Deployment Approval Workflow System - Interview 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.

---

2. Features

- Create deployment requests (POST /deployments).
- Approve or reject deployments at any stage (POST /deployments/{id}/approve / reject).
- Enforce multi-stage approvals (requested → approved_by_QA → approved_by_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 + SQLAlchemy.
- Full API documentation via FastAPI Swagger (/docs).

---

3. Architecture & Data Flow

Architecture Diagram:

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

Data Flow: 1. User submits a deployment request from the frontend. 2. Backend initializes a state machine in the `requested` state. 3. Deployment appears in the frontend list (polling ensures immediate update). 4. Approvals move the deployment through stages: requested → approved_by_QA → approved_by_devops → executed. 5. Rejection can occur at any stage (rejected). 6. Frontend reflects updated states and disables buttons if executed or rejected.

---

4. Deployment Lifecycle

Stage	Description
requested	Deployment request submitted, pending QA approval
approved_by_QA	Approved by QA, pending DevOps approval
approved_by_devops	Approved by DevOps, ready for execution
executed	Deployment executed successfully
rejected	Deployment rejected at any stage

---

5. Backend API Endpoints

Method	Endpoint	Description
POST	/deployments	Create deployment request
POST	/deployments/{id}/approve	Approve deployment stage
POST	/deployments/{id}/reject	Reject deployment
GET	/deployments	List all deployments and states

---

6. Technology Stack

Layer	Technology
Backend	Python, FastAPI
State Mgmt	transitions (state machine)
Database	SQLite + SQLAlchemy
Frontend	HTML, CSS, Vanilla JavaScript
API Docs	Swagger (/docs)

---

7. 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.
Deployment Flow	requested → approved_by_QA → approved_by_devops → executed	Ensures compliance and traceability. Trade-off: Cannot skip stages.

---

8. Setup Instructions

1. Clone the repository

```
git clone git@github.com:estiba-27/Deployment_flow.git
cd Deployment_flow
```

2. Backend Setup

```
cd backend
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

3. Initialize Database

```
python create_db.py
```

4. Run Backend Server

```
uvicorn main:app --reload
```

- API URL: http://127.0.0.1:8000 - Swagger Docs: http://127.0.0.1:8000/docs

5. Frontend Setup

```
cd frontend
python3 -m http.server 8080
```

- Open in browser: http://localhost:8080

---

9. Demo / Testing

- Submit deployment via frontend form → appears in the list immediately.
- Click Approve → moves to next state.
- Click Reject → moves to rejected.
- Buttons auto-disable when executed or rejected.
- Deployment states are color-coded.

---

10. Notes for Interview Discussion

- Emphasize state machine design and deterministic transitions.
- Explain trade-offs between simplicity and scalability.
- Discuss frontend polling vs WebSockets for live updates.
- Highlight multi-step approval workflow and real-time feedback.