Telerik Academy

Web Technologies

Lecture#25

Muhammad Kamran

API

Telerik Academy

Contents

1. Cookies
2. Session
3. Session in laravel

Telerik Academy

How to Create an API using Laravel

- An Application Programming Interface, denoted as API, enables applications to access data and other external software functionalities. APIs are gaining popularity among developers since they save time and resources. Companies do not need to develop complex systems from scratch.
- They can opt to consume data from other existing frameworks. An API is responsible in returning the appropriate response whenever an application sends a request.

- Laravel is a vital PHP framework used in creating interactive websites and APIs. It supports numerous dependencies and templates. Laravel lays a foundation that allows you, as a developer, to focus on more demanding things. It's popular due to its ability to support real-time communication, API authentication, and job queues

1. Creating the project

- **A new Laravel project is created by the command below. You can substitute taskmanager with any project name.**

```
Laravel new taskmanager.
```

Alternatively, you can use Composer to install the required dependencies.

```
composer create-project -prefer-dist Laravel/Laravel taskmanager
```

Ensure that the project you create is in the xampp/htdocs folder. You can view your generated website template by visiting localhost/taskmanager/public/ on your browser.

2. Creating the database

2. Creating the database

- Open Xampp and launch phpMyAdmin. Use it to create a database called tasks. We will create tables and insert data using migrations. You can now open the Laravel project in your preferred IDE. Visual Studio Code will be used for this project.
- The database can be changed to tasks in the .env. You can also change the authentication information including passwords and emails in the .env file.. This largely depends on the Xampp settings.

3. Migration

- We use the php artisan make:model Task -mf command to create a model.
- The '-mf' portion generates a task factory and database migration files for this model.
- The new files are stored in the factories and migrations folders.
- Open the 020_11_25_173913_create_tasks_table file and go to the up() function. We need to outline the names of our database columns in the up() function as shown below.

```php
class CreateTasksTable extends Migration{

  // Run the migrations.
  @return void

  public function up()  {

      Schema::create('tasks', function (Blueprint $table) {
          $table->id(); //table column
          $table->string('title'); //table column
          $table->string('description');//table column
          $table->timestamps();//table column
      });
  }

  // Reverse the migrations.
  @return void

  public function down(){
      Schema::dropIfExists('tasks');
```

Telerik Academy

4. Seeding data

- This stage entails adding some dummy data to our database.
- Let's create a TaskFactory class by using php artistan make:factory TaskFactory command.
- We then need to define the columns that will have fake data as follows.

```php
<?php

namespace Database\Factories;
use App\Models\Task;
use Illuminate\Database\Eloquent\Factories\Facto

class TaskFactory extends Factory

{

  // The name of the factory's corresponding mod
  @var string

  protected $model = Task::class;

  // Define the model's default state.
  @return array

  public function definition(){
      return [
          'title'=>$this->faker->word,
```

5. Controller

- The next step is to create a task controller by using php artisan make:controller TaskController --resource command. The Controller class helps in handling requests.

- You can find the created file at App\http\Controllers\TaskController.

- The --resource portion allows Laravel to add functions that support CRUD functionalities in the controller. The generated methods are index(), create(), store(), show(), edit(), and update().

```php
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;

class TaskController extends Controller {

  // Display a listing of the resource.
  @return \Illuminate\Http\Response

  public function index() {
      //showing all values present in the database
  }

  // Show the form for creating a new resource.

  @return \Illuminate\Http\Response

  public function create() {
      //not required in APIs
  }
```

```php
// Store a newly created resource in storage.
@param  \Illuminate\Http\Request  $request

@return \Illuminate\Http\Response

public function store(Request $request) {
    //storing new values in the database
}

// Display the specified resource.
@param  int  $id

@return \Illuminate\Http\Response

public function show($id) {
//viewing a particular task from a database
}

// Show the form for editing the specified resource.
 @param  int  $id
 @return \Illuminate\Http\Response

 */

public function edit($id) {
    //editing data
}
```

Let's modify the generated functions to activate the *CRUD* functionalities.

A) Index

This method will return all the data or tasks in the database.

```
public function index(){
    return Task::orderBy('created_at', 'asc')->get();  //returns value
}
```

B) Show

This method allows us to retrieve values of a specific object.

```
public function show($id) {
    return Task::findorFail($id); //searches for the object in the dat
}
```

C) Store

This method allows us to receive user inputs and store them in the database.

```
public function store(Request $request){

      $this->validate($request, [ //inputs are not empty or null
          'title' => 'required',
          'description' => 'required',
      ]);

      $task = new Task;
      $task->title = $request->input('title'); //retrieving user inputs
      $task->description = $request->input('description');  //retrieving
      $task->save(); //storing values as an object
      return $task; //returns the stored value if the operation was succ
  }
```

D) Update

This method allows the user to update existing values in the database.

```
public function update(Request $request, $id){
        $this->validate($request, [ // the new values should not be null
          'title' => 'required',
          'description' => 'required',
      ]);

      $task = Task::findorFail($id); // uses the id to search values tha
      $task->title = $request->input('title'); //retrieves user input
      $task->description = $request->input('description');////retrieves
      $task->save();//saves the values in the database. The existing dat
      return $task; // retrieves the updated object from the database
```

E) Destroy

This function is used to delete values in the database. It searches for an object in the database using the provided ID and deletes it.

```
public function destroy($id){ //receives an object's id
     $task = Task::findorFail($id); //searching for object in database
     if($task->delete()){ //deletes the object
         return 'deleted successfully'; //shows a message when the dele
     }
 }
```

6. Registering and Listing routes

We can register our Controller in the `api.php` file, as shown below. Routes are declared automatically.

```
Route::resource('tasks', TaskController::class);
```

```
php artisan route:list
```

Here are the available routes in the `taskmanager` project.

```
$ php artisan route:list
+--------+----------+----------------------+-------------+-----------------------------------------------------+------------+
| Domain | Method   | URI                  | Name        | Action                                              | Middleware |
+--------+----------+----------------------+-------------+-----------------------------------------------------+------------+
|        | GET|HEAD | /                    |             | Closure                                             | web        |
|        | GET|HEAD | api/tasks            | tasks.index | App\Http\Controllers\TaskController@index           | api        |
|        | POST     | api/tasks            | tasks.store | App\Http\Controllers\TaskController@store           | api        |
|        | GET|HEAD | api/tasks/create     | tasks.create| App\Http\Controllers\TaskController@create          | api        |
|        | GET|HEAD | api/tasks/{task}     | tasks.show  | App\Http\Controllers\TaskController@show            | api        |
|        | PUT|PATCH| api/tasks/{task}     | tasks.update| App\Http\Controllers\TaskController@update          | api        |
|        | DELETE   | api/tasks/{task}     | tasks.destroy| App\Http\Controllers\TaskController@destroy        | api        |
|        | GET|HEAD | api/tasks/{task}/edit| tasks.edit  | App\Http\Controllers\TaskController@edit            | api        |
+--------+----------+----------------------+-------------+-----------------------------------------------------+------------+
```

Telerik Academy

PHP Basics

http://academy.telerik.com