# CategoryController Refactoring - Applying SOLID Principles

## Introduction

In this document, we will cover the step-by-step refactoring of the CategoryController in a Laravel application.  
The goal was to apply SOLID principles, focusing on Single Responsibility Principle (SRP) and Dependency Inversion Principle (DIP).  
We also aimed to maintain existing route structures and avoid changes in the routing approach.

## Step-by-Step Refactoring

### Original Code

The original CategoryController directly interacted with the database using raw queries via the DB facade.  
The validation, data handling, and business logic were tightly coupled within the controller.  
Here is the original code:  
  
public function StoreCategory(Request $request){  
  
 $validatedData = $request->validate([  
 'category\_en' => 'required|unique:categories|max:255',  
 'category\_hin' => 'required|unique:categories|max:255',  
 ]);  
  
 $data = array();  
 $data['category\_en'] = $request->category\_en;  
 $data['category\_hin'] = $request->category\_hin;  
 DB::table('categories')->insert($data);  
  
 $notification = array(  
 'message' => 'Category Inserted Successfully',  
 'alert-type' => 'success'  
 );  
  
 return Redirect()->route('categories')->with($notification);  
}

### Applying the Single Responsibility Principle (SRP)

We introduced a service layer and repository pattern to separate concerns.  
Validation was moved to Form Requests (CategoryRequest.php) to keep the controller clean.  
  
New Store Method in the Controller:  
  
public function store(CategoryRequest $request): RedirectResponse  
{  
 $data = $request->validated();  
 $this->categoryService->storeCategory($data);  
  
 return redirect()->route('categories.index')->with([  
 'message' => 'Category Inserted Successfully',  
 'alert-type' => 'success'  
 ]);  
}

### Applying the Dependency Inversion Principle (DIP)

We created interfaces for both the repository and service layers to allow for dependency injection.  
This approach promotes flexibility and testability.  
  
Repository Interface:  
  
interface CategoryRepositoryInterface  
{  
 public function getAll();  
 public function store(array $data);  
 public function getById($id);  
 public function update(array $data, $id);  
 public function delete($id);  
}  
  
Service Interface:  
  
interface CategoryServiceInterface  
{  
 public function getAllCategories();  
 public function storeCategory(array $data);  
 public function getCategoryById($id);  
 public function updateCategory(array $data, $id);  
 public function deleteCategory($id);  
}

### Current Status and Issue

The refactored code is more modular and adheres to the SOLID principles.  
However, we encountered an issue with the updateCategory method:  
  
App\Services\CategoryService::updateCategory(): Argument #1 ($data) must be of type array, string given.  
  
This was due to a mismatch in how the request data was passed. The fix was to use:  
  
$data = $request->only(['category\_en', 'category\_hin']);  
  
instead of passing the entire request object directly.

## Summary

In this refactoring process, we applied SRP and DIP to make the CategoryController more maintainable and testable.  
We avoided changes to the routing structure as requested and ensured that all methods follow a clean separation of concerns.  
Further improvements could involve applying the Open-Closed Principle (OCP) and introducing more advanced design patterns if needed.