# Refactoring SubCategoryController to Follow SOLID Principles

In this document, we will outline the steps taken to refactor the SubCategoryController in a Laravel project to adhere to SOLID principles. The goal was to maintain simplicity while applying key principles such as Single Responsibility Principle (SRP), Dependency Inversion Principle (DIP), and Interface Segregation Principle (ISP). This approach ensures a clean, maintainable, and testable codebase.

## Step-by-Step Explanation

### Step 1: Create a SubCategoryRepositoryInterface

Why? The Dependency Inversion Principle (DIP) dictates that high-level modules should not depend on low-level modules but on abstractions. The SubCategoryRepositoryInterface allows the controller to depend on an interface rather than a concrete class.

Code:  
```php  
namespace App\Repositories;  
  
interface SubCategoryRepositoryInterface {  
 public function getAll();  
 public function store(array $data);  
 public function getById($id);  
 public function update(array $data, $id);  
 public function delete($id);  
}  
```

### Step 2: Implement the Repository Interface

Why? This implementation (SubCategoryRepository) allows data interaction to be managed separately from the controller. It adheres to the Single Responsibility Principle (SRP) by keeping database logic out of the controller.

Code:  
```php  
namespace App\Repositories;  
  
use Illuminate\Support\Facades\DB;  
  
class SubCategoryRepository implements SubCategoryRepositoryInterface {  
 public function getAll() {  
 return DB::table('subcategories')  
 ->join('categories', 'subcategories.category\_id', 'categories.id')  
 ->select('subcategories.\*', 'categories.category\_en')  
 ->orderBy('id', 'desc')->paginate(4);  
 }  
  
 public function store(array $data) {  
 return DB::table('subcategories')->insert($data);  
 }  
  
 public function getById($id) {  
 return DB::table('subcategories')->where('id', $id)->first();  
 }  
  
 public function update(array $data, $id) {  
 return DB::table('subcategories')->where('id', $id)->update($data);  
 }  
  
 public function delete($id) {  
 return DB::table('subcategories')->where('id', $id)->delete();  
 }  
}  
```

### Step 3: Create a Service Layer for SubCategory

Why? The Service Layer (SubCategoryService) acts as an intermediary between the controller and the repository. It handles business logic, ensuring the controller remains lean and focused on request handling only.

Code:  
```php  
namespace App\Services;  
  
use App\Repositories\SubCategoryRepositoryInterface;  
  
class SubCategoryService implements SubCategoryServiceInterface {  
 protected $subCategoryRepository;  
  
 public function \_\_construct(SubCategoryRepositoryInterface $subCategoryRepository) {  
 $this->subCategoryRepository = $subCategoryRepository;  
 }  
  
 public function getAllSubCategories() {  
 return $this->subCategoryRepository->getAll();  
 }  
  
 public function storeSubCategory(array $data) {  
 return $this->subCategoryRepository->store($data);  
 }  
  
 public function getSubCategoryById($id) {  
 return $this->subCategoryRepository->getById($id);  
 }  
  
 public function updateSubCategory(array $data, $id) {  
 return $this->subCategoryRepository->update($data, $id);  
 }  
  
 public function deleteSubCategory($id) {  
 return $this->subCategoryRepository->delete($id);  
 }  
}  
```

### Step 4: Refactor SubCategoryController

Why? The controller is now only responsible for handling HTTP requests, delegating all business logic to the service layer. This adheres to both SRP and DIP by ensuring the controller is not tightly coupled to the data source.

Code:  
```php  
namespace App\Http\Controllers\Backend;  
  
use App\Http\Controllers\Controller;  
use App\Services\SubCategoryServiceInterface;  
use Illuminate\Http\Request;  
use Illuminate\Http\RedirectResponse;  
use Illuminate\View\View;  
  
class SubCategoryController extends Controller {  
 protected SubCategoryServiceInterface $subCategoryService;  
  
 public function \_\_construct(SubCategoryServiceInterface $subCategoryService) {  
 $this->subCategoryService = $subCategoryService;  
 }  
  
 public function Index(): View {  
 $subcategories = $this->subCategoryService->getAllSubCategories();  
 return view('backend.subcategory.index', compact('subcategories'));  
 }  
  
 public function AddSubCategory(): View {  
 return view('backend.subcategory.create');  
 }  
  
 public function StoreSubCategory(Request $request): RedirectResponse {  
 $data = $request->only(['subcategory\_en', 'subcategory\_hin', 'category\_id']);  
 $this->subCategoryService->storeSubCategory($data);  
 return redirect()->route('subcategories')->with('message', 'SubCategory Inserted Successfully');  
 }  
  
 public function EditSubCategory($id): View {  
 $subcategory = $this->subCategoryService->getSubCategoryById($id);  
 return view('backend.subcategory.edit', compact('subcategory'));  
 }  
  
 public function UpdateSubCategory(Request $request, $id): RedirectResponse {  
 $data = $request->only(['subcategory\_en', 'subcategory\_hin', 'category\_id']);  
 $this->subCategoryService->updateSubCategory($data, $id);  
 return redirect()->route('subcategories')->with('message', 'SubCategory Updated Successfully');  
 }  
  
 public function DeleteSubCategory($id): RedirectResponse {  
 $this->subCategoryService->deleteSubCategory($id);  
 return redirect()->route('subcategories')->with('message', 'SubCategory Deleted Successfully');  
 }  
}  
```