SOLID Principles in Laravel CRUD

Implementing SOLID principles in Laravel CRUD ensures maintainability and scalability.

1. Single Responsibility Principle (SRP)

Each class should have only one responsibility. We use repositories to separate data handling logic.

Create an Interface:

```
interface ProductRepositoryInterface {
    public function getAll();
    public function getById($id);
    public function create(array $data);
    public function update($id, array $data);
    public function delete($id);
}
```

Implement the Interface:

```
class ProductRepository implements ProductRepositoryInterface {
    public function getAll() { return Product::all(); }
    public function getById($id) { return Product::findOrFail($id); }
    public function create(array $data) { return Product::create($data); }
    public function update($id, array $data) {
        $product = Product::findOrFail($id);
        $product->update($data);
        return $product;
    }
    public function delete($id) {
        $product = Product::findOrFail($id);
        $product->delete();
    }
}
```

2. Open/Closed Principle (OCP)

We use services to extend functionalities without modifying repository logic.

Create a Service Layer:

```
class ProductService {
    private $productRepository;
    public function __construct(ProductRepositoryInterface $productRepository) {
        $this->productRepository = $productRepository;
    }
    public function getAllProducts() { return $this->productRepository->getAll(); }
    public function createProduct(array $data) { return $this->productRepository->create($data); }
}
```

3. Dependency Inversion Principle (DIP)

The controller depends on an abstraction (ProductService) rather than a repository directly.

Controller Implementation:

```
class ProductController extends Controller {
    private $productService;
    public function __construct(ProductService $productService) {
        $this->productService = $productService;
    }
    public function index() { return response()->json($this->productService->getAllProducts()); }
}
```

4. Interface Segregation Principle (ISP)

Avoid forcing classes to implement unused methods by using separate interfaces.

5. Liskov Substitution Principle (LSP)

Derived classes should be interchangeable with base classes. This is ensured by the repository pattern.

6. Define Routes

```
use App\Http\Controllers\ProductController;
Route::get('/products', [ProductController::class, 'index']);
Route::get('/products/{id}', [ProductController::class, 'show']);
Route::post('/products', [ProductController::class, 'store']);
```

```
Route::put('/products/{id}', [ProductController::class, 'update']);
Route::delete('/products/{id}', [ProductController::class, 'destroy']);
```