Building Your First Laravel 10 CRUD Application in Under 30 Minutes

Udara Liyanage · Follow

8 min read · Dec 7, 2023

Have you heard about Laravel 10? It's the latest version of the popular PHP framework that makes building web applications a breeze. And in this tutorial, I'll show you how to build a fully functional CRUD (Create, Read, Update, Delete) application in under 30 minutes using Laravel 10 by creating a phone book app.

Table of content

1. Introduction

- Explain what a CRUD application is and why it's important

2. Prerequisites

- List the tools and software needed to follow along with the tutorial
- Provide links for downloading Laravel 10 and other necessary resources

3. Setting Up Your Environment

- Explain how to install and set up Laravel 10
- Walk through creating a new Laravel 10 project

4. Creating Your Database

- Walk through creating a new MySQL database and setting up your .env file

5. Building Your First Model

- Walk through creating a new model and its migration file

6. Creating Your First Controller

- Walk through creating a new controller for your CRUD application

7. Implementing CRUD Operations

- Discuss the four main CRUD operations (Create, Read, Update, Delete)
- Walk through implementing each CRUD operation for your application

8. Conclusion

1. Introduction

CRUD is an acronym that comes from the world of computer programming and refers to the four functions that are considered necessary to implement a persistent storage application: create, read, update and delete.

In this blog post, you will learn how to build a fully functional CRUD (Create, Read, Update, Delete) application using the latest version of Laravel, Laravel 10. This beginner-friendly tutorial will guide you through setting up your development environment, creating a database, building models, controllers, and views, and implementing CRUD operations. By the end of the article, you will have a solid understanding of how to use Laravel 10 to build powerful and dynamic web applications with ease.

2. Prerequisites

Keep in mind you have to fulfill below requirements before start the development on Laravel 10.

- You need to have updated PHP version ((8.1–8.2)
- Updated Composer version
- Code Editor (Vs Code https://code.visualstudio.com/ , PHPStorm https://www.jetbrains.com/phpstorm/)
- A development server (XAMPP , WAMP or any other)

You can check this by typing the command below on your terminal/CMD.

```
PHP -v
```

3. Setting Up Your Environment

If you have completed the above step successfully, now you can create a Laravel project.

```
composer create-project laravel/laravel laravel-10-crud
```

4. Creating Your Database

Then open the development server (XAMPP in my case) and then start MySQL and apache services .

Then you can open your favorite browser and browse this URL :- **http://localhost/phpmyadmin** (if you are using XAMPP or WAMP servers). Then you will navigate to the PhpMyadmin page. You can create a database from here..

Click on ‘New’ and give a name to your database. In my case I used laravel-10-db as my database name for this project and then created.

Now we have let Laravel know **‘laravel-10-db’** is the database name that we are going to use. To do this you can go to your project folder and open file called **.env**

Then edit below fields like below.

```
DB_DATABASE = laravel-10-db
DB_USERNAME = root
DB_PASSWORD =
```

We use root as our default database user name and also not provide any password to this account. Because it is null by default. If you have your own username and password for your database you can use it here.

Keep in mind.. You have to restart your project every single time after changing the .env file. If you have not run your project yet, leave this message.

5. Building Your First Model

The Model corresponds to the letter ‘M’ in the **MVC framework**. The motive of Models in any Model view controller framework web application is to manage business logic. A Model is nothing but a class in the Laravel framework. This class is responsible for interacting with underlying database tables. Laravel with Eloquent ORM is capable of handling the database more efficiently. ORM stands for ‘Object Relational Mapper’ and it is responsible for your swift interaction with the database. These Models provide you with an easy way to update, insert or retrieve data from the tables.

So today we are going to create a phone book and. Because of that we

need to have contacts in our database. So now we can create a model called ‘Contact’ by following the below command.

```
php artisan make:model Contact -m
```

-m means migration. When we use -m with the model make command, it will automatically create a migration file. In this model we need to have 3 fields to store the values in the database. We need to store the user’s name, phone number and email. So let’s configure our migration file like below. You can find this file under this /database/migrations route.

2023_04_11_071146_create_contacts_table.php

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('contacts', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('phone');
            $table->string('email');
            $table->timestamps();
        });
    }

    /**
```

```
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('contacts');
    }
};
```

It's time to migrate the model into the database. To do this run below command in your terminal/cmd.

```
Php artisan migrate
```

After running this command you will be able to see some new tables added under your database.

6. Creating Your First Controller

A controller is the C in the MVC model–view–controller, and Laravel is an **MVC framework**. The controller is where you enter or code most of your application logic. The good news is that you can create as many controllers as your application needs, allowing your application codes to have an easy-to-understand syntax.

Regarding this project we can create a controller called '**ContactController**' by following the below command.

```
php artisan make:controller ContactController
```

All the controllers are located under the app/http/controllers route in Laravel. If you navigate to the above route, you can see our newly created controller.

7. Implementing CRUD Operations

First we need to create our form to submit data into the database. I use bootstrap because of the simplicity. This is not a UI blog so I'm not much focused about the UI in this tutorial.

Bootstrap CDN :- https://getbootstrap.com/

Add bootstrap CDN into your view's head tag. In this blog i use default welcome.blade.php as my main page so I added the above CDN links inside this page like below. You can find this page under resources/views route

Welcome.blade.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Phone Book App</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css
```

```
    integrity="sha384-KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js
        </script>
    </head>
    <body>
    </body>
    </html
```

Now I need to create a form to insert data.

```
    <div class="container mt-5">
        <form method="POST" action="/addcontact">
            @csrf
            <div class="form-group mb-2">
                <label for="exampleInputEmail1">Email address</label>
                <input type="email" class="form-control" name="email" placeholde
            </div>
            <div class="form-group mb-2">
                <label for="exampleInputPassword1">Phone Number</label>
                <input type="text" class="form-control" name="phone" placeholder
            </div>
            <div class="form-group mb-2">
                <label for="exampleInputPassword1">Name</label>
                <input type="text" class="form-control" name="name" placeholder=
            </div>
            <button type="submit" class="btn btn-primary">Submit</button>
        </form>
    </div>
```

We use the POST method to send the data from frontend to backend and also I use ‘/addcontact’ as my router to make this post request. We will create this later. Now just write like this.

Keep in mind to add @csrf inside the form tag.

Now we are going to create All routes for this CRUD application. Open your web.php file and edit it like below.

Then we can create a new blade file for the edit page. Create a new file called ‘Edit.blade.php’ inside the resources/views/ folder and edit that file like below.

edit.blade.php

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Phone Book App</title>
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist
            integrity="sha384-KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SX
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dis
            integrity="sha384-ENjdO4Dr2bkBIFxQpeoTz1HIcje39Wm4jDKdf19U8gI4dd
        </script>
    </head>
    <body>
        <div class="container mt-5">
            <form method="POST" action="/edit/{{$contact->id}}">
                @csrf
                <div class="form-group mb-2">
                    <label for="exampleInputEmail1">Email address</label>
                    <input type="email" class="form-control" name="email" pl
                </div>
                <div class="form-group mb-2">
                    <label for="exampleInputPassword1">Phone Number</label>
                    <input type="text" class="form-control" name="phone" pla
                </div>
                <div class="form-group mb-2">
                    <label for="exampleInputPassword1">Name</label>
                    <input type="text" class="form-control" name="name" plac
                </div>
                <button type="submit" class="btn btn-primary">Update</button
```

```
            </form>
        </div>
    </body>
</html>
```

Web.php

```
<?php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ContactController;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider and all of them will
| be assigned to the "web" middleware group. Make something great!
|
*/

Route::get('/', [ContactController::class, 'index']);
Route::post('/addcontact', [ContactController::class, 'add']);
Route::get('/delete/{id}', [ContactController::class, 'delete']);
Route::get('/edit/{id}', [ContactController::class, 'edit']);
Route::post('/edit/{id}', [ContactController::class, 'update']);
```

Then we can create all our CRUD operations inside our controller.

ContactController.php

```php
<?php

namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Models\Contact;

class ContactController extends Controller
{
    public function index(Request $req){
        $contact = Contact::all();
        return  view('welcome')->with("contact",$contact);
    }
    public function add(Request $req){
        $contact = new Contact;
        $contact->email = $req->email;
        $contact->phone = $req->phone;
        $contact->name = $req->name;
        $contact->save();
        return redirect()->back();
    }
    public function delete(Request $req){
        $contact = Contact::find($req->id);
        $contact->delete();
        return redirect()->back();
    }
    public function edit(Request $req){
        $contact = Contact::find($req->id);
        return view('edit')->with("contact",$contact);
    }
    public function update(Request $req){
        $contact = Contact::find($req->id);
        $contact->update([
            'name' => $req->name,
            'phone' => $req->phone,
            'email' => $req->email,
        ]);
        return redirect()->back();
    }
}
```

Note:- update your model like below to make the update work on the model.

Contact.php

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Contact extends Model
{
    use HasFactory;
    protected $fillable = [
        'name',
        'email',
        'phone',
    ];
}
```

Then we need to have a table to display data under the above form. To do that add below code inside the welcome.blade.php file under your form.

```
<table class="table mt-5">
    <thead>
        <tr>
            <th scope="col">Id</th>
            <th scope="col">Name</th>
            <th scope="col">Phone</th>
            <th scope="col">Email</th>
            <th scope="col">Action</th>
        </tr>
    </thead>
    <tbody>
        @if (count($contact) > 0)
            @foreach ($contact as $cont)
                <tr>
                    <th>{{ $cont->id }}</th>
                    <th>{{ $cont->name }}</th>
                    <th>{{ $cont->phone }}</th>
                    <th>{{ $cont->email }}</th>
                    <th><a href="/edit/{{ $cont->id }}" class="btn btn-prima
                        <a href="/delete/{{ $cont->id }}" class="btn btn-dan
                    </th>
                </tr>
            @endforeach
```

```
        @else
            <tr>
                <th>No Data</th>
            </tr>
        @endif
    </tbody>
</table>
```

Full **welcome.blade.php**

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <title>Phone Book App</title>
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist
            integrity="sha384-KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SX
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dis
            integrity="sha384-ENjdO4Dr2bkBIFxQpeoTz1HIcje39Wm4jDKdf19U8gI4dd
        </script>
    </head>
    <body>
        <div class="container mt-5">
            <form method="POST" action="/addcontact">
                @csrf
                <div class="form-group mb-2">
                    <label for="exampleInputEmail1">Email address</label>
                    <input type="email" class="form-control" name="email" pl
                </div>
                <div class="form-group mb-2">
                    <label for="exampleInputPassword1">Phone Number</label>
                    <input type="text" class="form-control" name="phone" pla
                </div>
                <div class="form-group mb-2">
                    <label for="exampleInputPassword1">Name</label>
                    <input type="text" class="form-control" name="name" plac
                </div>
                <button type="submit" class="btn btn-primary">Submit</button
            </form>
```

```
            <table class="table mt-5">
                <thead>
                    <tr>
                        <th scope="col">Id</th>
                        <th scope="col">Name</th>
                        <th scope="col">Phone</th>
                        <th scope="col">Email</th>
                        <th scope="col">Action</th>
                    </tr>
                </thead>
                <tbody>
                    @if (count($contact) > 0)
                        @foreach ($contact as $cont)
                            <tr>
                                <th>{{ $cont->id }}</th>
                                <th>{{ $cont->name }}</th>
                                <th>{{ $cont->phone }}</th>
                                <th>{{ $cont->email }}</th>
                                <th><a href="/edit/{{ $cont->id }}" class="b
                                    <a href="/delete/{{ $cont->id }}" class=
                                </th>
                            </tr>
                        @endforeach
                    @else
                        <tr>
                            <th>No Data</th>
                        </tr>
                    @endif
                </tbody>
            </table>
        </div>
    </body>
</html>
```

8. Conclusion

Congratulations! You've successfully built your first Laravel 10 CRUD application in under 30 minutes. By following this tutorial, you've learned how to set up your development environment, create a database, build models, controllers, and views, and implement CRUD operations. With Laravel 10's powerful features and intuitive syntax, you can now build

dynamic web applications faster and more efficiently than ever before.

Remember, this tutorial is just the beginning of your journey with Laravel 10. There's always more to learn and explore, from advanced features like Eloquent relationships and middleware to building RESTful APIs and integrating authentication and authorization. So keep learning, keep practicing, and keep building amazing things with Laravel 10!.

Thank you for reading, and I hope this tutorial has been helpful for you. If you have any questions or feedback, feel free to ask from me. Happy coding!.

Fore More Read :- https://udarax.me/laravel/laravel-10-crud-application-in-under-30-minutes/

My Blog Posts :- https://udarax.me/blog/

Laravel | Laravel 10 | Crud | MySQL | Orm

Written by Udara Liyanage

Follow

10 Followers

more about me :- https://udarax.me/

More from Udara Liyanage

Udara Liyanage

PHP CRUD operations with MySQL and HTML Bootstrap...

PHP CRUD Application Road Map

Dec 11, 2023 3 1

Udara Liyanage

Realtime location tracking with React Js : A Beginner-Friendly...

In this article we are going to learn how to implement a Realtime location tracking...

Apr 20, 2023 5

Udara Liyanage

Firebase Read and Write data in JavaScript web SDK

Requirements you need.

Jan 12, 2023 1

Udara Liyanage

Google Gemini Pro AI Integration With React (Vite) Tutorial In Ju...

Are you a budding developer eager to explore the world of AI and React? In this...

Dec 15, 2023 141 1

See all from Udara Liyanage

Recommended from Medium

IAMWYNN

Integrating an Admin Template Into a Laravel 10 Web Application

In our previous tutorial, we learned how to set up a Laravel v10 web application. We'll...

May 29 2

Devendra Dode

Laravel 11 Simple CRUD Application Example

Creating a simple CRUD (Create, Read, Update, Delete) application in Laravel 11 is ...

Mar 31 6

Lists

Umar Farooque Khan

Laravel 11 CRUD Application Example Tutorial

Greetings! Welcome to this tutorial where I'll guide you through creating a Laravel 11...

Apr 14 138 5

Azizan Nur Rohman

Mastering User Authentication in Laravel 11: A Comprehensive...

Hello, this time I will discuss the PHP framework that is most widely used by...

Mar 29 96

Samuel

How to Install Bootstrap 5 with Laravel 10 & 11

Bootstrap is a popular front-end framework that provides a collection of CSS and...

Mar 26 6 1

Milenmk

Laravel CRUD: one for all

Hello. This is my first story here, but I hope it can help a lot of people.

Jan 21 13 1

See more recommendations