GitHub Repository Description: Laravel Dynamic Process Flow Module

This repository contains a modular, extensible dynamic process flow system implemented in a Laravel 8.75 application.   
It leverages the Spatie Laravel Permission package and a dynamic route-based middleware system as outlined in the CodeAndDeploy tutorial:  
https://codeanddeploy.com/blog/laravel/how-to-create-a-custom-dynamic-middleware-for-spatie-laravel-permission  
  
This module is designed to be model-agnostic, allowing workflows to be attached to any Eloquent model (e.g., Activity, PsipName) using polymorphic relationships.

# Key Features

• ✅ Model-agnostic workflow engine using polymorphic relationships  
• 🔐 Dynamic permission enforcement using Spatie + route names  
• 🔁 Step-based process flows with support for conditions and actions  
• 🧾 Logging of every step transition  
• 👤 Power user interface (Admin + Planning) for creating and managing flows

## Supported Models

- PsipName: May have a process flow. Automatically marked as "completed" if all related Activity flows are completed.  
- Activity: Must have a process flow.  
- SubActivity: Future support ready.

## Database Tables

• process\_flows — defines a workflow and the model it belongs to  
• process\_steps — stores steps in a flow  
• process\_conditions — stores logic to validate step transition  
• process\_actions — stores logic to execute on step completion  
• process\_instances — active state of a flow for a specific model instance  
• process\_logs — logs every transition, user, and timestamp

## Eloquent Models

Models are defined with full relationships and support for:  
- Flow to model polymorphism  
- Step-to-condition/action mapping  
- Instance tracking and logging

## Services

Located in `App\Services\ProcessFlow`:  
- ProcessConditionService: Checks if a step can advance  
- ProcessActionService: Executes actions after step completion  
- ProcessEngineService: Coordinates the entire step advancement process

## Controller

ProcessFlowController includes:  
- `advance()` method that moves the current instance to the next step after checking conditions and executing actions.

## Route

POST /process/advance/{instance}  
- Protected by Spatie `permission` middleware (dynamic based on route name)  
- Uses route name: `process.advance`

## Manual Advancement Button Example

@can('process.advance')  
<form action="{{ route('process.advance', $instance->id) }}" method="POST">  
 @csrf  
 <button type="submit" class="btn btn-success">Complete Step</button>  
</form>  
@endcan

## Automatic Completion Logic

When all Activity flows under a PsipName are completed:  
- PsipName status\_id is updated to "completed"  
- Optional: PsipName's own process flow may also be marked as completed

## Power User Interface

Restricted to users with roles: `admin` and `planning`.  
  
Enables users to:  
1. Create/edit process flows  
2. Add/remove/reorder steps  
3. Assign conditions and actions  
4. Link flows to Eloquent models  
  
UI Components:  
- Blade form examples for creating flows  
- Step list with edit actions  
- Can be extended using Livewire or Vue.js

## Setup and Requirements

• Laravel 8.75+  
• PHP 7.3+  
• Spatie Laravel Permission (^5.5)  
• Directory structure for Services and Controllers  
• Middleware registered in Kernel for dynamic permission handling

## Next Steps

- Build full CRUD Blade UI for process flows, steps, and logic  
- Create reusable condition/action classes  
- Implement UI validation and live previews of process logic