Design Document: Generic Execution Engine and Modeler

🎯 Purpose

The **Generic Execution Engine and Modeler** provides a powerful, extensible platform for automating and orchestrating workflows across various domains. By utilizing a **visual process modeler** (e.g., BPMN) and **token-based execution engine**, it offers the ability to:

- Model and visualize business processes or automated workflows.
- Manage the execution of tasks, data pipelines, and complex workflows in real-time.
- Track execution state through **immutable tokens** for auditability and historical analysis.
- Support multiple execution environments, including CLI, web, services, or microservices.

💡 Problem

Most process automation tools:

- Are **domain-specific**, limiting their use to one area (e.g., software build pipelines, business workflows).
- Lack **real-time visualization** or provide **limited integration** with existing systems.
- Provide **insufficient audit trails** and **execution state tracking**.
- Are difficult to extend or adapt to different use cases and require substantial custom work.

✅ Solution

This **generic execution engine** overcomes these limitations by providing:

Layer	Capability
🔹 **Process Modeler**	Visual workflow design (supports BPMN, other graphical models)
🔹 **Execution Engine**	Run, monitor, and manage the flow of tasks across systems (CLI, web, services)
🔹 **Immutable Token Tracking**	Track the execution history of each task and state with tokens
🔹 **Real-time Execution Monitoring**	Visualize live process status, progress, and bottlenecks
🔹 **Flexible Integration**	Support for both synchronous/asynchronous operations (CLI, API, microservices)
🔹 **Extensibility**	Custom workflows, user-defined tasks, and integrations with external systems

---

⚙️ Key Components

1. Process Modeler

- **Visual Interface**: Drag-and-drop editor for designing workflows (using BPMN or other flow models).
- **Tasks & Gateways**: Represent tasks, services, events, and conditional logic (exclusive, parallel, etc.).
- **Sub-Processes**: Enable recursive or modular task grouping.
- **Error Handling**: Define boundary events for error, timeout, or external triggers.

2. Execution Engine

- **Token-based Execution**: Tokens represent the state and progression of workflows.

- **Node Execution Logic**: Tasks (service, user, script) are executed asynchronously or synchronously, based on the task type.
- **Flow Evaluation**: For each task or gateway, evaluate conditions (e.g., `XOR`, `OR`, `AND`).
- **Parallel Execution**: Handle parallel branches (forked tasks) and merging of results using **join** mechanisms.

3. Immutable Token Tracking

- **Execution Tokens**: Immutable representations of each step, with historical states.
- **Auditability**: Full traceability from start to end with metadata (owner, timestamp, current state).
- **History Chains**: Track transitions of tokens through different tasks, keeping a chain of execution states.

4. Real-time Monitoring

- **WebSocket or Redis Pub/Sub**: Real-time broadcasting of execution state to front-end UI or external services.
- **State Visualization**: Live tracking of tasks and tokens, including performance metrics.
- **Custom Dashboards**: Display key metrics like task completion time, user actions, success/error rates, etc.

🧠 Architecture Overview

High-Level Design

- **Process Modeler**: UI for designing and visualizing workflows. Stores flow diagrams in JSON or XML format.
- **Execution Engine**: Orchestrates task execution. Evaluates task conditions, tracks execution state, and triggers next actions.

- **Token Management**: Immutable token chain for audit trail. Tokens move through the graph, tracking progress and storing metadata.
- **Real-time Events**: Event-driven architecture via WebSocket or Redis for live UI updates.

🧩 Data Model

1. Node (Task)

- **Fields**: `id`, `type`, `name`, `status`, `startedAt`, `endedAt`, `properties`
- **Types**: `serviceTask`, `userTask`, `startEvent`, `endEvent`, `gateway`, etc.

2. Edge (Flow)

- **Fields**: `id`, `sourceNode`, `targetNode`, `condition`
- **Condition**: Optional expression evaluated at runtime (e.g., `approval == true`).

3. Execution Token

- **Fields**: `id`, `currentNode`, `status`, `owner`, `startedAt`, `history`, `previousToken`
- **Immutable**: The history of each token is never modified. It is appended with each new state change.

🔧 Execution Flow Example

1. Token Initialization

- A new token is created at the **StartEvent** and placed into the execution engine.

2. Node Execution

- The engine processes each task (e.g., running a CLI command, executing a service call, or awaiting user input).
- Tokens are passed along edges to the next task, based on the flow conditions.

3. Error Handling

- If a task fails, the token triggers a boundary event (error or timer) and reroutes the process.

4. Finalization

- Once all tokens reach the **EndEvent**, the process is marked as complete.
- The execution state is stored, and the token chain remains immutable for audit purposes.

📦 Features Summary

Feature	Description
🎨 **Visual Workflow Modeling**	BPMN (or custom) process design, supports sub-processes, and parallel/conditional paths
⚡ **High-Performance Execution**	Token-based parallel execution, supports async tasks, isolate management
🔐 **Immutable Token History**	Full traceability of each execution step, ensuring auditability and compliance
🌍 **Real-time Execution Monitoring**	Live feedback via WebSocket/Redis Pub/Sub to UI/dashboard
🔌 **Customizable and Extensible**	Easy integration with external systems via APIs, supports plugin architecture
📊 **Execution Analytics**	Metrics on task durations, bottlenecks, failure rates, and process health

---

📈 Roadmap for Full Execution Engine and Modeler

Phase 1: Core Engine Development

- **CLI support** for external commands execution
- **Token-based execution** for processes
- **Real-time state tracking** with immutable tokens

Phase 2: Visual Process Modeler

- **BPMN editor** for drag-and-drop workflow design
- **Save/load workflows** in XML/JSON format
- **Execution engine integration** for modeling and running workflows

Phase 3: Real-Time Monitoring and Events

- **WebSocket or Redis integration** for live updates
- **Custom dashboards** for process status and analytics
- **Visual feedback** on task execution and token progression

Phase 4: Extensibility and Integration

- **Plugin architecture** for user-defined tasks
- **Support for additional workflow types** (e.g., data pipelines, AI workflows)
- **Extensive API support** for integrating with third-party services

Phase 5: Scalability and CI/CD Integration

- **Cloud-native support** (e.g., Kubernetes, microservices)
- **CI/CD pipeline integration** (e.g., with GitHub, Jenkins)

- **Advanced error handling**, retry logic, and fault tolerance

---

🧩 Use Cases

1. **Software Development**: Automate build pipelines, pre-commit hooks, and continuous integration workflows.
2. **Business Process Automation**: Model and execute HR, finance, or customer service workflows.
3. **Data Pipelines**: Orchestrate ETL (extract, transform, load) processes for analytics platforms.
4. **AI/ML Workflows**: Automate model training, hyperparameter tuning, and model deployment.

🚀 Next Steps

1. **Finalize Engine Core**: Token lifecycle, node execution, and real-time monitoring.
2. **Develop Process Modeler UI**: Implement BPMN support and basic workflow visualization.
3. **Integrate Real-time Monitoring**: WebSocket/Redis event handling.
4. **Test Extensibility**: Ensure the system can be extended for new task types or integrations.
