# Laravel CRUD Using Repository Pattern

This guide explains how to implement CRUD operations in Laravel using the Repository Pattern, promoting clean, maintainable, and testable code.

## Overview

We’ll create a simple CRUD for a Post model with title and content fields, using a repository interface and implementation.

## Step 1: Create Laravel Project

laravel new blog  
cd blog  
php artisan serve

## Step 2: Create Model and Migration

php artisan make:model Post -m

Edit migration:

Schema::create('posts', function (Blueprint $table) {  
 $table->id();  
 $table->string('title');  
 $table->text('content');  
 $table->timestamps();  
});

Run migration:

php artisan migrate

## Step 3: Create Repository Structure

Create files:

app/Repositories/PostRepositoryInterface.php  
app/Repositories/PostRepository.php

### PostRepositoryInterface

namespace App\Repositories;  
  
interface PostRepositoryInterface {  
 public function all();  
 public function find($id);  
 public function create(array $data);  
 public function update($id, array $data);  
 public function delete($id);  
}

### PostRepository

namespace App\Repositories;  
  
use App\Models\Post;  
  
class PostRepository implements PostRepositoryInterface  
{  
 public function all() {  
 return Post::all();  
 }  
  
 public function find($id) {  
 return Post::findOrFail($id);  
 }  
  
 public function create(array $data) {  
 return Post::create($data);  
 }  
  
 public function update($id, array $data) {  
 $post = Post::findOrFail($id);  
 $post->update($data);  
 return $post;  
 }  
  
 public function delete($id) {  
 return Post::destroy($id);  
 }  
}

## Step 4: Register Binding in Service Provider

// app/Providers/AppServiceProvider.php  
use App\Repositories\PostRepository;  
use App\Repositories\PostRepositoryInterface;  
  
public function register()  
{  
 $this->app->bind(PostRepositoryInterface::class, PostRepository::class);  
}

## Step 5: Create Controller

php artisan make:controller PostController

### PostController

namespace App\Http\Controllers;  
  
use App\Repositories\PostRepositoryInterface;  
use Illuminate\Http\Request;  
  
class PostController extends Controller  
{  
 protected $postRepository;  
  
 public function \_\_construct(PostRepositoryInterface $postRepository) {  
 $this->postRepository = $postRepository;  
 }  
  
 public function index() {  
 $posts = $this->postRepository->all();  
 return view('posts.index', compact('posts'));  
 }  
  
 public function create() {  
 return view('posts.create');  
 }  
  
 public function store(Request $request) {  
 $request->validate([  
 'title' => 'required|string|max:255',  
 'content' => 'required|string',  
 ]);  
  
 $this->postRepository->create($request->only('title', 'content'));  
 return redirect()->route('posts.index')->with('success', 'Post created successfully.');  
 }  
  
 public function edit($id) {  
 $post = $this->postRepository->find($id);  
 return view('posts.edit', compact('post'));  
 }  
  
 public function update(Request $request, $id) {  
 $request->validate([  
 'title' => 'required|string|max:255',  
 'content' => 'required|string',  
 ]);  
  
 $this->postRepository->update($id, $request->only('title', 'content'));  
 return redirect()->route('posts.index')->with('success', 'Post updated successfully.');  
 }  
  
 public function destroy($id) {  
 $this->postRepository->delete($id);  
 return redirect()->route('posts.index');  
 }  
}

## Step 6: Define Routes

// routes/web.php  
use App\Http\Controllers\PostController;  
  
Route::middleware(['auth'])->group(function () {  
 Route::resource('posts', PostController::class);  
});

## Step 7: Blade Views

### resources/views/posts/index.blade.php

<h1>All Posts</h1>  
<a href="{{ route('posts.create') }}">Create Post</a>  
@foreach($posts as $post)  
 <h2>{{ $post->title }}</h2>  
 <p>{{ $post->content }}</p>  
 <a href="{{ route('posts.edit', $post->id) }}">Edit</a>  
 <form method="POST" action="{{ route('posts.destroy', $post->id) }}">  
 @csrf  
 @method('DELETE')  
 <button type="submit">Delete</button>  
 </form>  
@endforeach

### resources/views/posts/create.blade.php

<h1>Create Post</h1>  
<form method="POST" action="{{ route('posts.store') }}">  
 @csrf  
 <input name="title" placeholder="Title" required>  
 <textarea name="content" placeholder="Content"></textarea>  
 <button type="submit">Create</button>  
</form>

### resources/views/posts/edit.blade.php

<h1>Edit Post</h1>  
<form method="POST" action="{{ route('posts.update', $post->id) }}">  
 @csrf  
 @method('PUT')  
 <input name="title" value="{{ $post->title }}" required>  
 <textarea name="content">{{ $post->content }}</textarea>  
 <button type="submit">Update</button>  
</form>

## Step 8: Add Fillable to Post Model

// app/Models/Post.php  
protected $fillable = ['title', 'content'];

## Step 9: Authentication with Laravel Breeze

### Install Breeze

composer require laravel/breeze --dev  
php artisan breeze:install  
npm install && npm run dev  
php artisan migrate

Now you have registration, login, logout, and password reset features. Routes are protected using auth middleware.

## Step 10: API Support (Optional)

### Create API Controller

php artisan make:controller Api/PostApiController

### PostApiController

namespace App\Http\Controllers\Api;  
  
use App\Http\Controllers\Controller;  
use App\Repositories\PostRepositoryInterface;  
use Illuminate\Http\Request;  
  
class PostApiController extends Controller  
{  
 protected $postRepository;  
  
 public function \_\_construct(PostRepositoryInterface $postRepository)  
 {  
 $this->postRepository = $postRepository;  
 }  
  
 public function index()  
 {  
 return response()->json($this->postRepository->all());  
 }  
  
 public function show($id)  
 {  
 return response()->json($this->postRepository->find($id));  
 }  
}

### Add API Routes

// routes/api.php  
use App\Http\Controllers\Api\PostApiController;  
  
Route::get('posts', [PostApiController::class, 'index']);  
Route::get('posts/{id}', [PostApiController::class, 'show']);

## Step 11: Unit Testing the Repository

### Create Test

php artisan make:test PostRepositoryTest

### Sample Test

use App\Models\Post;  
use App\Repositories\PostRepository;  
use Illuminate\Foundation\Testing\RefreshDatabase;  
  
class PostRepositoryTest extends TestCase  
{  
 use RefreshDatabase;  
  
 public function test\_create\_post()  
 {  
 $repository = new PostRepository();  
  
 $data = [  
 'title' => 'Test Post',  
 'content' => 'Test Content',  
 ];  
  
 $post = $repository->create($data);  
  
 $this->assertDatabaseHas('posts', ['title' => 'Test Post']);  
 }  
}

## Extra: Repository Artisan Command

Laravel doesn’t include a make:repository command by default. You can:

1. Create manually using php artisan make:class.
2. Use a package like larswoistje/laravel-repository-pattern:

composer require larswoistje/laravel-repository-pattern  
php artisan make:repository Post

1. Or build your own custom command.

Now you have a fully working CRUD in Laravel using the Repository Pattern with authentication, form validation, optional API, and testing support!