Laravel provides various tools and features to help build modern, efficient web applications. One of the key features of Laravel is its ability to handle HTTP requests and generate responses. In this tutorial, you will learn how to use the Laravel Response class to generate different types of responses in your application.

in Laravel, a Response instance represents an HTTP response. It allows setting HTTP status codes, headers, and response content. It also provides various functions for generating different types of responses, such as JSON responses, file downloads, redirects, and more.

Here is an example of how the Response class can be used to send a JSON response which is often used for API responses:

Example:

use Illuminate\Http\Response;

Route::get('/', function () {

    return response()->json([

        'message' => 'Hello, World!'

    ]);

});

In the above code, the response() helper function creates a new response instance, and the json() function sets the HTTP status code to 200 (OK) and the Content-Type header to application/json. The array passed to the json() function will be encoded as JSON and sent as the response body.

Response TypesIn Laravel, different types of responses can be returned from routes and controllers. Some common types of responses include:

In this Laravel file upload example tutorial, we will generate two routes one for creating a form with getting method and another route for file uploading or post file upload data.

Home  »  Laravel   »   Laravel 10 File Upload Tutorial: Validation + Store in Database

Laravel 10 File Upload Tutorial: Validation + Store in Database

Updated: September 20, 2023 | Created: June 1, 2020 by Digamber

This is a step by step Laravel File Upload tutorial with example, and In this tutorial, we will learn how to upload files in Laravel with basic validation in MySQL database.

This tutorial will cover Laravel file uploading concepts:

Laravel project Installation

Route creation and configuration

Blade file creation

Database migration in Laravel

Implement validation on file uploading component

File uploading status via displaying messages

Allow uploading only specific file types. e.g .csv, .txt, .xlx, .xls, .pdf with file size limitation max upto 2MB

Storing an uploaded file in the database

If you are a beginner in Laravel development, you must check out our detailed tutorial on Building Laravel CRUD Web Application.

Table of Contents

File Upload in Laravel Example

Install Laravel Project

Connect to Database

Create File Model & Migrations

Create Routes

Create File Upload Controller

Create Blade File

Conclusion

File Upload in Laravel 10 Example

In this Laravel file upload example tutorial, we will generate two routes one for creating a form with getting method and another route for file uploading or post file upload data.

We develop a simple form using Bootstrap and its Form UI component.

It will allow us to choose a file that needs to be uploaded in the storage > public > uploads folder. We will also configure the database model and store the file path along with its name in the MySQL database.

Install Laravel Project

Open command-line tool and execute the following command to create a Laravel project from scratch.

composer create-project laravel/laravel --prefer-dist laravel-file-upload

Bash

Get into the freshly installed laravel project’s directory.

cd laravel-file-upload

Bash

Connect to Database

You can use MAMP or XAMPP as a local web server for uploading files to storage in laravel. Define a database name in MySQL and add the correct configuration in .env file.

Laravel includes Eloquent, an object-relational mapper (ORM) that makes it enjoyable to interact with your database. When using Eloquent, each database table has a corresponding "Model" that is used to interact with that table. In addition to retrieving records from the database table, Eloquent models allow you to insert, update, and delete records from the table as well.

Eloquent Model ConventionsModels generated by the make:model command will be placed in the app/Models directory. Let's examine a basic model class and discuss some of Eloquent's key conventions:

<?php namespace App\Models; use Illuminate\Database\Eloquent\Model; class Flight extends Model{    // ...}

Table NamesAfter glancing at the example above, you may have noticed that we did not tell Eloquent which database table corresponds to our Flight model. By convention, the "snake case", plural name of the class will be used as the table name unless another name is explicitly specified. So, in this case, Eloquent will assume the Flight model stores records in the flights table, while an AirTrafficController model would store records in an air\_traffic\_controllers table.

If your model's corresponding database table does not fit this convention, you may manually specify the model's table name by defining a table property on the model:

<?php namespace App\Models; use Illuminate\Database\Eloquent\Model; class Flight extends Model{    /\*\*     \* The table associated with the model.     \*     \* @var string     \*/    protected $table = 'my\_flights';}

Primary KeysEloquent will also assume that each model's corresponding database table has a primary key column named id. If necessary, you may define a protected $primaryKey property on your model to specify a different column that serves as your model's primary key:

<?php namespace App\Models; use Illuminate\Database\Eloquent\Model; class Flight extends Model{    /\*\*     \* The primary key associated with the table.     \*     \* @var string     \*/    protected $primaryKey = 'flight\_id';}In addition, Eloquent assumes that the primary key is an incrementing integer value, which means that Eloquent will automatically cast the primary key to an integer. If you wish to use a non-incrementing or a non-numeric primary key you must define a public $incrementing property on your model that is set to false:

<?php class Flight extends Model{    /\*\*     \* Indicates if the model's ID is auto-incrementing.     \*     \* @var bool     \*/    public $incrementing = false;}If your model's primary key is not an integer, you should define a protected $keyType property on your model. This property should have a value of string:

<?php class Flight extends Model{    /\*\*     \* The data type of the auto-incrementing ID.     \*     \* @var string     \*/    protected $keyType = 'string';}

"Composite" Primary KeysEloquent requires each model to have at least one uniquely identifying "ID" that can serve as its primary key. "Composite" primary keys are not supported by Eloquent models. However, you are free to add additional multi-column, unique indexes to your database tables in addition to the table's uniquely identifying primary key.