF04 MINGW64 ~/Desktop/Laravel-Workspace/laravel-demo
$ php artisan make:middleware CheckAge
```

An information message at the bottom of the terminal states: `Middleware [C:\Users\yitayew\Desktop\Laravel-Workspace\laravel-demo\app\Http\Middleware\CheckAge.php] created successfully.`

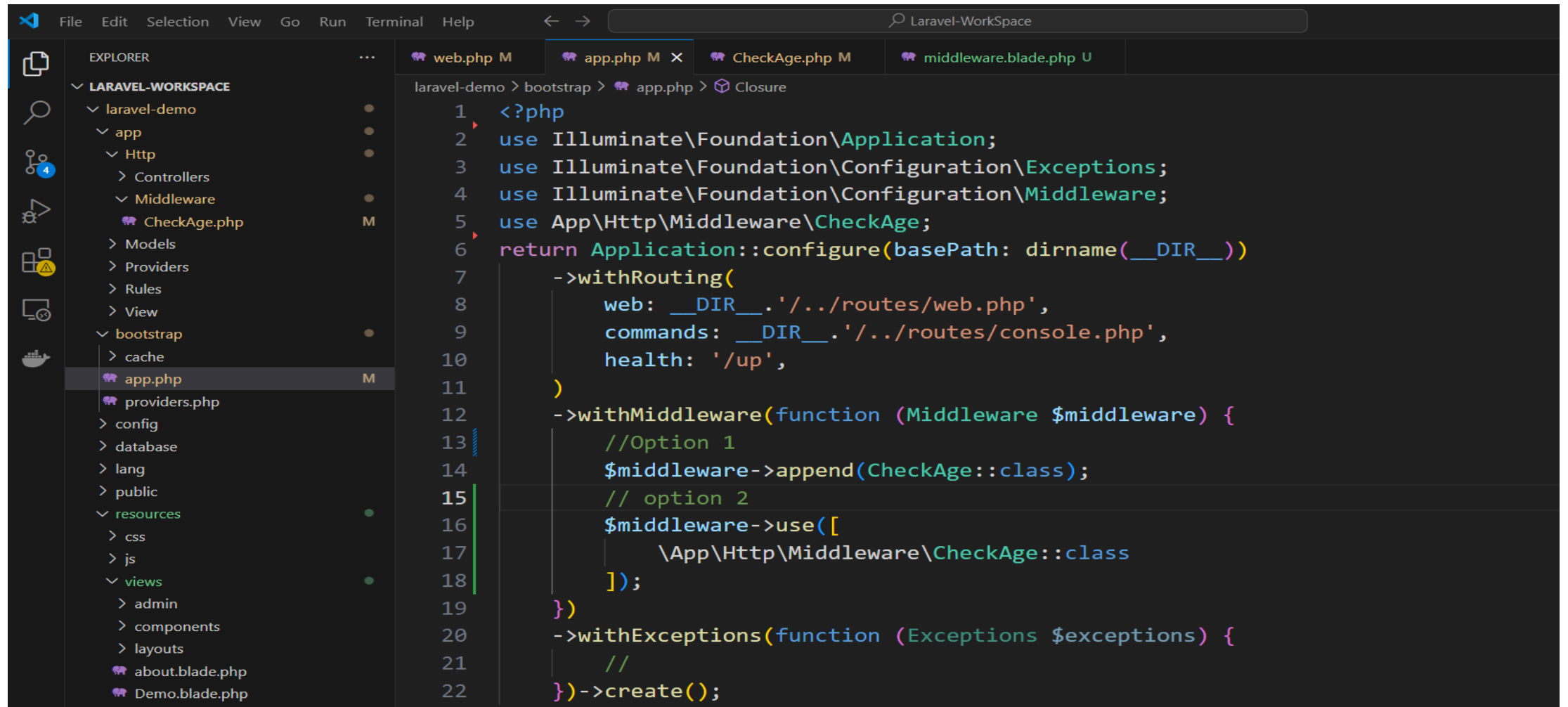
# Cont. ...



The screenshot shows a code editor with a dark theme. On the left is the Explorer sidebar showing a project structure for 'LARAVEL-WORKSPACE'. The 'Middleware' folder is expanded, and 'CheckAge.php' is selected. The main editor area shows the code for the 'CheckAge' class. The breadcrumb navigation at the top indicates the path: 'laravel-demo > app > Http > Middleware > CheckAge.php > CheckAge > handle'.

```
9 class CheckAge
10 {
11     /**
12      * Handle an incoming request.
13      *
14      * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)
15      */
16     public function handle(Request $request, Closure $next): Response
17     {
18         if($request->age<18){
19             die("You are not allowed to accesses this website");
20         }
21         return $next($request);
22     }
23 }
```

# Cont. ...

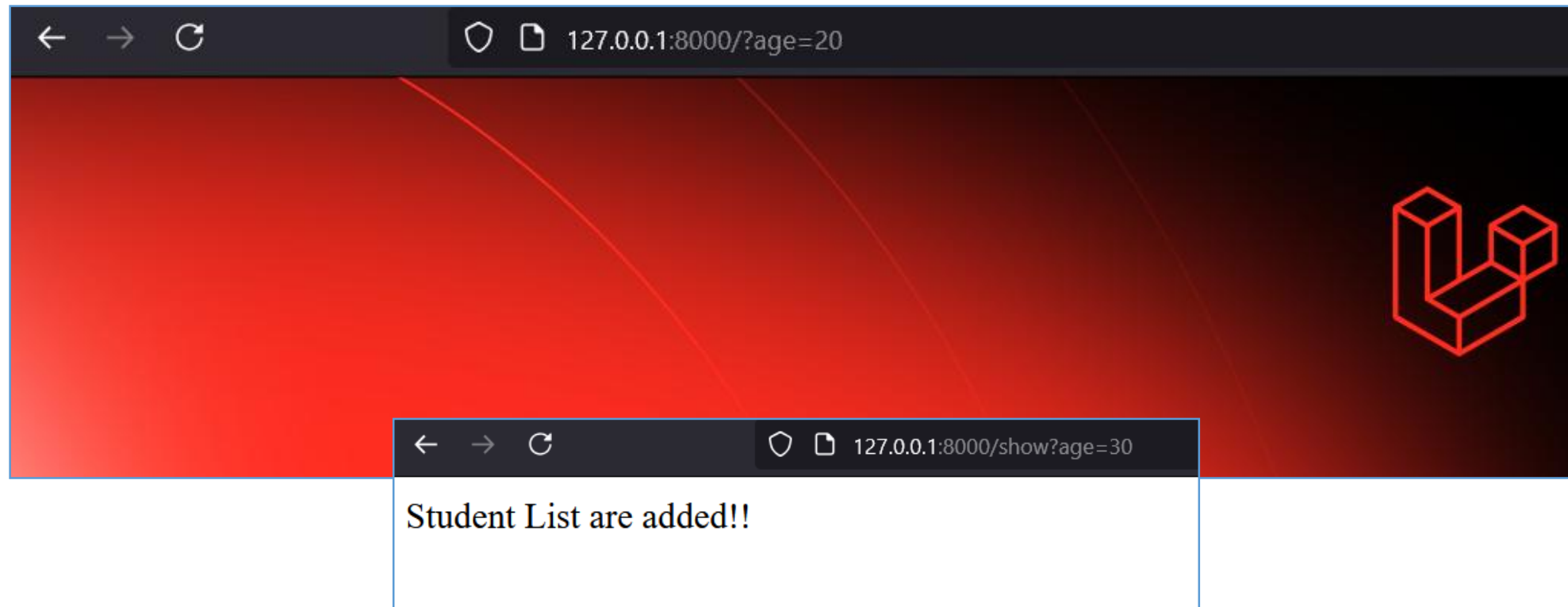
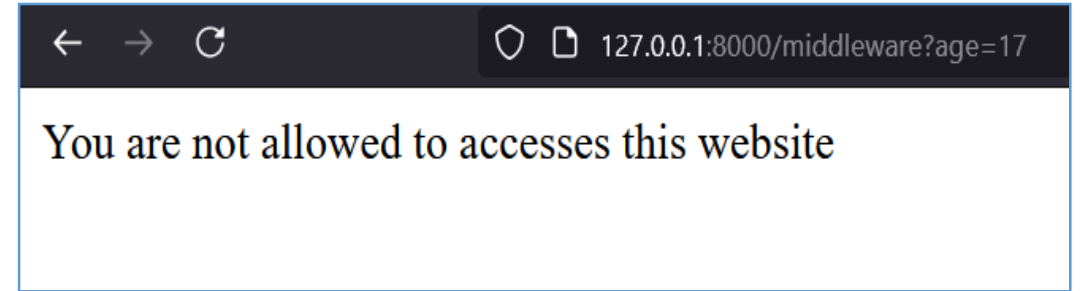
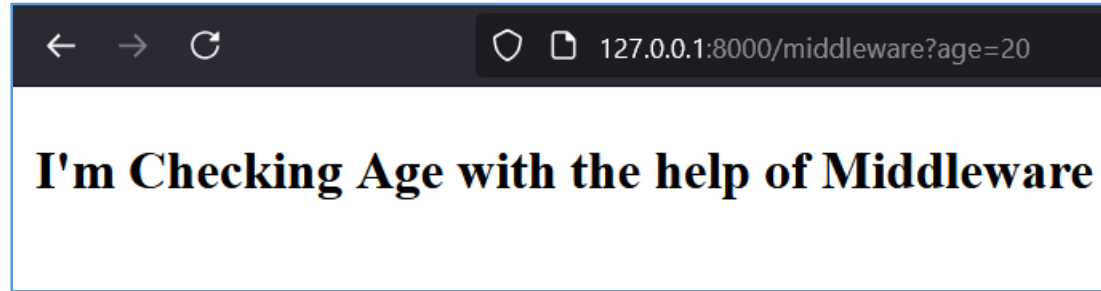


```
File Edit Selection View Go Run Terminal Help
Laravel-WorkSpace

EXPLORER
LARAVEL-WORKSPACE
  laravel-demo
    app
      Http
        Controllers
        Middleware
          CheckAge.php M
      Models
      Providers
      Rules
      View
    bootstrap
    cache
    app.php M
    providers.php
    config
    database
    lang
    public
    resources
      css
      js
      views
        admin
        components
        layouts
        about.blade.php
        Demo.blade.php

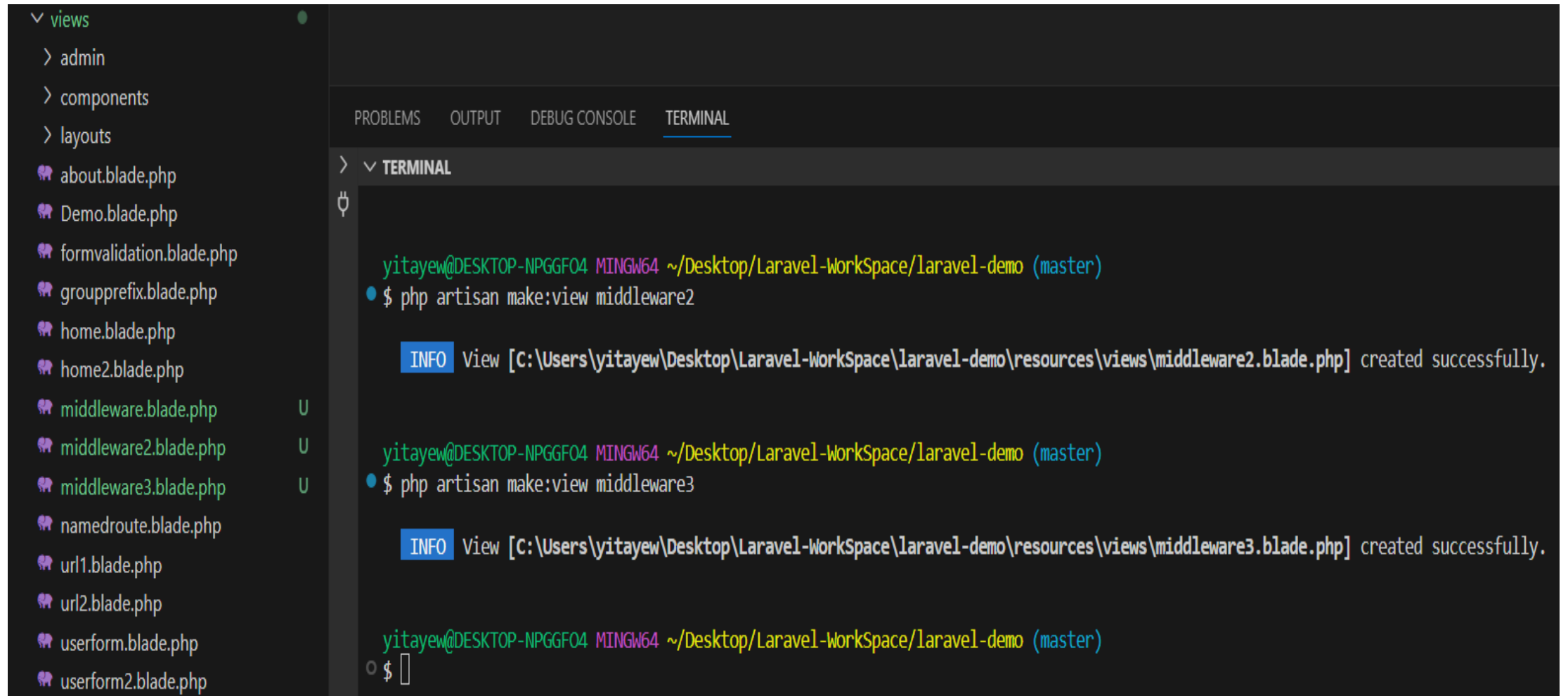
laravel-demo > bootstrap > app.php > Closure
1 <?php
2 use Illuminate\Foundation\Application;
3 use Illuminate\Foundation\Configuration\Exceptions;
4 use Illuminate\Foundation\Configuration\Middleware;
5 use App\Http\Middleware\CheckAge;
6 return Application::configure(basePath: dirname(__DIR__))
7     ->withRouting(
8         web: __DIR__.'/../routes/web.php',
9         commands: __DIR__.'/../routes/console.php',
10        health: '/up',
11    )
12    ->withMiddleware(function (Middleware $middleware) {
13        //Option 1
14        $middleware->append(CheckAge::class);
15        // option 2
16        $middleware->use([
17            \App\Http\Middleware\CheckAge::class
18        ]);
19    })
20    ->withExceptions(function (Exceptions $exceptions) {
21        //
22    })->create();
```

# Cont. ...





# Example (Middleware Group) Apply on Single Route



The image shows a screenshot of an IDE interface. On the left, a file explorer displays the 'views' directory with a list of Blade templates: 'admin', 'components', 'layouts', 'about.blade.php', 'Demo.blade.php', 'formvalidation.blade.php', 'groupprefix.blade.php', 'home.blade.php', 'home2.blade.php', 'middleware.blade.php', 'middleware2.blade.php', 'middleware3.blade.php', 'namedroute.blade.php', 'url1.blade.php', 'url2.blade.php', 'userform.blade.php', and 'userform2.blade.php'. The 'middleware' files are marked with a 'U' icon. On the right, the 'TERMINAL' tab is active, showing the following commands and output:

```
yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/laravel-demo (master)
$ php artisan make:view middleware2

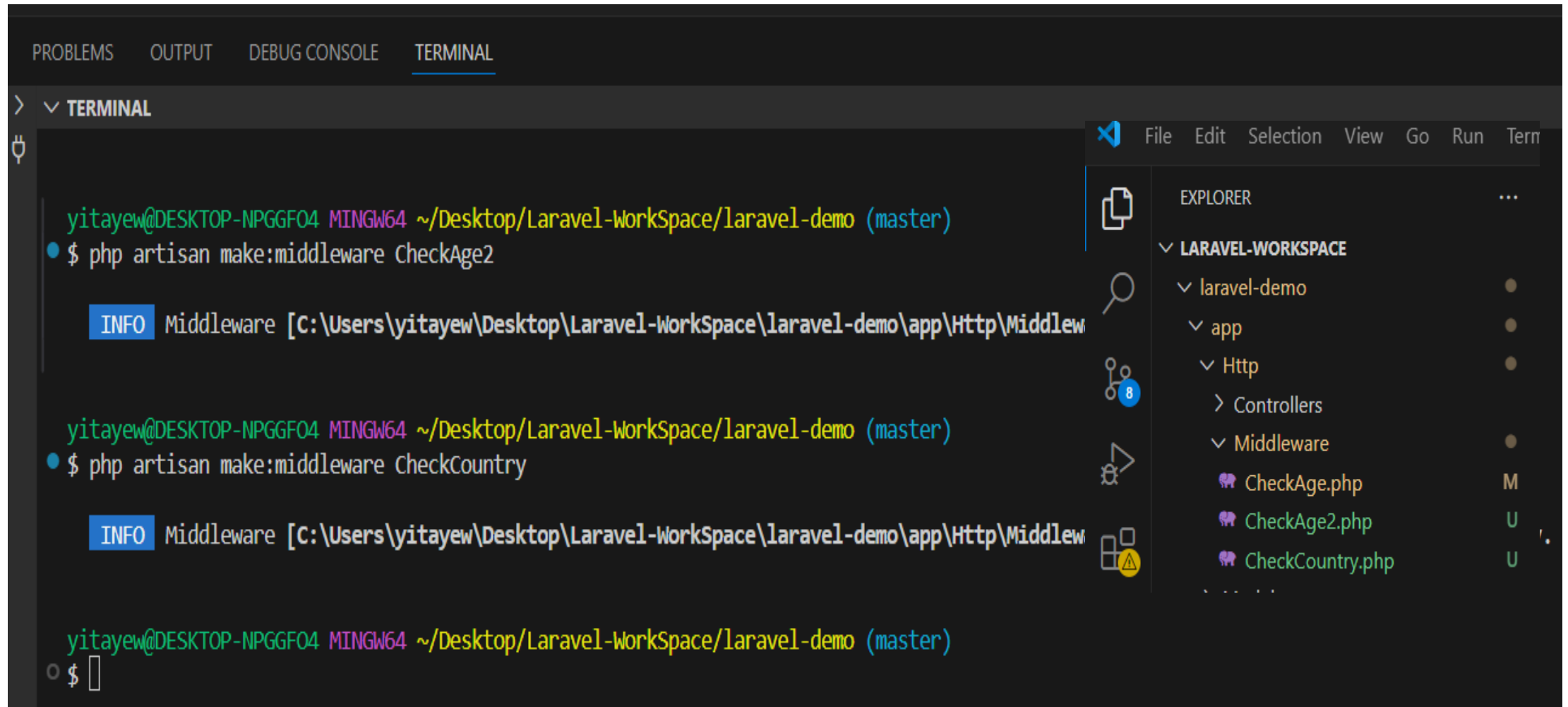
INFO View [C:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\resources\views\middleware2.blade.php] created successfully.

yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/laravel-demo (master)
$ php artisan make:view middleware3

INFO View [C:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\resources\views\middleware3.blade.php] created successfully.

yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/laravel-demo (master)
$
```

# Cont. ...



The screenshot shows the Visual Studio Code interface with the Terminal and Explorer views. The Terminal view is active, showing the execution of two Laravel Artisan commands to create middleware. The Explorer view on the right shows the project structure, including the 'app\Http\Middleware' directory where the new middleware files are being created.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
```

> ▼ TERMINAL

yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/laravel-demo (master)

- \$ php artisan make:middleware CheckAge2

INFO Middleware [C:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\app\Http\Middleware]

yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/laravel-demo (master)

- \$ php artisan make:middleware CheckCountry

INFO Middleware [C:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\app\Http\Middleware]

yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/laravel-demo (master)

\$

File Edit Selection View Go Run Term

EXPLORER

▼ LARAVEL-WORKSPACE

▼ laravel-demo

▼ app

▼ Http

> Controllers

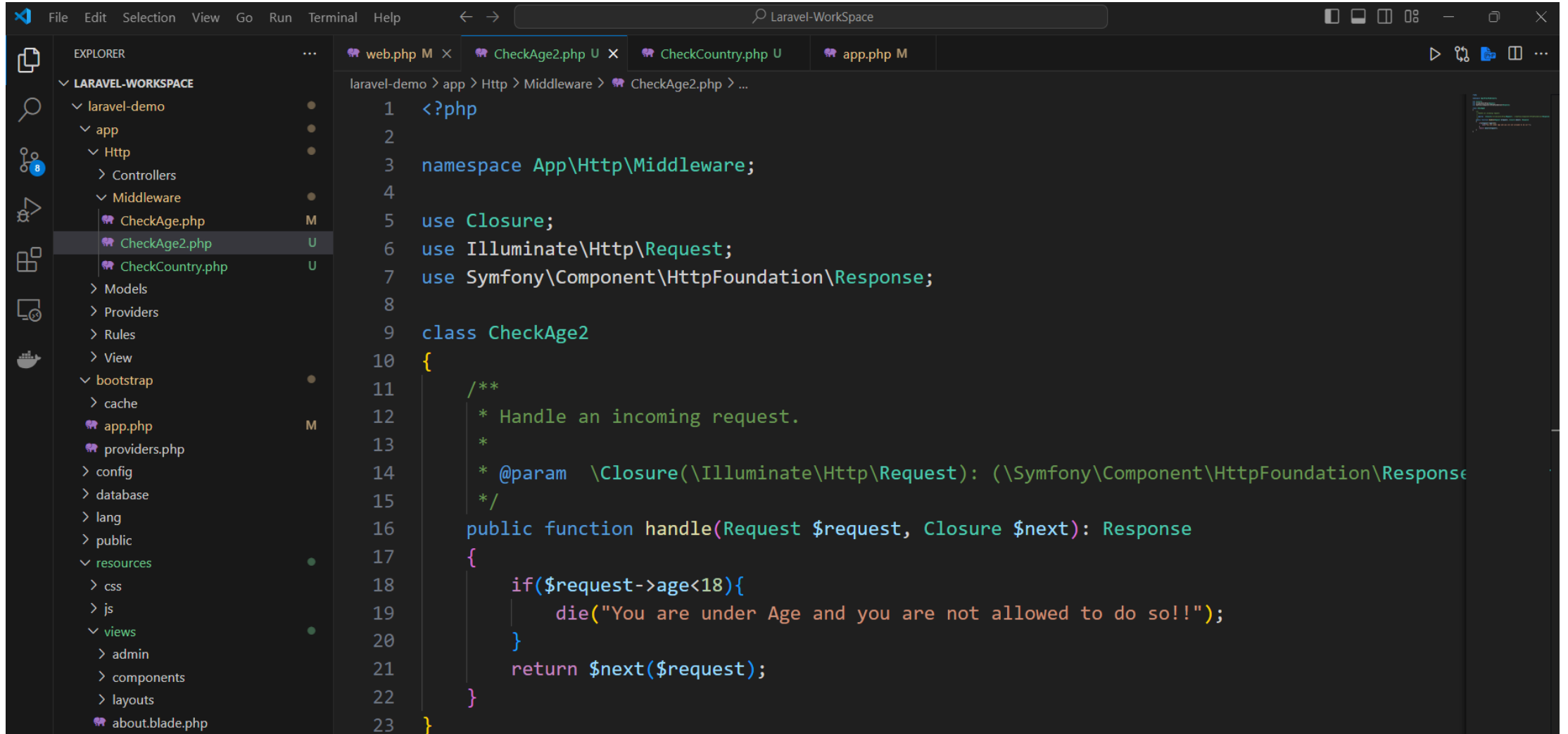
▼ Middleware

CheckAge.php M

CheckAge2.php U

CheckCountry.php U

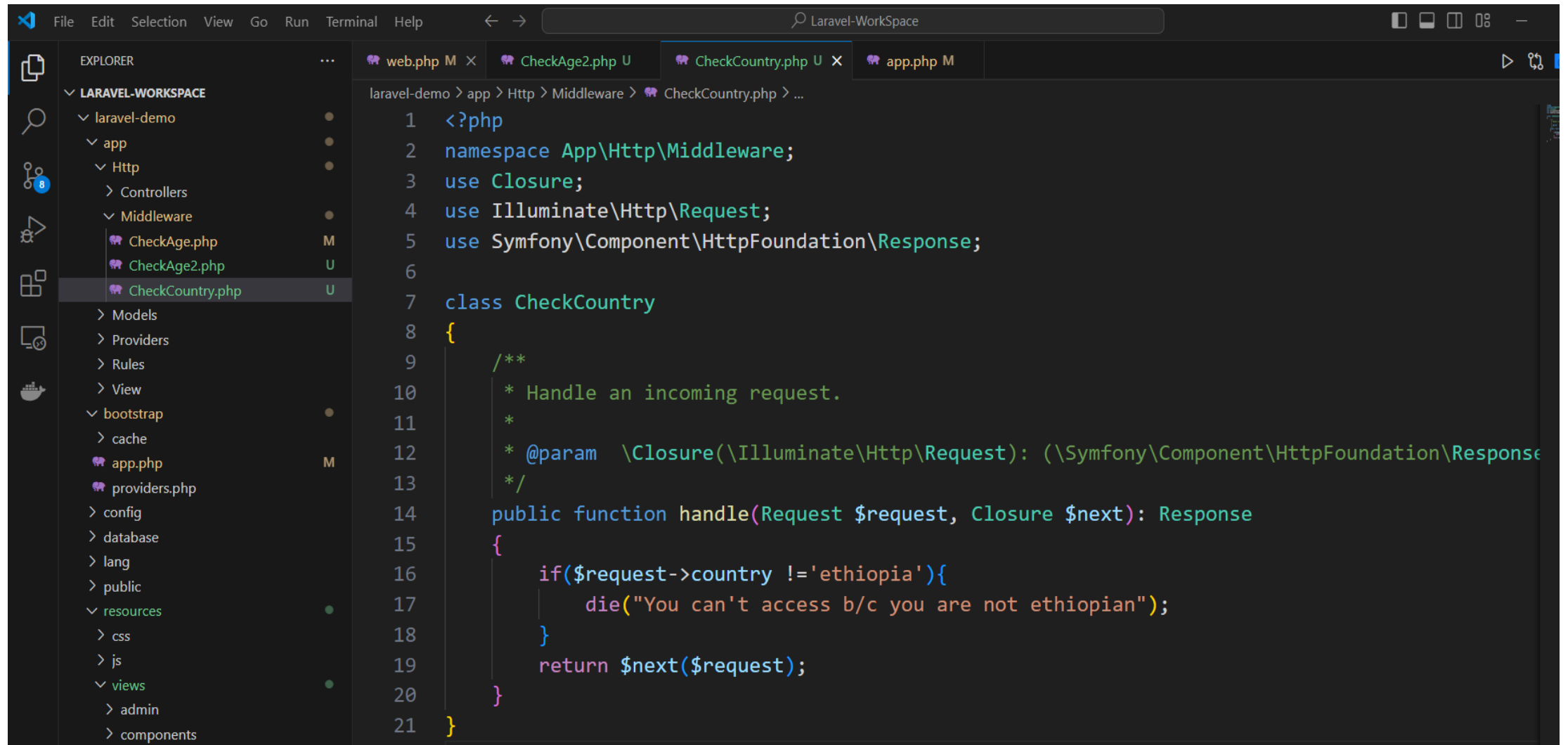
# Cont. ...



The screenshot shows an IDE window titled "Laravel-WorkSpace". The Explorer panel on the left displays the project structure for "laravel-demo". The file "CheckAge2.php" is selected in the "app/Http/Middleware" directory. The main editor shows the code for "CheckAge2.php".

```
laravel-demo > app > Http > Middleware > CheckAge2.php > ...  
1  <?php  
2  
3  namespace App\Http\Middleware;  
4  
5  use Closure;  
6  use Illuminate\Http\Request;  
7  use Symfony\Component\HttpFoundation\Response;  
8  
9  class CheckAge2  
10 {  
11     /**  
12      * Handle an incoming request.  
13      *  
14      * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response) $next  
15      */  
16     public function handle(Request $request, Closure $next): Response  
17     {  
18         if($request->age<18){  
19             die("You are under Age and you are not allowed to do so!!");  
20         }  
21         return $next($request);  
22     }  
23 }
```

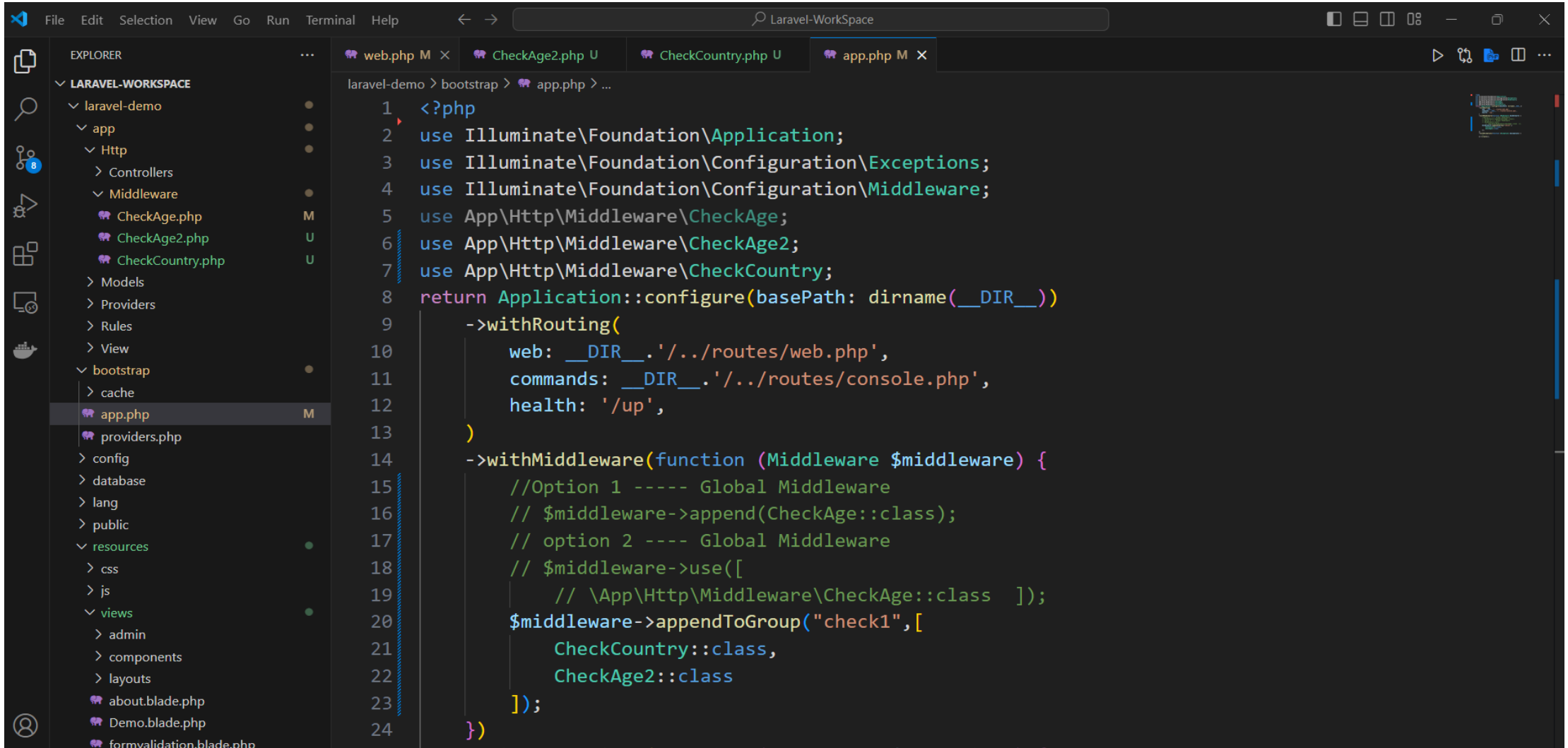
# Cont. ...



The screenshot shows an IDE window titled "Laravel-WorkSpace". The Explorer panel on the left shows the project structure for "LARAVEL-WORKSPACE". The file "CheckCountry.php" is selected in the "Middleware" directory. The main editor displays the code for "CheckCountry.php".

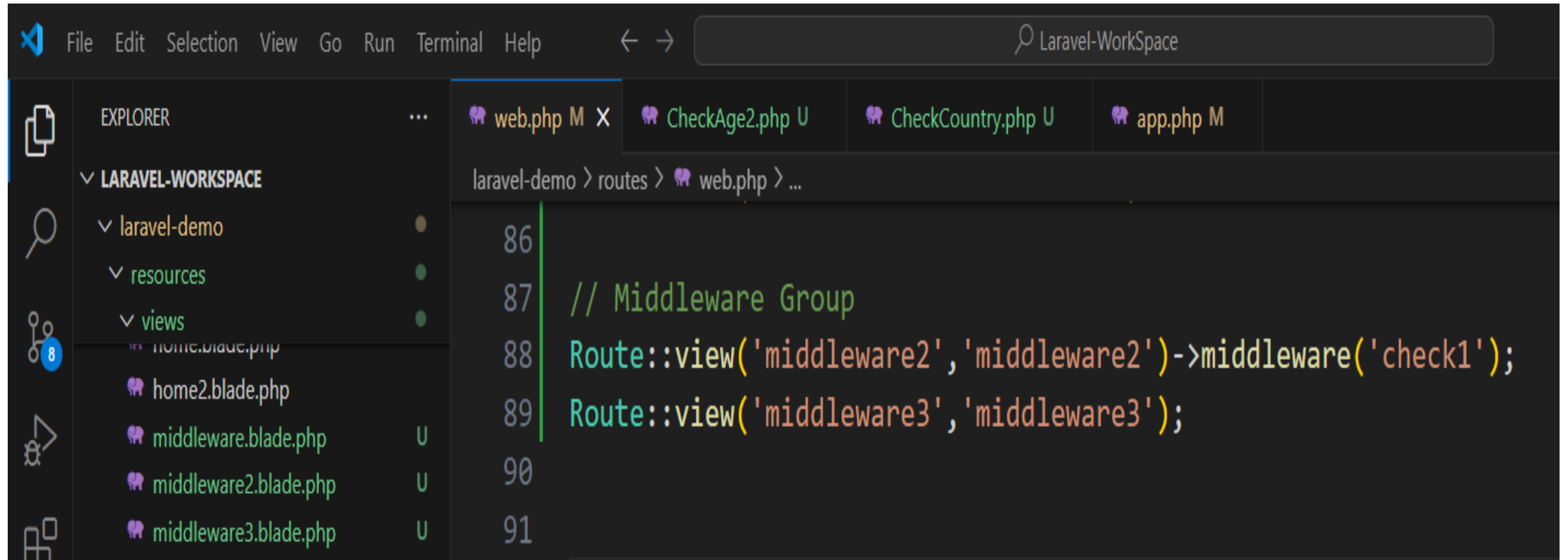
```
1 <?php
2 namespace App\Http\Middleware;
3 use Closure;
4 use Illuminate\Http\Request;
5 use Symfony\Component\HttpFoundation\Response;
6
7 class CheckCountry
8 {
9     /**
10      * Handle an incoming request.
11      *
12      * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)
13      */
14     public function handle(Request $request, Closure $next): Response
15     {
16         if($request->country != 'ethiopia'){
17             die("You can't access b/c you are not ethiopian");
18         }
19         return $next($request);
20     }
21 }
```

# Count. ...



```
1 <?php
2 use Illuminate\Foundation\Application;
3 use Illuminate\Foundation\Configuration\Exceptions;
4 use Illuminate\Foundation\Configuration\Middleware;
5 use App\Http\Middleware\CheckAge;
6 use App\Http\Middleware\CheckAge2;
7 use App\Http\Middleware\CheckCountry;
8 return Application::configure(basePath: dirname(__DIR__))
9     ->withRouting(
10         web: __DIR__.'/../routes/web.php',
11         commands: __DIR__.'/../routes/console.php',
12         health: '/up',
13     )
14     ->withMiddleware(function (Middleware $middleware) {
15         //Option 1 ----- Global Middleware
16         // $middleware->append(CheckAge::class);
17         // option 2 ----- Global Middleware
18         // $middleware->use([
19             // \App\Http\Middleware\CheckAge::class ]);
20         $middleware->appendToGroup("check1",[
21             CheckCountry::class,
22             CheckAge2::class
23         ]);
24     })
```

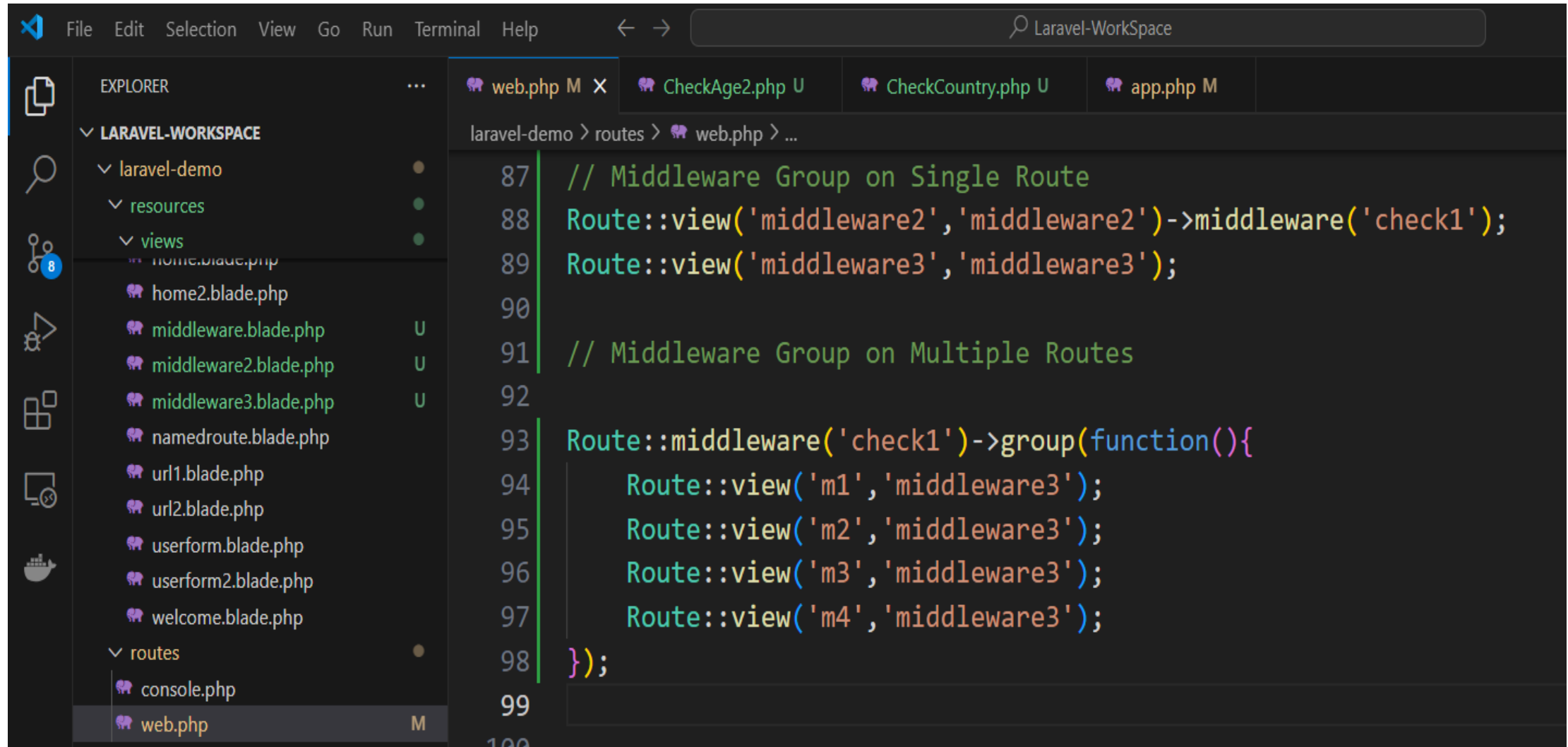
# Cont. ...



The screenshot shows the Visual Studio Code interface with a Laravel workspace. The Explorer panel on the left shows the project structure: **LARAVEL-WORKSPACE** containing **laravel-demo**, which has **resources** and **views** folders. The **resources** folder contains **home.blade.php**, **home2.blade.php**, **middleware.blade.php**, **middleware2.blade.php**, and **middleware3.blade.php**. The **views** folder contains **home.blade.php**. The **home.blade.php** file is selected, showing its content in the editor. The editor also shows the **routes** file, which contains the following code:

```
86
87 // Middleware Group
88 Route::view('middleware2','middleware2')->middleware('check1');
89 Route::view('middleware3','middleware3');
90
91
```

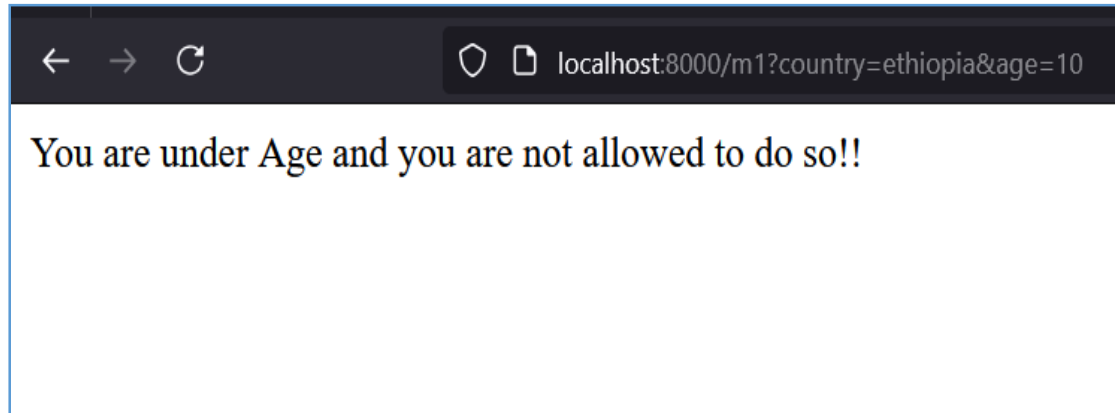
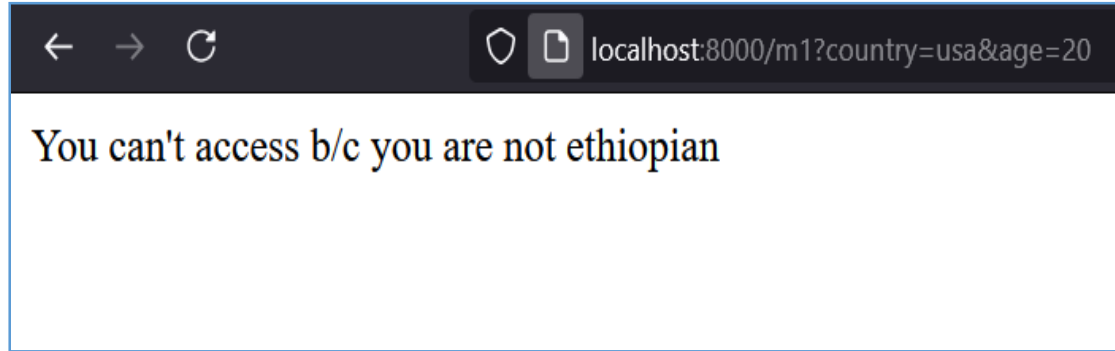
# For Multiple Route



The screenshot shows a Visual Studio Code editor interface with a Laravel workspace. The Explorer panel on the left shows the project structure, including the `resources/views` directory and the `routes` directory. The `web.php` file is selected in the routes directory. The main editor area displays the code for `web.php`, which includes two middleware groups: one for a single route and one for multiple routes.

```
87 // Middleware Group on Single Route
88 Route::view('middleware2','middleware2')->middleware('check1');
89 Route::view('middleware3','middleware3');
90
91 // Middleware Group on Multiple Routes
92
93 Route::middleware('check1')->group(function(){
94     Route::view('m1','middleware3');
95     Route::view('m2','middleware3');
96     Route::view('m3','middleware3');
97     Route::view('m4','middleware3');
98 });
99
```

# Cont. ...





# Assigning Middleware to Route

The screenshot shows an IDE with a file explorer on the left, a code editor in the center, and a terminal at the bottom.

**File Explorer:** Lists files including `about.blade.php`, `Demo.blade.php`, `formvalidation.blade.php`, `groupprefix.blade.php`, `home.blade.php`, `home2.blade.php`, `middleware.blade.php`, `middleware2.blade.php`, `middleware3.blade.php`, `middleware4.blade.php`, `middleware5.blade.php`, `namedroute.blade.php`, `url1.blade.php`, `url2.blade.php`, `userform.blade.php`, and `userform2.blade.php`.

**Code Editor:** Shows the following code:

```
100 // Assigning Middleware to Route
101 use App\Http\Middleware\CheckAge3;
102
103
```

**Terminal:** The terminal shows the execution of two commands:

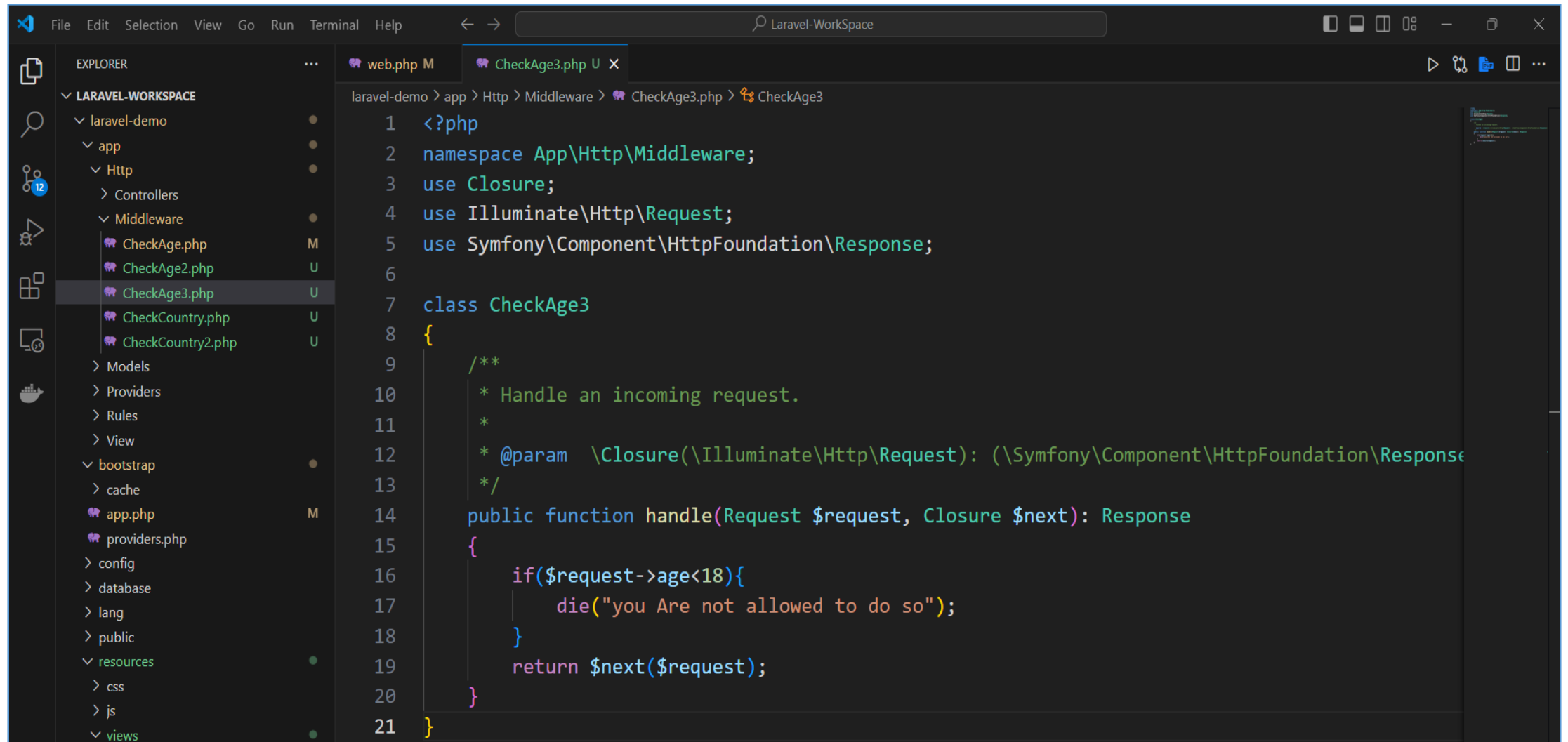
```
yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/laravel-demo (master)
$ php artisan make:view middleware4
```

**INFO** View [C:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\resources\views\middleware4.blade.php] created successfully.

```
yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/laravel-demo (master)
$ php artisan make:view middleware5
```

**INFO** View [C:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\resources\views\middleware5.blade.php] created successfully.

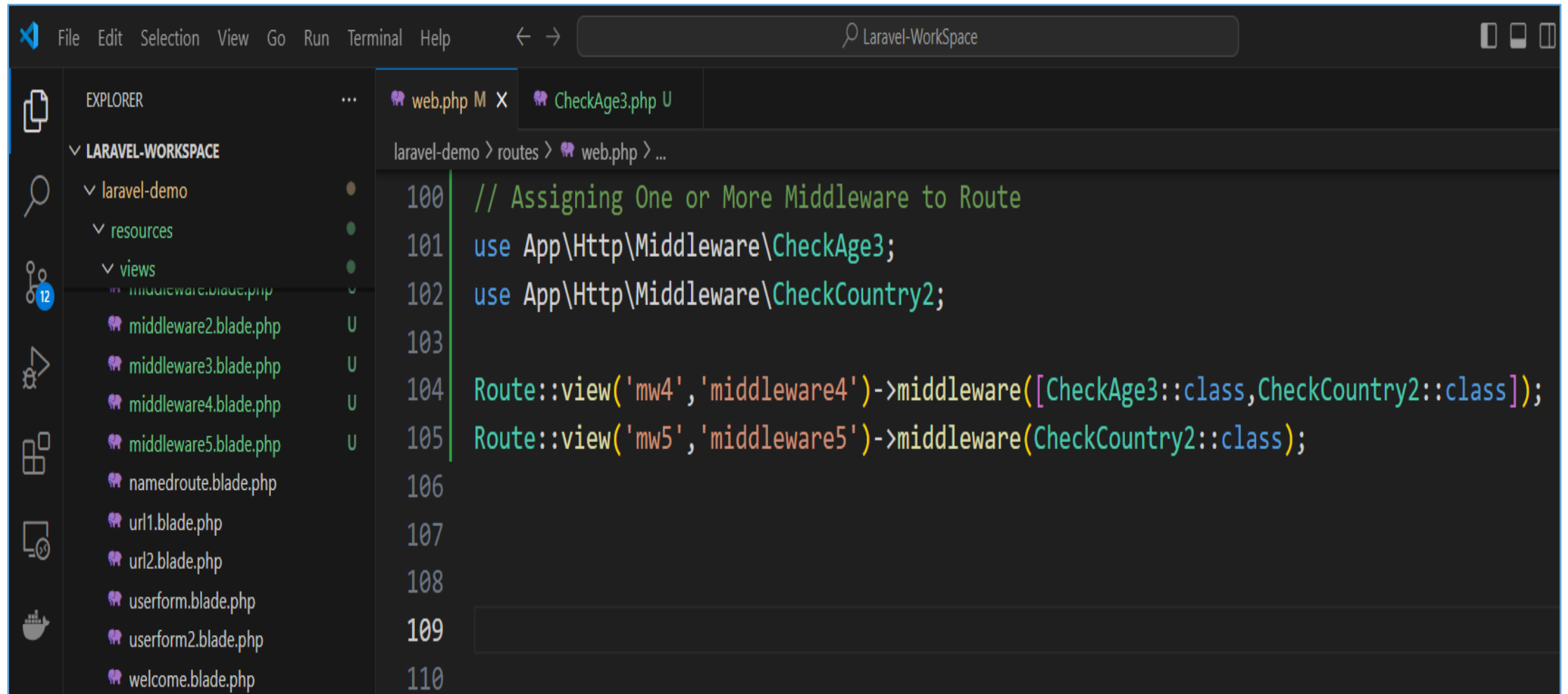
# Cont. ...



The screenshot shows a Visual Studio Code editor window with a dark theme. The Explorer sidebar on the left displays the project structure for 'LARAVEL-WORKSPACE'. The 'laravel-demo' directory is expanded, showing 'app', 'bootstrap', and 'resources' folders. The 'app' folder is further expanded to show 'Http' > 'Middleware', where 'CheckAge3.php' is selected. The main editor area shows the code for 'CheckAge3.php' with the following content:

```
1 <?php
2 namespace App\Http\Middleware;
3 use Closure;
4 use Illuminate\Http\Request;
5 use Symfony\Component\HttpFoundation\Response;
6
7 class CheckAge3
8 {
9     /**
10      * Handle an incoming request.
11      *
12      * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)
13      */
14     public function handle(Request $request, Closure $next): Response
15     {
16         if($request->age<18){
17             die("you Are not allowed to do so");
18         }
19         return $next($request);
20     }
21 }
```

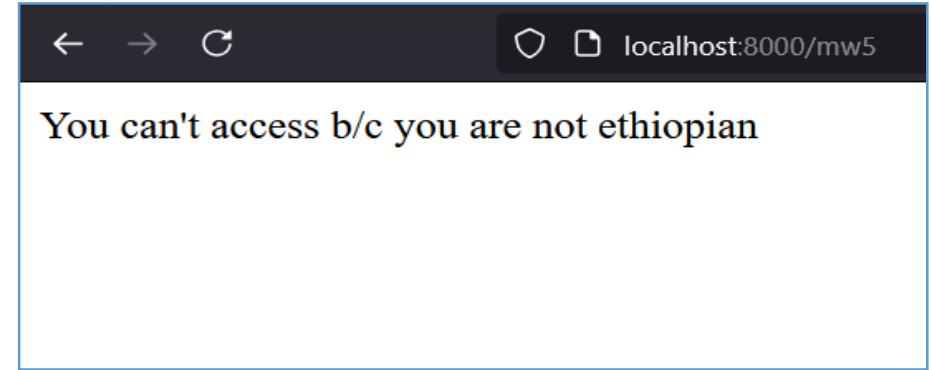
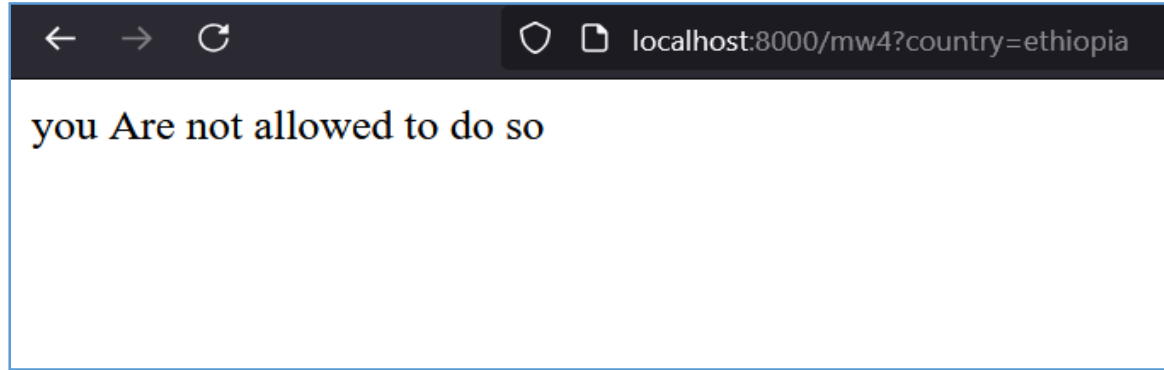
# Cont. ...



The screenshot shows the Visual Studio Code interface with a Laravel workspace. The Explorer sidebar on the left shows the file structure: **LARAVEL-WORKSPACE** > **laravel-demo** > **resources** > **views**. Under 'views', there are several Blade templates: `middleware.blade.php`, `middleware2.blade.php`, `middleware3.blade.php`, `middleware4.blade.php`, `middleware5.blade.php`, `namedroute.blade.php`, `url1.blade.php`, `url2.blade.php`, `userform.blade.php`, `userform2.blade.php`, and `welcome.blade.php`. The main editor area has two tabs: `web.php` (modified) and `CheckAge3.php` (unused). The active tab is `web.php`, showing the following code:

```
laravel-demo > routes > web.php > ...  
100 // Assigning One or More Middleware to Route  
101 use App\Http\Middleware\CheckAge3;  
102 use App\Http\Middleware\CheckCountry2;  
103  
104 Route::view('mw4','middleware4')->middleware([CheckAge3::class,CheckCountry2::class]);  
105 Route::view('mw5','middleware5')->middleware(CheckCountry2::class);  
106  
107  
108  
109  
110
```

## Cont. ...



# Connect to MySQL Database



In Laravel 11, connecting to a MySQL database is straightforward thanks to its configuration and built-in support for various database systems. Here's how to set it up:

## Step 1: Set Up Database Configuration

1. Open the `.env` file in the root of your Laravel project. This file contains environment-specific configurations, including database settings.
2. Locate the following lines:

plaintext

Copy code

```
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
```


# Cont. ...

## 3. Modify these values to match your MySQL database settings:

- `DB_DATABASE` : Name of your MySQL database.
- `DB_USERNAME` : Username for connecting to the database.
- `DB_PASSWORD` : Password for the database user.
- Optionally, you can also change `DB_HOST` if your MySQL is hosted on another server, or `DB_PORT` if it's using a non-default port.

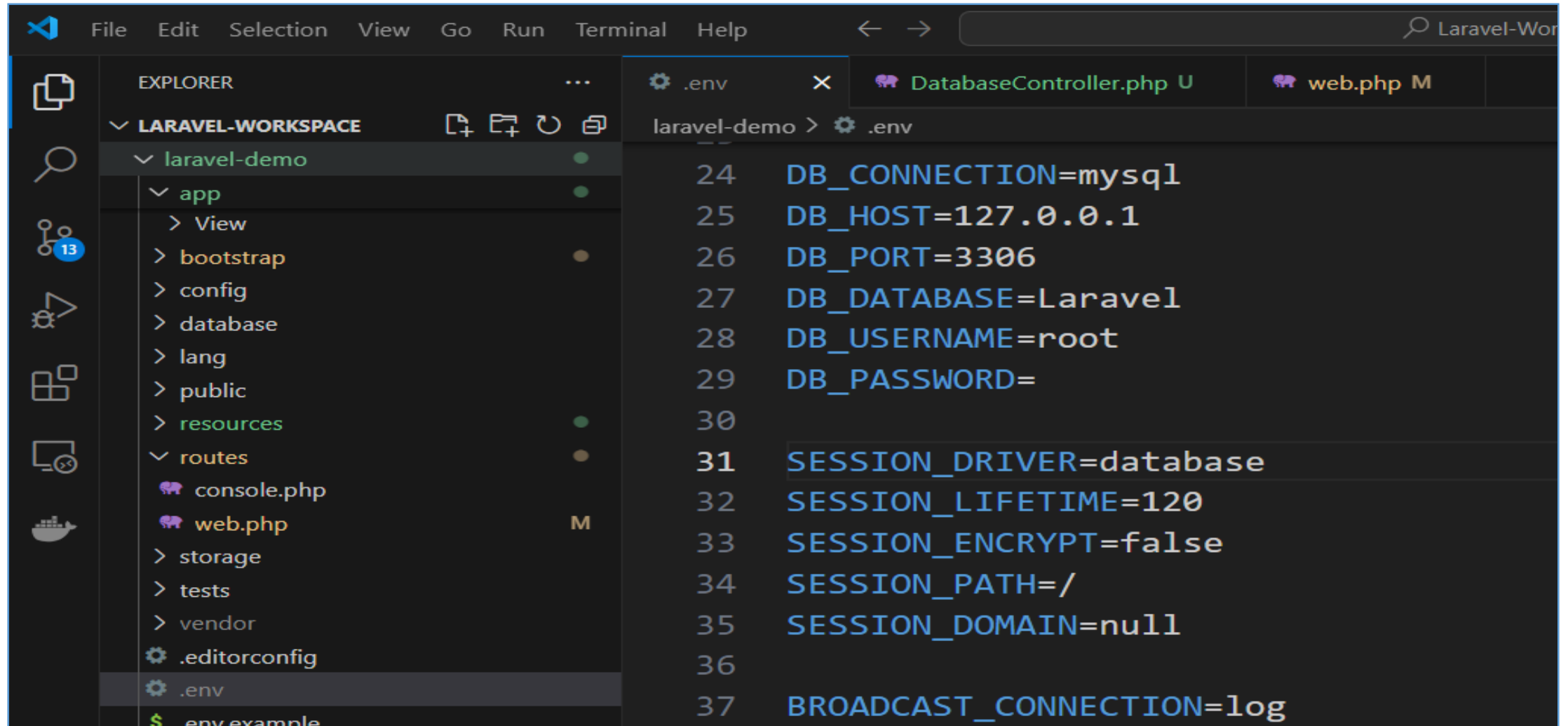
For example:

plaintext

 Copy code

```
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=example_database
DB_USERNAME=root
DB_PASSWORD=password123
```

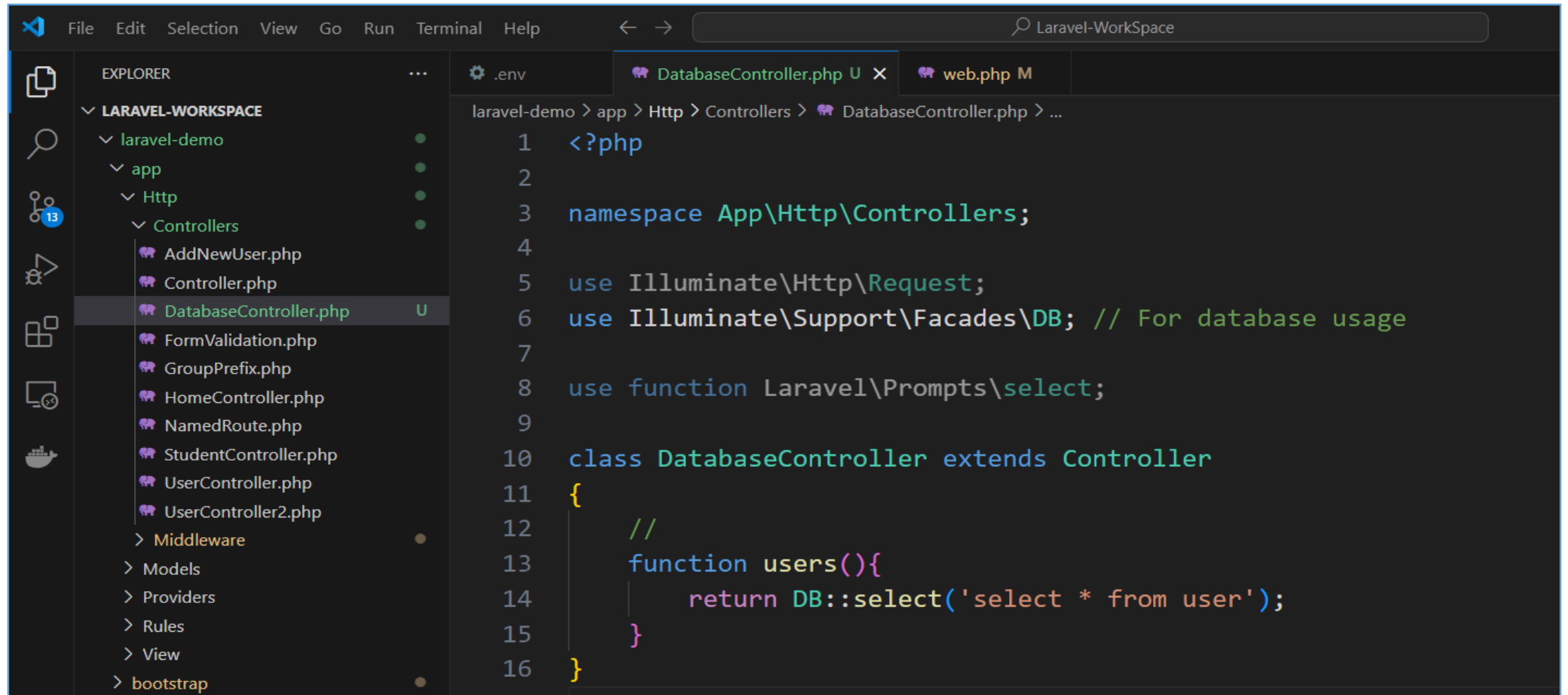
# Cont. ...



The screenshot shows the Visual Studio Code interface with a Laravel workspace. The Explorer panel on the left displays the project structure, including the `laravel-demo` folder with subfolders like `app`, `bootstrap`, `config`, `database`, `lang`, `public`, `resources`, `routes`, `storage`, `tests`, and `vendor`. The `routes` folder contains `console.php` and `web.php`. The `.env` file is selected in the Explorer panel. The main editor area shows the contents of the `.env` file, which contains database and session configuration variables.

```
24 DB_CONNECTION=mysql
25 DB_HOST=127.0.0.1
26 DB_PORT=3306
27 DB_DATABASE=Laravel
28 DB_USERNAME=root
29 DB_PASSWORD=
30
31 SESSION_DRIVER=database
32 SESSION_LIFETIME=120
33 SESSION_ENCRYPT=false
34 SESSION_PATH=/
35 SESSION_DOMAIN=null
36
37 BROADCAST_CONNECTION=log
```

# Cont. ...

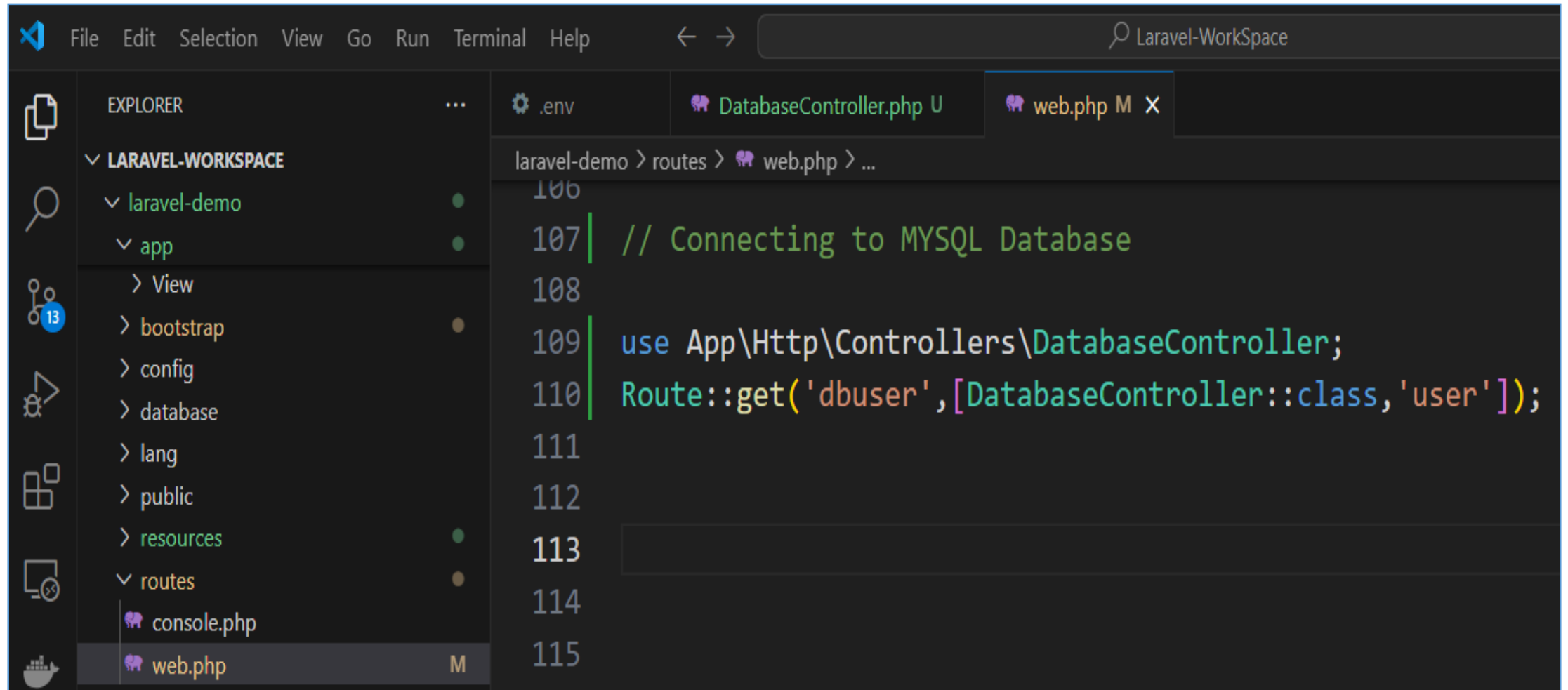


The screenshot shows the Visual Studio Code interface with a Laravel project. The Explorer panel on the left displays the project structure, including the 'DatabaseController.php' file which is currently selected. The main editor area shows the code for 'DatabaseController.php'.

```
laravel-demo > app > Http > Controllers > DatabaseController.php > ...  
1  <?php  
2  
3  namespace App\Http\Controllers;  
4  
5  use Illuminate\Http\Request;  
6  use Illuminate\Support\Facades\DB; // For database usage  
7  
8  use function Laravel\Prompts\select;  
9  
10 class DatabaseController extends Controller  
11 {  
12     //  
13     function users(){  
14         return DB::select('select * from user');  
15     }  
16 }
```



# Cont. ...



```
File Edit Selection View Go Run Terminal Help
LARAVEL-WORKSPACE
  laravel-demo
    app
      View
      bootstrap
      config
      database
      lang
      public
      resources
      routes
        console.php
        web.php

DatabaseController.php U
web.php M X

laravel-demo > routes > web.php > ...
106
107 // Connecting to MYSQL Database
108
109 use App\Http\Controllers\DatabaseController;
110 Route::get('dbuser',[DatabaseController::class,'user']);
111
112
113
114
115
```

# Cont. ...

The screenshot shows the phpMyAdmin web interface in a browser. The address bar indicates the URL: `localhost/phpmyadmin/index.php?route=/sql&pos=0&db=laravel_demo&table=users`. The interface includes a sidebar with a database tree on the left and a main panel on the right. The main panel shows the 'Table: users' view with a toolbar containing 'Browse', 'Structure', 'SQL', 'Search', 'Insert', 'Export', and 'Import'. A green message box states: 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0090 seconds.)'. Below this, the SQL query `SELECT * FROM `users`` is displayed. A 'Query results operations' dropdown menu is open, showing options like 'Edit inline', 'Edit', 'Explain SQL', 'Create PHP code', 'Refresh', and 'Create view'. The table structure is visible with columns: `id`, `name`, `email`, `email_verified_at`, `password`, `remember_token`, `created_at`, and `updated_at`. The table is currently empty. A secondary browser window is visible in the foreground, showing the URL `localhost:8000/dbuser` and a 'Pretty-print' checkbox.

phpMyAdmin

Recent Favorites

Server: 127.0.0.1 » Database: laravel\_demo » Table: users

Browse Structure SQL Search Insert Export Import

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0090 seconds.)

`SELECT * FROM `users``

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

id	name	email	email_verified_at	password	remember_token	created_at	updated_at
----	------	-------	-------------------	----------	----------------	------------	------------

Query results operations

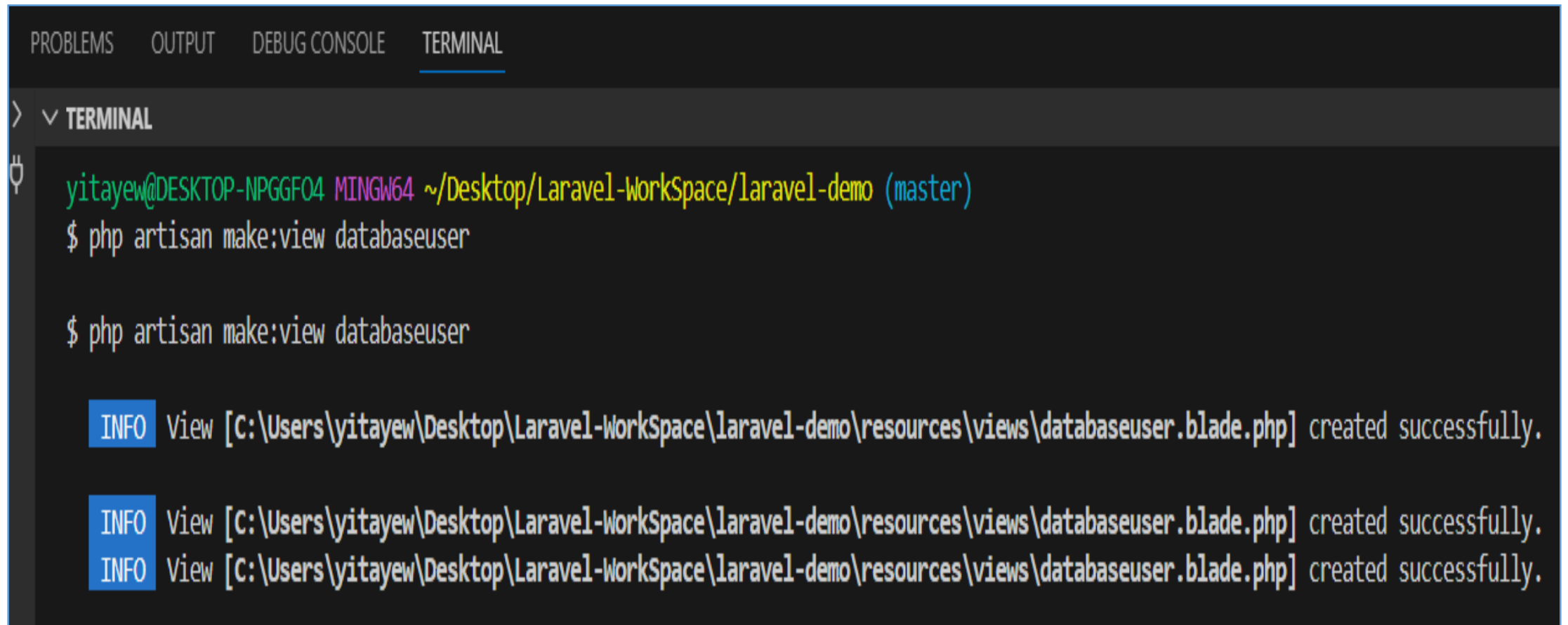
Create view

localhost:8000/dbuser

Pretty-print ☐

[ ]

# Display Database Records on UI



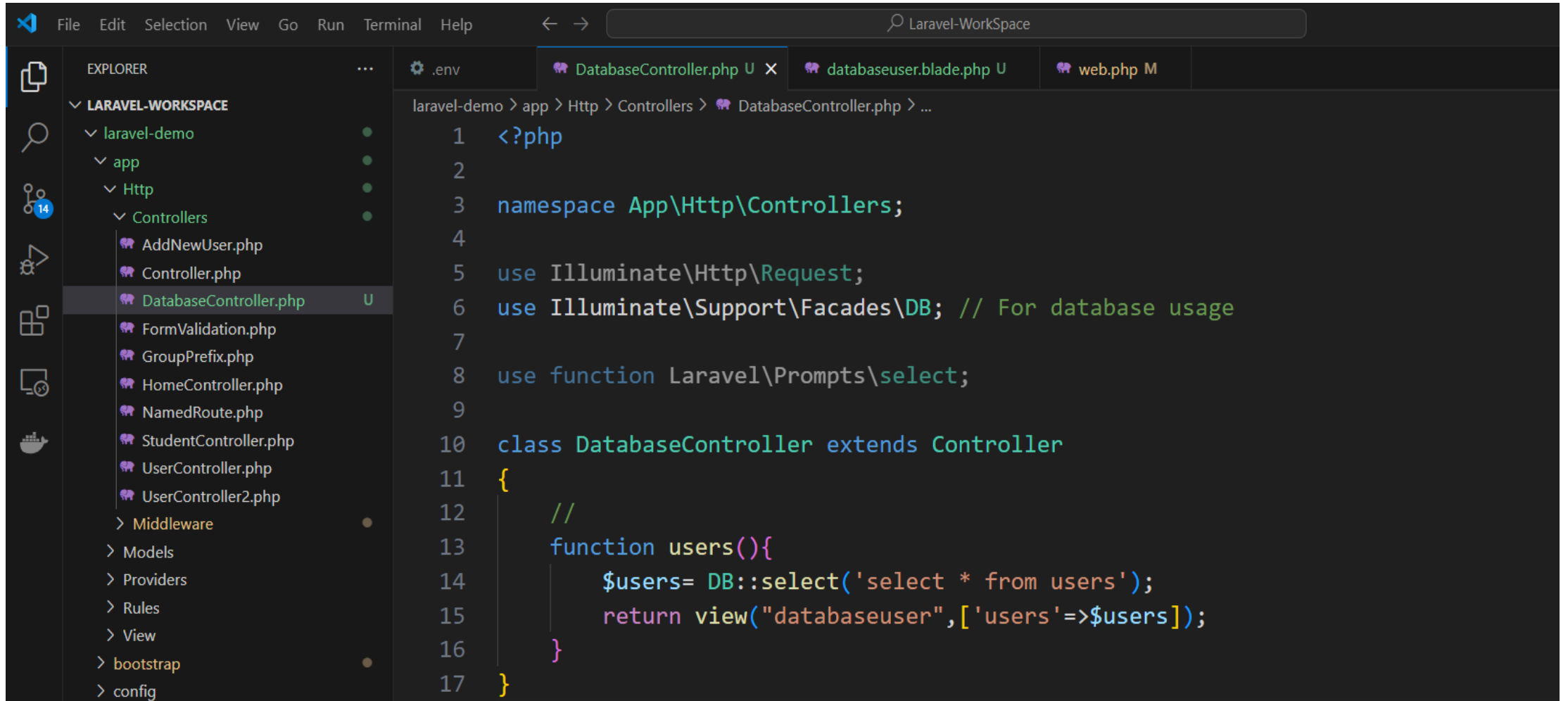
The image shows a screenshot of a Visual Studio Code terminal window. The terminal has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL, with the TERMINAL tab selected. The terminal output shows the following commands and messages:

```
> ▾ TERMINAL
yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/laravel-demo (master)
$ php artisan make:view databaseuser

$ php artisan make:view databaseuser

INFO View [C:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\resources\views\databaseuser.blade.php] created successfully.
INFO View [C:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\resources\views\databaseuser.blade.php] created successfully.
INFO View [C:\Users\yitayew\Desktop\Laravel-WorkSpace\laravel-demo\resources\views\databaseuser.blade.php] created successfully.
```

# Controller (Database Controller)

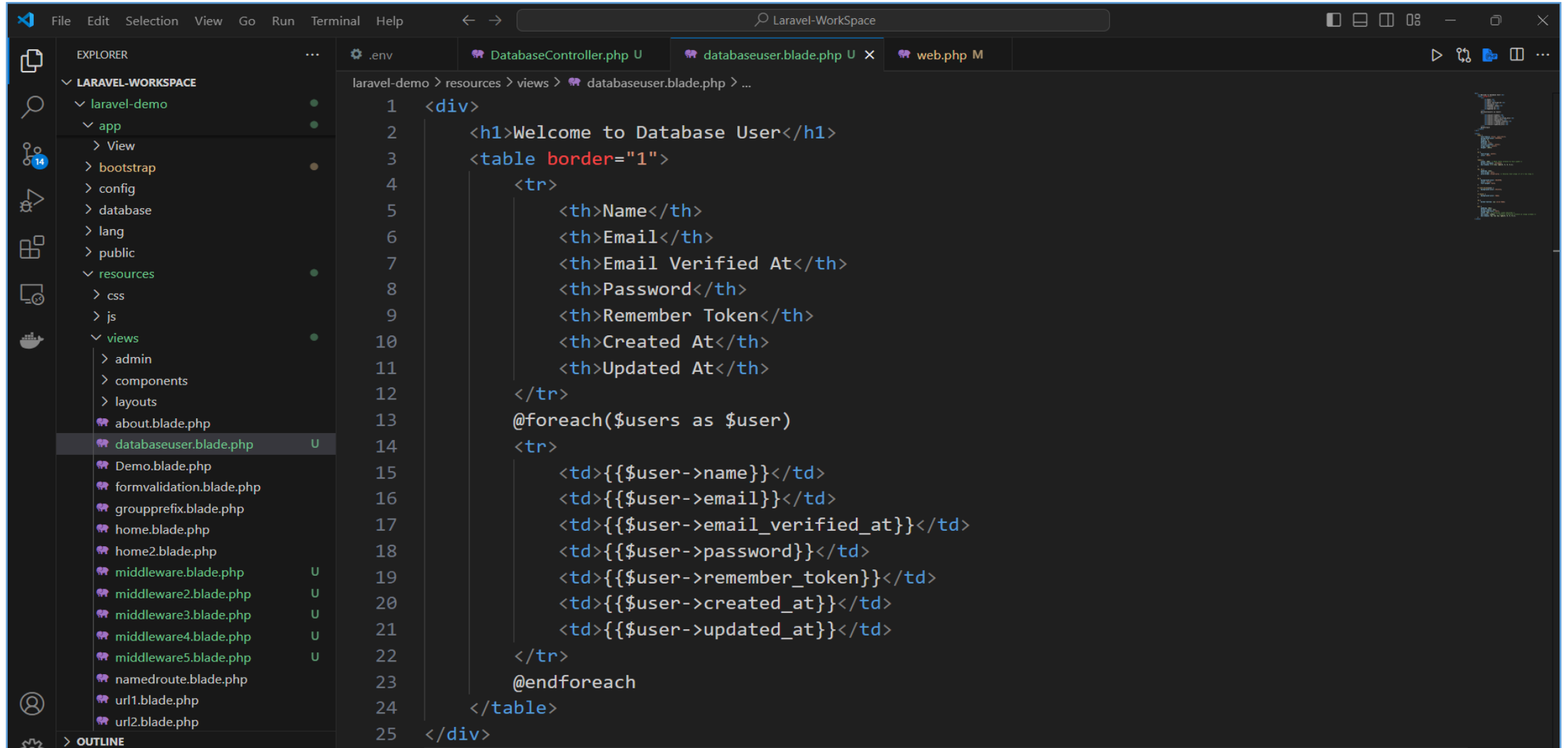


The screenshot shows a code editor with a dark theme. On the left is the Explorer sidebar showing the project structure: **LARAVEL-WORKSPACE** > **laravel-demo** > **app** > **Http** > **Controllers**. The file **DatabaseController.php** is selected and highlighted. The main editor area shows the code for **DatabaseController.php** with the following content:

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\DB; // For database usage
7
8 use function Laravel\Prompts\select;
9
10 class DatabaseController extends Controller
11 {
12     //
13     function users(){
14         $users= DB::select('select * from users');
15         return view("databaseuser",['users'=>$users]);
16     }
17 }
```

The breadcrumb at the top of the editor reads: **laravel-demo > app > Http > Controllers > DatabaseController.php > ...**. The top of the editor shows the menu (File, Edit, Selection, View, Go, Run, Terminal, Help) and the search bar (Laravel-WorkSpace).

# View (Databaseuser View)

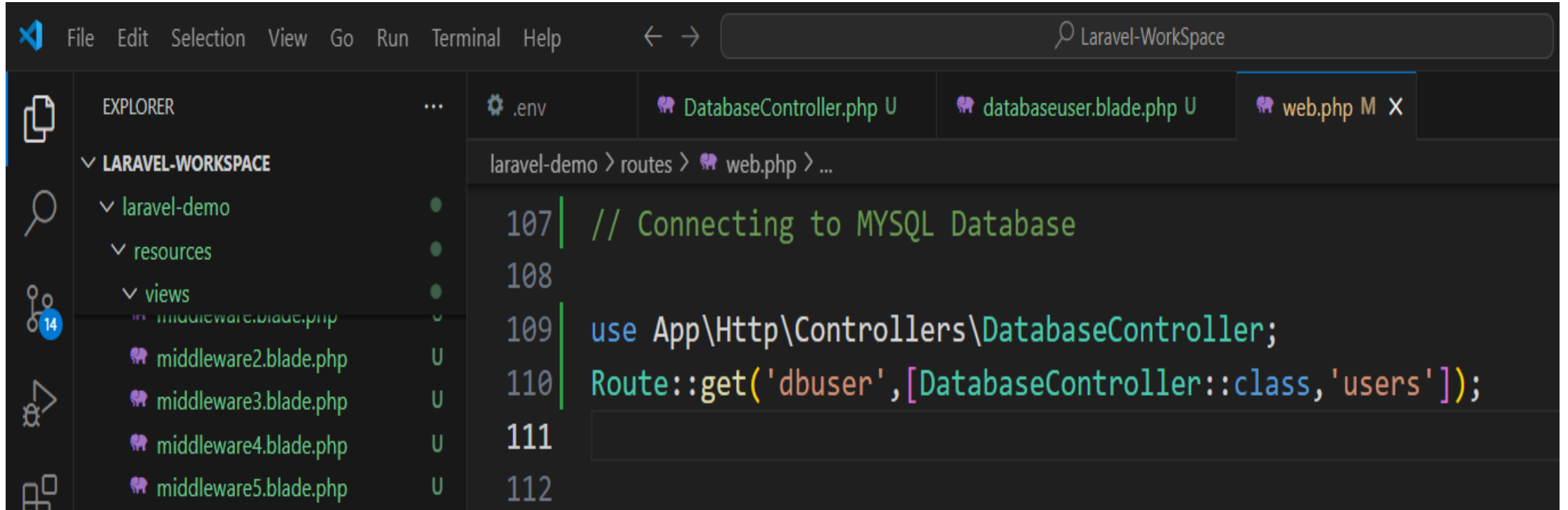


```
File Edit Selection View Go Run Terminal Help
Laravel-WorkSpace

EXPLORER
LARAVEL-WORKSPACE
  laravel-demo
    app
      View
      bootstrap
      config
      database
      lang
      public
      resources
        css
        js
        views
          admin
          components
          layouts
          about.blade.php
          databaseuser.blade.php
          Demo.blade.php
          formvalidation.blade.php
          groupprefix.blade.php
          home.blade.php
          home2.blade.php
          middleware.blade.php
          middleware2.blade.php
          middleware3.blade.php
          middleware4.blade.php
          middleware5.blade.php
          namedroute.blade.php
          url1.blade.php
          url2.blade.php

laravel-demo > resources > views > databaseuser.blade.php > ...
1 <div>
2   <h1>Welcome to Database User</h1>
3   <table border="1">
4     <tr>
5       <th>Name</th>
6       <th>Email</th>
7       <th>Email Verified At</th>
8       <th>Password</th>
9       <th>Remember Token</th>
10      <th>Created At</th>
11      <th>Updated At</th>
12    </tr>
13    @foreach($users as $user)
14      <tr>
15        <td>{{$user->name}}</td>
16        <td>{{$user->email}}</td>
17        <td>{{$user->email_verified_at}}</td>
18        <td>{{$user->password}}</td>
19        <td>{{$user->remember_token}}</td>
20        <td>{{$user->created_at}}</td>
21        <td>{{$user->updated_at}}</td>
22      </tr>
23    @endforeach
24  </table>
25 </div>
```

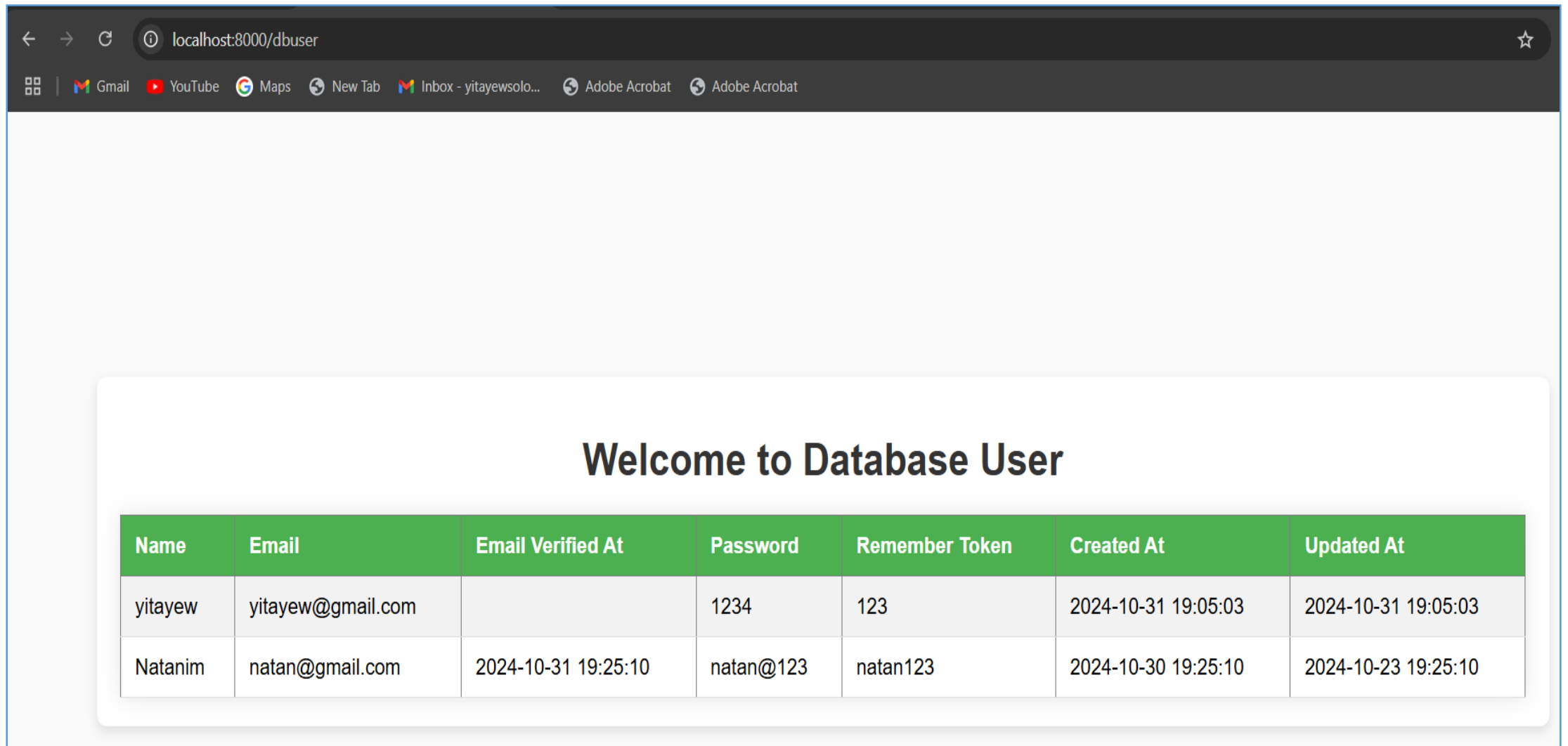
# Route (dbuser)



The screenshot shows the Visual Studio Code interface with a Laravel project. The Explorer sidebar on the left displays the project structure: **LARAVEL-WORKSPACE** containing **laravel-demo**, which has **resources** and **views** folders. Under **resources**, there are five Blade templates: **middleware.blade.php**, **middleware2.blade.php**, **middleware3.blade.php**, **middleware4.blade.php**, and **middleware5.blade.php**. The main editor area shows the **web.php** file in the **routes** directory. The code defines a GET route for **dbuser** that uses the **DatabaseController** class's **users** method.

```
107 | // Connecting to MYSQL Database
108 |
109 | use App\Http\Controllers\DatabaseController;
110 | Route::get('dbuser',[DatabaseController::class,'users']);
111 |
112 |
```

# Output (http://127.0.0.1:8000/dbuser)



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/dbuser'. The browser's tab bar includes icons for Gmail, YouTube, Maps, New Tab, and two instances of Adobe Acrobat. The main content area of the browser displays a white card with the heading 'Welcome to Database User' and a table containing user information.

Name	Email	Email Verified At	Password	Remember Token	Created At	Updated At
yitayew	yitayew@gmail.com		1234	123	2024-10-31 19:05:03	2024-10-31 19:05:03
Natanim	natan@gmail.com	2024-10-31 19:25:10	natan@123	natan123	2024-10-30 19:25:10	2024-10-23 19:25:10

# Model in Laravel

- In Laravel (and other MVC frameworks), a model represents the **data** and the **business logic** of an application. It is one of the core components of the Model-View-Controller (MVC) architecture, which Laravel follows.



# Key Points about Models in Laravel

- ❖ **Data Representation:** Models are typically used to represent a **single table** in the database, and each **instance** of a model represents a **single row** in that table.
- For example, a **User** model would correspond to a **users** table in the database, and each User instance would represent a specific user record.

## Cont. ...

- ❖ **Eloquent ORM**: Laravel uses Eloquent ORM (Object-Relational Mapping) for database interaction. Eloquent makes it easy to work with database records using an expressive, simple syntax, without writing raw SQL queries.
- Each model is linked to a database table, and **Eloquent** provides methods to query, **insert**, **update**, and **delete** records with ease.

## Cont. ...

- ❖ **Database Interaction:** Models act as an intermediary between the **database** and the **application** logic. They contain methods and properties for querying and manipulating the data.
- For instance, with Eloquent, you can fetch a record like ***User::find(1)*** to retrieve a user with ID 1, or
- ***User::where('email', 'example@example.com')->get()*** to fetch users based on their email.

## Cont. ...

❖ **Relationships:** Eloquent allows defining relationships between models, making it easy to work with related data. Examples of relationships include:

- ✓ One-to-One (e.g., a User has one Profile)
- ✓ One-to-Many (e.g., a Post has many Comments)
- ✓ Many-to-Many (e.g., a User belongs to many Roles)


## Cont. ...

- ❖ **Business Logic:** Models can also contain **business logic** or **data manipulation** methods specific to that model.
- This helps keep the application logic organized and separated from the controller or view.

# Summary On Model in Laravel

Concept	Description	Example Code
Database Representation	Models map to database tables.	A <code>User</code> model represents the <code>users</code> table.
Eloquent ORM	Provides an Object-Relational Mapping for working with database records as objects.	<pre>\$users = User::all();</pre>
CRUD Operations	Simplifies Create, Read, Update, and Delete operations.	<pre>\$user = User::find(1); \$user-&gt;delete();</pre>
Relationships	Defines relationships like <code>one-to-one</code> , <code>one-to-many</code> , <code>many-to-many</code> , and <code>polymorphic</code> .	<pre>\$user-&gt;posts();</pre> defines a <code>hasMany</code> relationship.
Mass Assignment	Protects against mass assignment vulnerabilities using <code>\$fillable</code> or <code>\$guarded</code> properties.	<pre>protected \$fillable = ['name', 'email'];</pre>

# Cont. ...

Custom Methods	Encapsulates reusable business logic within the model.	<pre>public function isAdmin() { return \$this-&gt;role === 'admin'; }</pre>
Timestamps	Automatically manages <code>created_at</code> and <code>updated_at</code> timestamps.	Enabled by default; can be disabled with <pre>public \$timestamps = false;</pre>
Query Scopes	Encapsulates reusable query logic in the model.	<pre>public function scopeActive(\$query) { return \$query-&gt;where('active', 1); }</pre>
Accessors/ Mutators	Transform attributes when retrieving or setting them.	<pre>public function getNameAttribute(\$value) { return ucfirst(\$value); }</pre>
Soft Deletes	Marks records as deleted without actually removing them.	<pre>use SoftDeletes;</pre>
Events	Allows handling model lifecycle events like <code>created</code> , <code>updated</code> , or <code>deleted</code> .	<pre>User::created(function (\$user) { // logic });</pre>
Factories	Enables model instance generation for testing and seeding. 	<pre>User::factory()-&gt;count(10)-&gt;create();</pre>


# Example

## Example of a Simple Laravel Model

Let's say we have a `User` model in Laravel that maps to a `users` table in the database.

Creating a Model: You can create a model using the Artisan CLI:

bash

 Copy code

```
php artisan make:model User
```

```
yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/laravel-demo (master)  
○ $ php artisan make:model Student
```



# Database Table (Students)

← Server: 127.0.0.1 » Database: laravel » Table: college\_students ⚙️ ↗️

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

✓ Showing rows 0 - 3 (4 total, Query took 0.0005 seconds.)

```
SELECT * FROM `college_students`
```

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ] [ [Explain SQL](#) ] [ [Create PHP code](#) ] [ [Refresh](#) ]

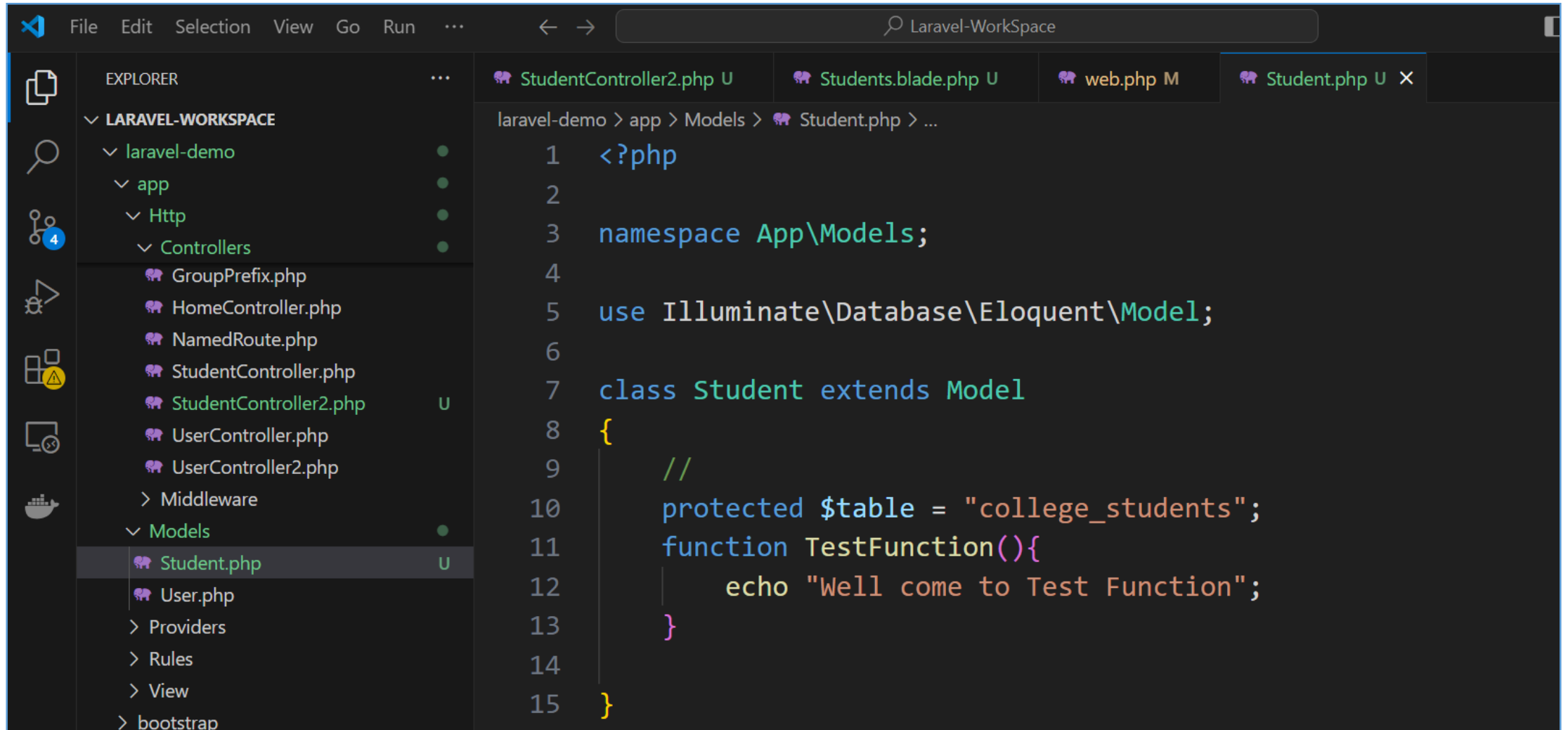
☐ Show all | Number of rows: 25 ▾ | Filter rows:  | Sort by key: None ▾

Extra options

				id	name	email	batch
<input type="checkbox"/>		Edit		Copy		Delete	1 yitayew yitayew@gmail.com 2024
<input type="checkbox"/>		Edit		Copy		Delete	2 yared yared@gmail.com 2015
<input type="checkbox"/>		Edit		Copy		Delete	3 Natanim natan@gmail.com 2014
<input type="checkbox"/>		Edit		Copy		Delete	4 yared yared@gmail.com 2014

Insert data manually using insert button

# Model(Student)



The screenshot displays a code editor interface for a Laravel project named 'Laravel-Workspace'. The Explorer panel on the left shows the project structure, with the 'Models' directory expanded and 'Student.php' selected. The main editor area shows the contents of 'Student.php', which is a PHP class extending the 'Model' class. The code includes a namespace declaration, a use statement for 'Illuminate\Database\Eloquent\Model', and a class definition with a protected static property '\$table' and a 'TestFunction()' method.

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Model;
6
7 class Student extends Model
8 {
9     //
10    protected static $table = "college_students";
11    function TestFunction(){
12        echo "Well come to Test Function";
13    }
14
15 }
```

# Configuring Model and Tables in Laravel



If your table name and model class name are different in Laravel, you need to explicitly specify the table name in your model using the `$table` property. By default, Laravel assumes the table name is the plural form of the model class name (e.g., `User` model maps to `users` table).

## Steps to Configure the Model Class:

### 1. Specify the Table Name in the Model:

- Use the `$table` property to define the custom table name.


2. **Example Configuration:** Suppose you have a table named `user_data` but your model class is `User`.

# Example

2. **Example Configuration:** Suppose you have a table named `user_data` but your model class is

`User`.

php

 Copy code

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    // Specify the table name
    protected $table = 'user_data';
}
```




# Cont. ...

## 3. Additional Customizations (Optional):

- If the table does not have `created_at` and `updated_at` columns, disable timestamps:


php

 Copy code

```
public $timestamps = false;
```

- If the primary key is not `id`, specify it:

php


 Copy code

```
protected $primaryKey = 'user_id';
```

## Cont. ...

- If the primary key is not auto-incrementing, set `$incrementing` to `false`:


php

 Copy code

```
public $incrementing = false;
```

4. **Using the Configured Model:** You can now use the model as usual, and it will interact with the `user_data` table:

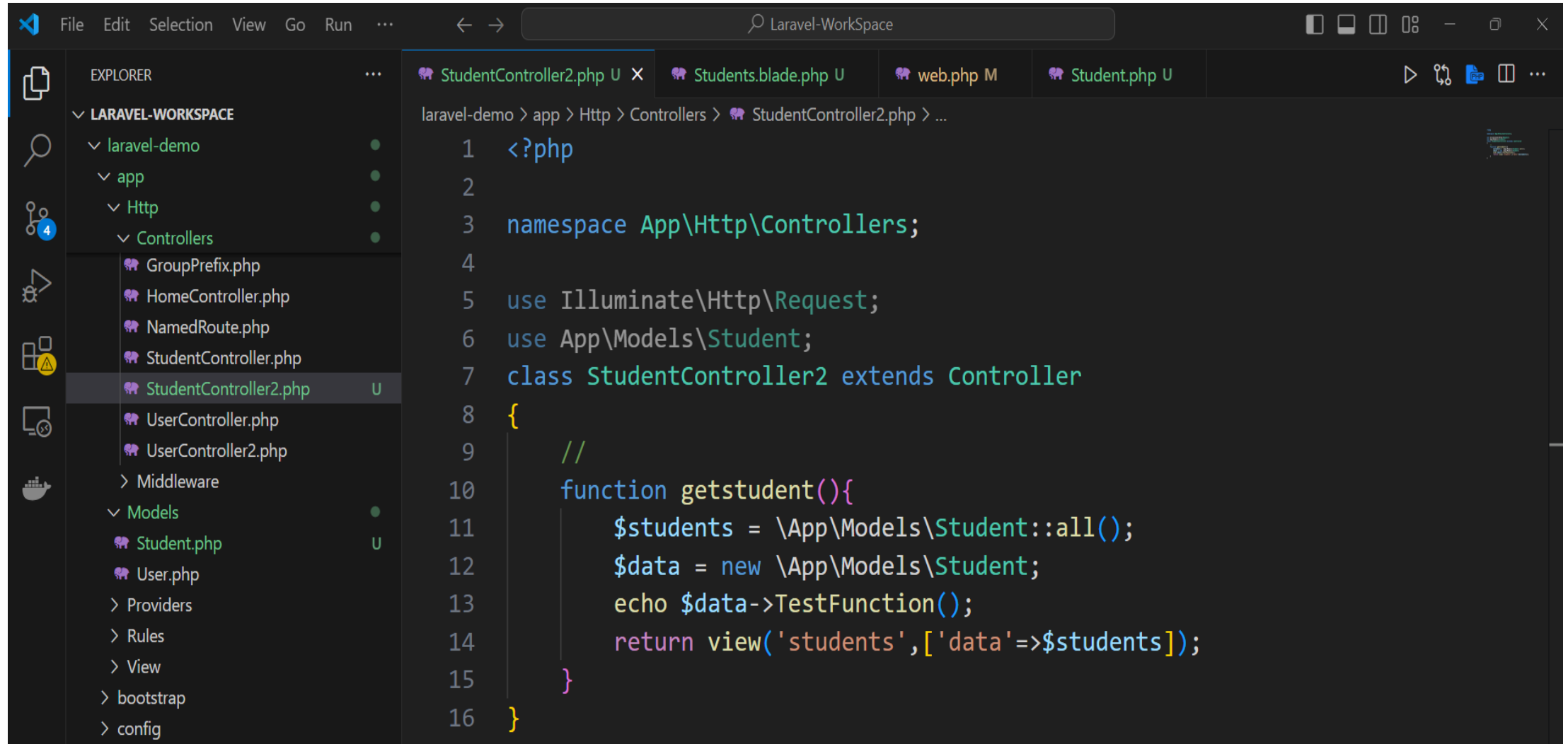
php

 Copy code

```
$users = User::all(); // Fetches all records from the 'user_data' table
```

This approach ensures that Laravel maps the model to the correct table in cases where the naming convention differs.

# Controller (Student Controller2)



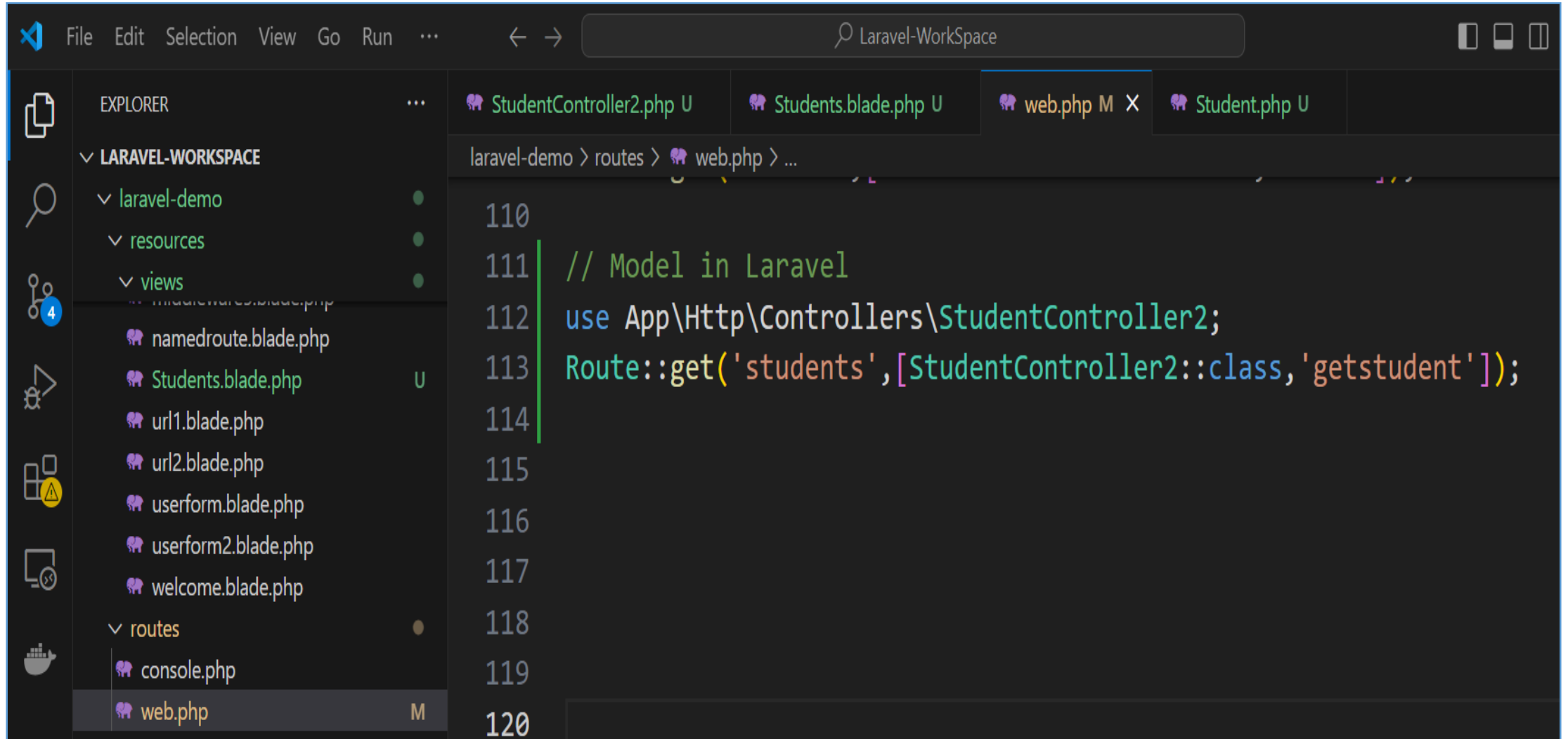
The screenshot shows a code editor with a dark theme. On the left is the Explorer sidebar showing the project structure. The main editor area displays the code for StudentController2.php. The breadcrumb path is 'laravel-demo > app > Http > Controllers > StudentController2.php > ...'. The code is as follows:

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\Student;
7 class StudentController2 extends Controller
8 {
9     //
10    function getstudent(){
11        $students = \App\Models\Student::all();
12        $data = new \App\Models\Student;
13        echo $data->TestFunction();
14        return view('students',['data'=>$students]);
15    }
16 }
```

The Explorer sidebar on the left shows the following structure:

- LARAVEL-WORKSPACE
  - laravel-demo
    - app
      - Http
        - Controllers
          - GroupPrefix.php
          - HomeController.php
          - NamedRoute.php
          - StudentController.php
          - StudentController2.php** U
          - UserController.php
          - UserController2.php
        - Middleware
        - Models
          - Student.php** U
          - User.php
        - Providers
        - Rules
        - View
        - bootstrap
        - config

# Route (students)

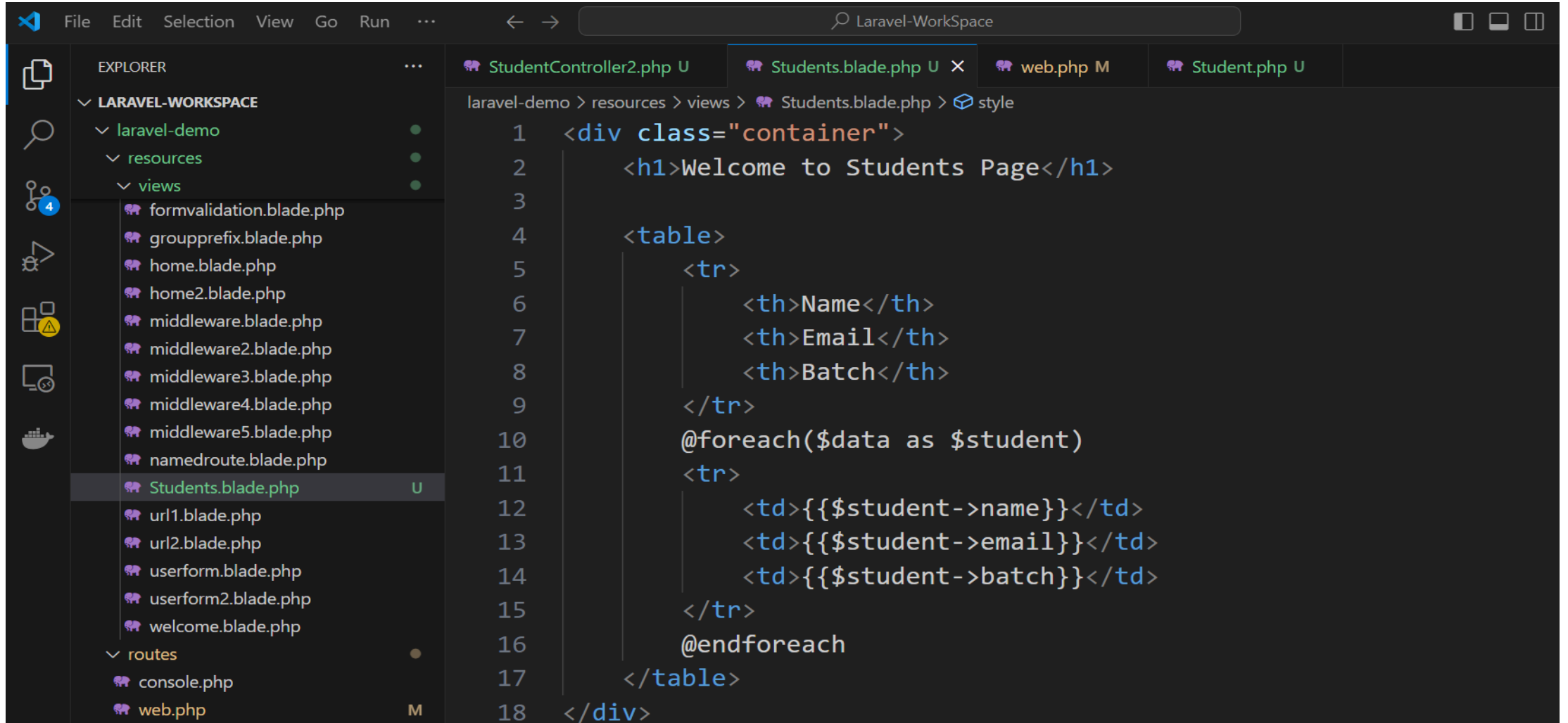


The screenshot shows the Visual Studio Code interface with a Laravel project. The Explorer sidebar on the left displays the project structure under 'LARAVEL-WORKSPACE', including 'laravel-demo' with subfolders 'resources' and 'views'. The 'routes' folder is expanded, showing 'console.php' and 'web.php'. The 'web.php' file is selected and its content is displayed in the main editor. The code defines a GET route for 'students' using the 'StudentController2' class and the 'getstudent' method. The breadcrumb at the top of the editor indicates the path: 'laravel-demo > routes > web.php > ...'.

```
110
111 // Model in Laravel
112 use App\Http\Controllers\StudentController2;
113 Route::get('students', [StudentController2::class, 'getstudent']);
114
115
116
117
118
119
120
```



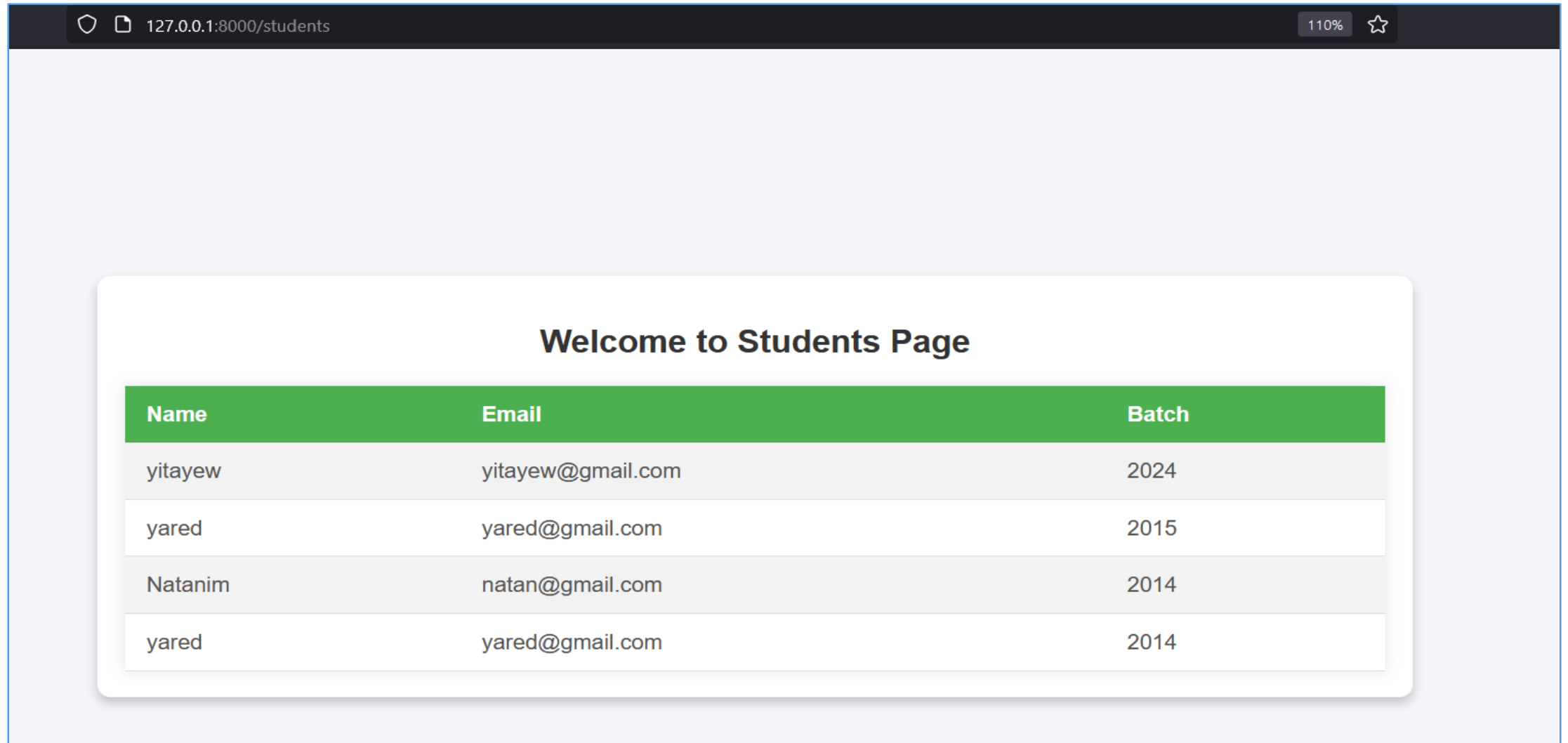
# View (students)



The screenshot shows a code editor with a dark theme. On the left is the Explorer sidebar showing the project structure. The main editor area displays the content of `Students.blade.php`. The code is a Blade template that creates a table to display a list of students. The table has columns for Name, Email, and Batch. The data is iterated over using the `@foreach` directive.

```
laravel-demo > resources > views > Students.blade.php > style
1  <div class="container">
2      <h1>Welcome to Students Page</h1>
3
4      <table>
5          <tr>
6              <th>Name</th>
7              <th>Email</th>
8              <th>Batch</th>
9          </tr>
10         @foreach($data as $student)
11             <tr>
12                 <td>{{ $student->name }}</td>
13                 <td>{{ $student->email }}</td>
14                 <td>{{ $student->batch }}</td>
15             </tr>
16         @endforeach
17     </table>
18 </div>
```

# Output (<http://127.0.0.1:8000/students>)



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8000/students'. The page content is centered and features a heading 'Welcome to Students Page' above a table. The table has three columns: 'Name', 'Email', and 'Batch'. It contains four rows of student data. The browser interface includes a dark header bar with a shield icon, a document icon, and a star icon, along with a zoom level of 110%.

Name	Email	Batch
yitayew	yitayew@gmail.com	2024
yared	yared@gmail.com	2015
Natanim	natan@gmail.com	2014
yared	yared@gmail.com	2014

# Model Inspect in Laravel

- In Laravel, Model Inspection allows you to gain insights into the **structure** and **properties** of a model, including its *table name*, *attributes*, *relationships*, *scopes*, and more.
- Laravel provides several methods and tools that you can use to inspect models, which is particularly useful when **debugging** or **developing** complex applications.

# Cont. ...

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

## ✓ TERMINAL

● \$ php artisan model:show student

```
App\Models\Student .....  
Database ..... mysql  
Table ..... college_students
```


```
Attributes ..... type / cast  
id increments, unique ..... int(30) / int  
name ..... varchar(100)  
email ..... varchar(100)  
batch ..... varchar(100)
```

```
Relations .....
```

```
Events .....
```

```
Observers .....
```

# Common Model inspection tools/methods

Tool/Method	Description	Example Code
<code>all()</code>	Retrieves all records from the model's associated table.	<code>User::all();</code>
<code>find(\$id)</code>	Fetches a record by its primary key.	<code>User::find(1);</code>
<code>where()</code>	Filters records based on specified conditions.	<code>User::where('role', 'admin')-&gt;get();</code>
<code>first()</code>	Retrieves the first record that matches the query.	<code>User::where('active', 1)-&gt;first();</code>
<code>pluck()</code>	Retrieves a single column's values as an array.	<code>User::pluck('email');</code>
<code>count()</code>	Returns the number of records that match the query.	<code>User::count();</code>
<code>exists()</code>	Checks if any records match the query. 	<code>User::where('email', 'test@example.com')-&gt;exists();</code>

## Cont. ...

<code>getTable()</code>	Returns the table name associated with the model.	<code>(new User)-&gt;getTable();</code>
<code>getFillable()</code>	Retrieves the fillable attributes of the model.	<code>(new User)-&gt;getFillable();</code>
<code>toArray()</code>	Converts the model or collection to an array.	<code>\$user-&gt;toArray();</code>
<code>toJson()</code>	Converts the model or collection to JSON format.	<code>\$user-&gt;toJson();</code>
<code>getAttributes()</code>	Fetches all the model's attributes as an associative array.	<code>\$user-&gt;getAttributes();</code>
<code>refresh()</code>	Refreshes the model with fresh data from the database.	<code>\$user-&gt;refresh();</code>
<code>load()</code>	Eager loads related models to avoid N+1 query issues.	<code>\$user-&gt;load('posts');</code>

## Cont. ...

<code>withTrashed()</code>	Includes soft-deleted records in the query (requires <code>SoftDeletes</code> ).	<code>User::withTrashed()-&gt;get();</code>
<code>onlyTrashed()</code>	Fetches only soft-deleted records (requires <code>SoftDeletes</code> ).	<code>User::onlyTrashed()-&gt;get();</code>
<code>getConnection()</code>	Retrieves the database connection associated with the model.	<code>(new User)-&gt;getConnection();</code>
<code>getKey()</code>	Retrieves the primary key value of the model instance.	<code>\$user-&gt;getKey();</code>
<code>isDirty()</code>	Checks if the model has unsaved changes.	<code>\$user-&gt;isDirty();</code>
<code>wasChanged()</code>	Checks if the model's attributes were changed after the last save.	<code>\$user-&gt;wasChanged('email');</code>

This table provides a concise overview of the tools and methods used for inspecting and interacting with models in Laravel.



# Thank you!

Appreciate your action.