Deployment Approval Workflow System Documentation

1. Project Overview

The Deployment Approval Workflow System is a backend-driven system that manages deployment requests through a formal multi-step approval process. It ensures compliance, traceability, and deterministic state transitions before a deployment is executed.

Key Objectives:

- Manage deployment requests.
- Require approvals in stages: QA → DevOps.
- Track deployment states with a state machine.
- Provide REST APIs and a simple frontend for operators.
- Ensure frontend reflects real-time state transitions.

2.Features

- Create deployment requests (`POST /deployments`).
- Approve or reject deployments at any stage (`POST /deployments/{id}/approve` / `reject`).
- Enforce multi-stage approvals (QA → DevOps → Executed).
- Deterministic state transitions using transitions state machine library.
- Real-time frontend updates (polling).
- Color-coded deployment states for visual clarity.
- Persistent storage using SQLite.
- Full API documentation via FastAPI Swagger (`/docs`).

3. Architecture & Data Flow

3.1 Data Flow:

1.Developer submits a deployment request from the frontend.
2.Backend initializes a state machine in the requested state.
3.Deployment appears in the frontend list.
4.Approvals move the deployment through stages:

- `requested → approved_by_QA → approved_by_devops → execute`

5.Rejection can occur at any stage.
6.Frontend reflects updated states and disables buttons if executed or rejected

3.2 Architecture Diagram

```
 +----------------+
 |   Frontend     |
 | HTML + JS + CSS|
 |  Actions:      |
 |  - Create      |
 |  - Approve     |
 |  - Reject      |
 +--------+-------+
          |
          | HTTP Requests (REST API)
          v
 +----------------+
 |   Backend      |
 |   FastAPI      |
 |   State Machine|
 |   Transitions  |
 +--------+-------+
          |
          v
 +----------------+
 |   Database     |
 |   SQLite       |
 |   SQLAlchemy   |
 +----------------+
```

4.Design Decisions and Trade-offs

Component	Choice	Reasoning / Trade-offs
Backend Framework	FastAPI	Fast development, async support, Swagger docs. Trade-off: Less mature than Django for large-scale apps.
State Management	`transitions` library	Deterministic, formal state transitions. Trade-off: Adds complexity but ensures correctness.
Database	SQLite + SQLAlchemy	Lightweight and simple setup. Trade-off: Not suitable for heavy production workloads.
Frontend	HTML + JS + CSS	Simple and lightweight. Trade-off: Uses polling instead of WebSockets for real-time updates.
Action Buttons	Disabled on executed/rejected	Prevents invalid actions. Trade-off: Logic duplicated in backend, but improves UX.
Color-coded States	Frontend visual feedback	Enhances clarity for operators. Trade-off: Hard-coded in JS, less scalable for large apps.

5. API Endpoints

Method	Endpoint	Description
POST	/deployments	Create a new deployment request
POST	/deployments/{id}/approve	Approve a deployment
POST	/deployments/{id}/reject	Reject a deployment
GET	/deployments	List all deployments & status

6. Example Workflow

1.Submit Deployment

POST /deployments
{ "name": "Deploy Feature X" }
Response: { "id": 1, "name": "Deploy Feature X", "state": "requested" }

2.Approve by QA

POST /deployments/1/approve
Response: { "id": 1, "state": "approved_by_QA" }

3.Approve by DevOps

POST /deployments/1/approve
Response: { "id": 1, "state": "approved_by_devops" }

4.Executed

POST /deployments/1/approve
Response: { "id": 1, "state": "executed" }

5.Reject at any stage

POST /deployments/1/reject
Response: { "id": 1, "state": "rejected" }

7. Enhancements / Future Work

1. Add websocket support for real-time frontend updates.
2. Integrate authentication & role-based access for QA and DevOps.
3. Switch from SQLite to PostgreSQL for production.
4. Add audit logs for all state transitions.
5. Enhance frontend using React or Vue for more dynamic UI.