ChatGPT

Epic: Replace Legacy Approval Workflow Engine

Objective

Migrate from Windows Workflow Foundation (WF) to a **home-grown, maintainable, and scalable** workflow engine that integrates cleanly with existing APIs, databases, and UIs while preserving business logic and approval processes.

Acceptance Criteria

• **Functional Parity**: Must support existing operations:
• Submission
• Approval/Rejection
• Withdrawal
• Approver Changes

• Heartbeat/Status

• **State Management**:

• Preserve auditable transitions: Pending, Approved, Rejected, Withdrawn, Completed, Error

• **API Compatibility**:

• Maintain existing API endpoints with minimal or no change

• **Database Integration**:

• Persist state/history in the existing schema or clearly documented schema

• **Notifications**:

• Maintain existing email/SMS/etc. notifications for approvers, submitters, admins

• **Monitoring**:

• Health check and workflow visibility via scheduled job/endpoint

• **Extensibility**:

• Easily add/modify workflow states and steps

- **Authorization**:

- Role-based access control to restrict actions (e.g., submitters, approvers, admins)

- **Documentation**:

- Provide state diagrams, API contracts, migration steps

- **Testing**:

- Unit/integration tests for all workflow scenarios and edge cases
- Include contract tests and rollback/recovery scenarios

Design Details

1. **Workflow Engine Implementation**
2. Build a custom home-grown workflow engine from scratch
3. Written in Node.js using the most modern and stable LTS version
4. Follow industry best practices for structure, modularity, and error handling

5. Stateless architecture: workflow state changes are performed via consumer-facing APIs

6. **Workflow Definition**

7. Define states/transitions explicitly using code (JSON or configuration-driven)
8. Implement deterministic state machine logic for all paths

9. Support metadata (e.g., rejection reasons, comments) per transition

10. **API Layer**

11. Expose RESTful endpoints:
    - `/workflow/submit`
    - `/workflow/respond`
    - `/workflow/withdraw`
    - `/workflow/status/:id`
    - `/workflow/heartbeat`
12. Validate payloads and enforce preconditions through middleware

13. **Persistence**

14. Store workflow instance, current state, transition history, and metadata
15. Ensure atomicity and auditability

16. Track `workflowDefinitionVersion` to support future migrations

17. **Notifications**

18. Trigger notifications (email/SMS) on state transitions

19. Log delivery outcomes for traceability

20. **Monitoring**

21. Scheduled job or REST endpoint to report health/status

22. Log errors, failed transitions, and exception traces

23. **Migration**

24. Allow in-flight workflows to complete in WF or migrate

25. Avoid data loss and minimize service interruption

26. **Documentation & Training**

27. Document engine architecture, state models, API contracts, and procedures

28. Train developers and support staff on usage

29. **Operational Metrics**

30. Total workflows by status
31. Time spent in each state
32. Failed transitions

33. Notification delivery success/failure

34. **Pre-Approved Paths (Optional)**

35. Define auto-transitions based on business rules (e.g., low-risk auto-approval)

Spikes

- **Spike 1**: Define architecture and state engine interfaces
- Document lifecycle, extensibility strategy, and transition validation model
- **Spike 2**: Prototype stateless Node.js state machine
- Implement transition APIs that write to the persistence layer
- Persistence to MongoDB or PostgreSQL

- **Spike 3**: API compatibility validation

- Mock legacy endpoints and test against new engine behavior

User Stories

- **Submitter**: Can submit an assessment → goes to “Pending”
- **Approver**: Can approve/reject → changes state and notifies
- **Submitter**: Can withdraw → moves to “Withdrawn”
- **Admin**: Can monitor status → via health/status endpoint
- **Developer**: Can extend workflow → modular, testable logic

---

Design Document

Architecture Overview

- **Frontend**: Angular (unchanged)
- **Backend**: Node.js/Express REST API (stateless)
- **Workflow Engine**: Custom-built, stateless Node.js service
- **Database**: MongoDB or PostgreSQL
- **Notifications**: NodeMailer or equivalent

Workflow Model

State	Trigger/Event	Next State
Pending	Approver approves	Approved
Pending	Approver rejects	Rejected
Pending	Submitter withdraws	Withdrawn
Approved	-	Completed
Rejected	-	Completed
Withdrawn	-	Completed
Any	Error	Error

API Endpoints

Endpoint	Method	Description
`/workflow/submit`	POST	Submit new assessment

Endpoint	Method	Description
`/workflow/respond`	POST	Approver responds
`/workflow/withdraw`	POST	Withdraw assessment
`/workflow/status/:id`	GET	Get status/history
`/workflow/heartbeat`	GET	Monitor workflow health

🗃️ MongoDB Schema Example

```
{
  "assessmentId": "string",
  "currentState": "string",
  "history": [
    {
      "state": "string",
      "timestamp": "date",
      "triggeredBy": "string",
      "metadata": {
        "reason": "string",
        "comments": "string"
      }
    }
  ],
  "workflowDefinitionVersion": "string",
  "createdAt": "date",
  "updatedAt": "date"
}
```

Key Design Decisions

- Stateless architecture; all state changes are driven by API calls
- Use a code-based, deterministic state machine
- Persist state and history for traceability
- Maintain legacy-compatible API contract
- Decouple notifications to improve maintainability
- Design engine for modularity and extension
- Include versioning support and validation hooks

Implementation Phases

Phase 1: Foundation & Planning

- Finalize architecture and tech stack
- Complete Spike 1 (architecture/interfaces)
- Establish version control, CI/CD pipelines, coding standards
- Define initial workflows and transitions
- Draft API contracts

Phase 2: Core Engine & Persistence

- Build core stateless engine logic
- Implement state transition logic
- Complete Spike 2 (Node.js prototype with persistence)
- Set up MongoDB/PostgreSQL schema
- Implement transition metadata handling
- Unit test basic transitions

Phase 3: API Layer & Integration

- Implement full API layer for transitions and status
- Complete Spike 3 (API compatibility)
- Add middleware for validation and RBAC
- Begin front-end API integration and adapter layer
- Support legacy API behavior where required

Phase 4: Notifications & Monitoring

- Add notification service (email/SMS)
- Implement monitoring endpoints and logs
- Add operational metrics: throughput, failure rates, etc.

Phase 5: Migration Path

- Implement dual-write or shadow mode
- Route new submissions to new engine
- Monitor in-flight workflows and plan legacy shutdown
- Document versioning/migration support

Phase 6: Harden & Extend

- Finalize automated tests (integration, rollback, contract)
- Train devs/support; prepare rollout plan
- Add support for optional auto-approvals
- Optimize performance and error handling

Copilot Enablement & Prompting Guidelines

To improve consistency and development velocity using GitHub Copilot:

Setup Requirements

- Use latest Node.js LTS version
- Include ESLint/Prettier configs
- JSDoc for all major functions and modules

Prompting Practices

```
// Transition a workflow from Pending to Approved
async function approveWorkflow(workflowId: string, approverId: string) { ... }

// Log a transition with metadata to MongoDB
```

Bootstrapping Suggestions

Add a header in main engine files:

```
/**
 * Workflow Engine
 * Stateless engine for handling assessment transitions:
 * States: Pending, Approved, Rejected, Withdrawn, Completed, Error
 * API-driven transition logic with audit trail and metadata support.
 */
```

Documentation & Onboarding Notes

- Add standard imports in README
- Document naming conventions for services and transitions
- Include prompt templates for Copilot to follow during development

---

🛠 Migration Plan

1. Deploy new engine in parallel with legacy WF
2. Route new submissions to custom engine
3. Allow legacy workflows to complete; decommission WF gradually

---

⚠ Risks & Mitigations

Risk	Mitigation
Data loss	Dual-write strategy or audit-logged cutover
API incompatibility	Contract testing and backward compatibility
Missed business logic	Comprehensive test coverage and stakeholder review