Laravel CRUD Generator – Technical Assessment

Objective:

Develop a **dynamic CRUD Generator** that allows developers to generate models, controllers, requests, and views automatically via a single command.

Scenario:

You are tasked with creating a **custom Laravel CRUD generator** that works with any model schema. The generator should:
✅ Generate models, controllers, requests, and views using a CLI command
✅ Support **relationships** (e.g., one-to-many, many-to-many)
✅ Generate API routes with proper authentication and validation
✅ Generate Eloquent scopes and query filters
✅ Generate blade views using reusable components

Tasks:

1. Create CLI Command

✅ Create a new Laravel Artisan command:

```
php artisan make:crud {model} --fields="name:string, description:text, status:enum(open,closed)" --
relations="tasks:hasMany"
```

✅ The command should:

- Create a **Model** with fillable attributes
- Create a **Migration** with correct data types and indexes
- Create a **Controller** with CRUD methods using RESTful conventions
- Create a **Form Request** for validation
- Create API routes in api.php
- Create **Blade views** using reusable components
- Create relationships based on the --relations flag

2. Model Generation

✅ The generator should create a model like this:

```php
class Project extends Model
{
    protected $fillable = ['name', 'description', 'status'];

    public function tasks()
    {
        return $this->hasMany(Task::class);
    }
}
```

✅ The generator should also create an appropriate migration:

```php
Schema::create('projects', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->text('description');
    $table->enum('status', ['open', 'closed']);
    $table->timestamps();
});

```

✅ Automatically add timestamps, soft deletes, and indexing where applicable.

3. Controller Generation

✅ Generate a controller using a RESTful structure:

```php
class ProjectController extends Controller
{
    public function index()
    {
        return Project::all();
    }

    public function store(ProjectRequest $request)
    {
        $project = Project::create($request->validated());
        return response()->json($project, 201);
    }

    public function show(Project $project)
    {
        return response()->json($project);
    }

    public function update(ProjectRequest $request, Project $project)
    {
        $project->update($request->validated());
        return response()->json($project);
    }

    public function destroy(Project $project)
    {
        $project->delete();
        return response()->json(null, 204);
    }
}

```

✅ Include **API Resource** for structured responses.

4. Request Validation Generation

✅ Create a form request like this:

```
class ProjectRequest extends FormRequest
{
    public function rules()
    {
        return [
            'name' => 'required|string|max:255',
            'description' => 'nullable|string',
            'status' => 'required|in:open,closed'
        ];
    }
}

```

---

5. Route Generation

✅ Generate routes in api.php and web.php:

```
Route::apiResource('projects', ProjectController::class);
```

✅ Include **route model binding** and middleware protection.

6. Blade View Generation

✅ Generate the following views using **Blade components**:

- **index.blade.php** – Display list with pagination
- **create.blade.php** – Form for creating a record
- **edit.blade.php** – Form for editing a record
- **show.blade.php** – Detailed view
- **layout.blade.php** – Use reusable components for styling

✅ Example for the index.blade.php:

```
<x-layout>
    <div class="container">
        <table>
```

```
            @foreach ($projects as $project)
                <tr>
                    <td>{{ $project->name }}</td>
                    <td>{{ $project->status }}</td>
                    <td>
                        <a href="{{ route('projects.edit', $project) }}">Edit</a>
                        <form action="{{ route('projects.destroy', $project) }}" method="POST">
                            @csrf
                            @method('DELETE')
                            <button type="submit">Delete</button>
                        </form>
                    </td>
                </tr>
            @endforeach
        </table>
    </div>
</x-layout>

```

✅ Include responsive design and form validation feedback.

7. Relationship Handling

✅ Handle relationships automatically:

- If --relations="tasks:hasMany", add the hasMany relationship in the model
- Create a corresponding Task model and controller

Example for Task model:

```
class Task extends Model
{
    protected $fillable = ['title', 'description', 'status'];

    public function project()
    {
        return $this->belongsTo(Project::class);
    }
}

```

✅ Handle nested resource routes if necessary.

---

8. Code Review Scenario

You are presented with the following generator command:

```
php artisan make:crud Project --fields="name:string, status:enum(open,closed)"
```

Generated Code:

```
public function store(Request $request)
{
    Project::create($request->all());
}

```

Tasks:

1. Identify the issues in the generated code.
2. Optimize the code to follow Laravel best practices.
3. Explain why your solution is more secure and scalable.

9. System Design & Scaling

Design the generator to handle:

✅ Large codebases with 100+ models
✅ Namespaced models and controllers
✅ Generator output caching to improve performance
✅ Consistent coding style across all generated files

Evaluation Criteria:

✅ Code Quality – Clean and consistent structure
✅ Reusability – Ability to modify and extend generator for future needs
✅ Performance – Efficient handling of large-scale code generation
✅ Security – Proper handling of validation and model binding
✅ Problem Solving – Ability to identify and solve issues
✅ Communication – Clear explanation of generator logic and options

Bonus Points:

➕ Add support for generating **API Resource Controllers**
➕ Support nested relationships (e.g., belongsToMany)
➕ Create a generator dashboard to track generated models and controllers

Submission:

- Codebase should be submitted via GitHub
- Include test cases for the generator
- Provide a README with setup and usage instructions