Dynamic Workflow Automation Challenge

Problem Statement

1. Executive Summary

We are seeking a configurable, visual, and extensible Dynamic Workflow Orchestration Platform to automate policy lifecycle actions across multiple insurance systems.

Today, policy workflows (e.g., underwriting, rating, payments, e-signature, third-party risk evaluation) are tightly coupled, hard-coded, and difficult to modify without engineering effort. This results in slow time-to-market, brittle integrations, and limited experimentation.

The objective of this challenge is to build a ***low-code / no-code workflow engine*** that allows business and technical users to visually design, test, and execute policy workflows using configurable actions, conditions, and custom logic—similar in experience to ***Microsoft Power Automate***, but purpose-built for insurance policy systems.

Note: This is a black-box challenge. You must design the solution architecture, execution model, and UI framework independently. Only the problem description and expected capabilities are provided.

---

2. Background

In a modern insurance platform, a single policy transaction can trigger multiple dependent actions:

- Calling third-party services (eSign, Payment Gateways, Risk & Property Data Providers)
- Executing underwriting rules
- Invoking the rating engine
- Updating policy or transaction statuses
- Handling failures, retries, and conditional routing

Currently:

- These workflows are **hardcoded** in backend services.
- Changes require **developer intervention** and **redeployments**.

- There is no centralized way to visualize or test end-to-end flows.

As carriers, products, and regulatory requirements grow, workflow logic must become **configurable, dynamic, and observable.**

---

3. The Problem

Manual and code-driven workflow orchestration has become a bottleneck.

Key Pain Points

- Each new carrier or product requires **custom code paths**
- No visual representation of policy flows
- Business teams cannot safely modify flows
- Difficult to test alternate paths (e.g., failures, underwriting referrals)
- No reusable action catalogue across products

We need a **dynamic workflow system** that behaves like a human-designed flowchart but executes like a deterministic backend engine.

---

4. Technical Requirements

4.1 Workflow Definition & Actions

The system **must support configurable actions**, including but not limited to:

- Calling Third-Party APIs
  - eSignature providers
  - Payment gateways
  - Risk evaluation services (Hazard Hub, E2Value, etc.)
- Updating Policy or Transaction Status
- Executing Underwriting Rules
- Invoking the Rater Engine
- Emitting Events or Notifications

Each action must:

- Declare its **expected inputs**

- Produce a **standardized output**
- Be reusable across workflows

4.2 Conditional Routing & Decisioning

The workflow engine must dynamically determine the **next action** based on conditions such as:

- Policy Status
- Rater Success / Failure
- Underwriting Rule Outcomes
- Third-Party API Responses
- Custom expressions or flags

The system should support:

- Conditional branches (IF / ELSE)
- Multiple exit paths
- Failure handling and alternate routing

4.3 Visual Workflow Builder (UI Sandbox)

The solution must include a UI-based workflow designer, like Microsoft Power Automate.

Required Capabilities

- **Drag & Drop Actions**
    - From a predefined action catalogue
- **Draggable Connections**
    - Visual arrows to connect workflow steps
- **Conditional Nodes**
    - Insertable between actions
    - Supports rule-based expressions
- **Custom Action Builder**
    - JavaScript editor to define new actions

- Custom actions become reusable like prebuilt ones
- **Prebuilt Action Library**
  - Based on core POS / policy components
  - Strongly typed inputs and outputs
- **Built-in Testing Sandbox**
  - Execute workflows with sample policy data
  - Inspect step-by-step execution results
- **Export Capabilities**
  - Download workflow diagrams as PDF or Image
  - Export workflow definition for version control

4.4 Execution Engine

The workflow engine must:

- Interpret the visual workflow into an executable format
- Execute steps deterministically
- Handle synchronous and asynchronous actions
- Support retries, failures, and timeouts
- Be scalable across simple and complex workflows

---

5. Expected Outputs (Deliverables)

The solution must produce:

1. **Workflow Definition Artifact**
   - JSON / YAML / XML representing the workflow graph
   - Includes actions, conditions, and connections
2. **Executable Runtime Model**
   - A normalized format consumed by the workflow engine
3. **UI-Generated Flow Diagram**
   - Exportable as PDF/Image

4. **Execution Results**
    - Step-by-step execution logs
    - Inputs, outputs, and decision paths

---

6. Key Challenges to Solve

6.1 Dynamic Action Modelling

- Standardizing diverse actions with different inputs and outputs
- Ensuring extensibility without breaking existing workflows

6.2 Condition Expression Engine

- Supporting complex conditions without exposing unsafe code
- Balancing flexibility and security

6.3 Visual-to-Executable Translation

- Converting a UI flowchart into a deterministic runtime model
- Maintaining execution order and state

6.4 Scalability & Maintainability

- Supporting simple linear flows and complex branching workflows
- Reusability across products, carriers, and policy transactions

---

7. What Is NOT Provided

- No existing workflow engine or orchestration framework
- No predefined UI components
- No predefined execution model
- No backend services for third-party integrations (Use dummy for challenge)

All architectural, technical, and design decisions are part of the challenge.

8. Success Criteria

A successful solution will:

- Allow workflows to be built **without code changes**
- Be intuitive for both technical and semi-technical users
- Execute reliably at scale
- Reduce onboarding and change effort from weeks to days
- Provide visibility, testing, and governance over policy workflows

---

9. Next Steps

Sample workflow scenarios and expected execution outcomes will be provided for validation.
Your goal is to build the **"Black Box"** workflow platform that connects visual design, execution logic, and real-world insurance operations.