# Base Prompt

Help me in creating a detailed prompt to be fed into Claude, which can help me in creating a workflow of various agents as required for n8n. My objective is to convert old legacy AngularJS applications to latest versions of React.

I have few ideas which you can use to help me in writing the prompt

1. There should be an agent which will take the existing AngularJS code base.
2. another agent will analyse the codebase and create the necessary dependency graphs, user stories, styles and other that will be required in the next step.
3. third agent will take in the generated output from 2nd step and devise a plan for migration and ask humans for code best practices and specific framework libraries to opt for state management, styling or other.
4. 4th agent will follow the migration plan or strategy and will refer the dependency graph, user story, styles generated and use it to create the project in react out of it.
5. 5th agent will review the output of the 4th agent and give feedbacks from the existing angularJS code base whether new react project has all the features or styles or not., if not it ll give the feedback to 4th agent to upgrade the react base else it ll move to step 6.
6. the 6th agent will run the code and ask the human to verify and if any feedback will be there it ll give the feedback to 4th agent to update accordingly.
7. once the code and functionalities is verified by human, it ll be pushed to git by 7th agent.
8. 8th agent will be monitoring each activities in the entire Flow and

Note: use vector db for angularjs code based and other purpose for making use of chunking to fetch relevant data set and clears once the code is pushed to git use postgresql to store relevant logs, monitoring activities by 8th agent and which can be helpful to make the process better int he future.

analyse and add other things as applicable

# Claude **AngularJS to React Migration Multi-Agent Workflow System**

**System Overview**

Create a comprehensive n8n workflow system with 8 specialized agents to automate the migration of legacy AngularJS applications to modern React applications. The system should leverage vector databases for code analysis, PostgreSQL for logging and monitoring, and include human-in-the-loop validation at critical stages.

**Infrastructure Requirements**

* **Vector Database**: Store and chunk AngularJS codebase for efficient retrieval and analysis
* **PostgreSQL Database**: Log all activities, store migration metadata, and maintain audit trails
* **Git Integration**: Automated repository management and deployment
* **n8n Workflow Orchestration**: Coordinate agent interactions and data flow

**Agent Specifications**

**Agent 1: Codebase Ingestion Agent**

**Purpose**: Intake and prepare AngularJS codebase for analysis

**Responsibilities**:

* Accept AngularJS project files (ZIP, Git repository, or directory structure)
* Validate project structure and identify AngularJS version
* Extract and catalog all source files (JS, HTML, CSS, JSON, config files)
* Identify external dependencies from package.json, bower.json, or similar
* Store codebase in vector database with appropriate chunking strategy
* Generate initial project metadata (file count, size, complexity indicators)

**Input**: AngularJS project files or repository URL **Output**: Structured codebase data, project metadata, vector DB storage confirmation

**Agent 2: Analysis and Documentation Agent**

**Purpose**: Comprehensive codebase analysis and documentation generation

**Responsibilities**:

* **Dependency Analysis**:
  + Map all internal and external dependencies
  + Identify third-party libraries and their React equivalents
  + Create dependency graph visualization
  + Flag deprecated or incompatible dependencies
* **Architecture Analysis**:
  + Identify application structure (modules, controllers, services, directives)
  + Map routing configuration and navigation patterns
  + Analyze data flow and state management patterns
  + Document API integrations and HTTP services
* **User Story Generation**:
  + Extract functional requirements from existing code
  + Generate user stories based on UI components and user interactions
  + Document business logic and validation rules
  + Create feature inventory with priority classification
* **Style Analysis**:
  + Extract CSS/SCSS styles and themes
  + Identify UI component patterns and design system elements
  + Document responsive breakpoints and media queries
  + Catalog custom directives and their styling implications
* **Code Quality Assessment**:
  + Identify code smells and technical debt
  + Assess test coverage and testing patterns
  + Document performance bottlenecks
  + Generate complexity metrics

**Input**: Vector DB codebase chunks **Output**: Dependency graphs, user stories, style guide, architecture documentation, quality assessment report

**Agent 3: Migration Strategy and Planning Agent**

**Purpose**: Create detailed migration plan with human consultation

**Responsibilities**:

* **Migration Strategy Development**:
  + Analyze complexity and recommend migration approach (big bang vs. incremental)
  + Prioritize features and components for migration order
  + Identify reusable components and shared utilities
  + Plan for data migration and API compatibility
* **Technology Stack Recommendation**:
  + Suggest React version and related ecosystem choices
  + Recommend state management solutions (Redux, Zustand, Context API)
  + Propose styling solutions (Styled Components, Emotion, Tailwind CSS)
  + Suggest testing frameworks and development tools
* **Human Consultation Interface**:
  + Present multiple technology options with pros/cons
  + Request feedback on coding standards and conventions
  + Validate business logic interpretation
  + Confirm migration priorities and timeline constraints
* **Risk Assessment**:
  + Identify potential migration challenges
  + Document breaking changes and compatibility issues
  + Plan rollback strategies
  + Estimate effort and timeline

**Input**: Analysis report from Agent 2, human preferences and requirements **Output**: Migration strategy document, technology stack recommendations, risk assessment, implementation roadmap

**Agent 4: React Development Agent**

**Purpose**: Execute migration plan and generate React application

**Responsibilities**:

* **Project Setup**:
  + Initialize React project with chosen technology stack
  + Configure build tools, linting, and development environment
  + Set up folder structure following React best practices
  + Install and configure selected dependencies
* **Component Migration**:
  + Convert AngularJS controllers to React functional components
  + Transform AngularJS directives to React components
  + Implement React hooks for lifecycle management
  + Convert AngularJS services to React custom hooks or utilities
* **State Management Implementation**:
  + Implement chosen state management solution
  + Migrate AngularJS scope variables to React state
  + Convert two-way data binding to React patterns
  + Implement global state for shared data
* **Routing and Navigation**:
  + Implement React Router based on AngularJS routing
  + Convert route guards to React route protection
  + Migrate URL parameters and navigation logic
* **Styling Migration**:
  + Convert CSS/SCSS to chosen styling solution
  + Implement responsive design patterns
  + Maintain visual consistency with original design
* **API Integration**:
  + Convert AngularJS HTTP services to modern fetch/axios patterns
  + Implement error handling and loading states
  + Maintain API contract compatibility

**Input**: Migration plan, dependency graph, user stories, style guide **Output**: Complete React application, component library, documentation

**Agent 5: Quality Assurance and Validation Agent**

**Purpose**: Verify migration completeness and quality

**Responsibilities**:

* **Feature Completeness Validation**:
  + Compare React application features against original user stories
  + Verify all UI components and interactions
  + Validate business logic implementation
  + Check data flow and state management accuracy
* **Visual Consistency Check**:
  + Compare styling and visual elements
  + Verify responsive behavior across devices
  + Validate theme and branding consistency
  + Check accessibility compliance
* **Performance Analysis**:
  + Compare load times and runtime performance
  + Analyze bundle size and optimization opportunities
  + Verify mobile performance and responsiveness
* **Automated Testing**:
  + Run unit tests and integration tests
  + Perform accessibility audits
  + Execute cross-browser compatibility checks
  + Validate API integration functionality
* **Feedback Generation**:
  + Generate detailed comparison reports
  + Identify missing features or discrepancies
  + Provide specific feedback for Agent 4 improvements
  + Document any breaking changes or differences

**Input**: React application from Agent 4, original AngularJS codebase reference **Output**: Quality assessment report, feature gap analysis, improvement recommendations

**Agent 6: Human Validation and Testing Agent**

**Purpose**: Facilitate human testing and feedback collection

**Responsibilities**:

* **Test Environment Setup**:
  + Deploy React application to staging environment
  + Provide testing instructions and scenarios
  + Set up feedback collection mechanisms
* **Human Interaction Management**:
  + Present application for human testing
  + Collect and categorize user feedback
  + Document bug reports and feature requests
  + Manage approval/rejection workflow
* **Feedback Processing**:
  + Analyze human feedback for patterns
  + Prioritize issues based on severity and impact
  + Generate actionable improvement tasks for Agent 4
  + Track feedback resolution progress
* **Sign-off Management**:
  + Manage approval workflow
  + Document final acceptance criteria
  + Generate testing reports and documentation

**Input**: Deployed React application, QA reports from Agent 5 **Output**: Human feedback compilation, approval status, improvement tasks

**Agent 7: Deployment and Git Management Agent**

**Purpose**: Handle code deployment and version control

**Responsibilities**:

* **Git Repository Management**:
  + Create new React project repository
  + Set up branching strategy and commit conventions
  + Migrate relevant commit history if needed
  + Configure repository settings and access controls
* **Code Quality Enforcement**:
  + Set up pre-commit hooks and code formatting
  + Configure continuous integration pipelines
  + Implement automated testing in CI/CD
  + Set up code quality gates
* **Deployment Pipeline**:
  + Configure build and deployment processes
  + Set up staging and production environments
  + Implement rollback mechanisms
  + Configure monitoring and alerting
* **Documentation Generation**:
  + Generate README and setup instructions
  + Document deployment procedures
  + Create developer onboarding guide
  + Archive migration process documentation

**Input**: Approved React application, deployment requirements **Output**: Deployed application, configured repository, deployment documentation

**Agent 8: Monitoring and Orchestration Agent**

**Purpose**: Oversee entire workflow and continuous improvement

**Responsibilities**:

* **Workflow Orchestration**:
  + Coordinate agent interactions and data flow
  + Manage workflow state and transitions
  + Handle error recovery and retry logic
  + Ensure data consistency across agents
* **Activity Monitoring**:
  + Log all agent activities to PostgreSQL
  + Track workflow progress and performance metrics
  + Monitor resource utilization and bottlenecks
  + Generate real-time status dashboards
* **Analytics and Reporting**:
  + Analyze migration patterns and success rates
  + Generate performance reports and insights
  + Identify workflow optimization opportunities
  + Track ROI and migration benefits
* **Continuous Improvement**:
  + Learn from completed migrations
  + Suggest workflow enhancements
  + Update agent configurations based on learnings
  + Maintain knowledge base of best practices
* **Error Handling and Recovery**:
  + Detect and handle agent failures
  + Implement automatic retry mechanisms
  + Escalate critical issues to human operators
  + Maintain audit trail of all interventions

**Input**: All agent activities and outputs **Output**: Monitoring dashboards, analytics reports, workflow optimizations, audit logs

**Data Management Strategy**

**Vector Database Implementation**

* **Chunking Strategy**: Break code files into logical chunks (functions, classes, modules)
* **Embedding Model**: Use code-specific embeddings for better semantic search
* **Retrieval**: Implement similarity search for finding related code patterns
* **Cleanup**: Automatically purge vector data after successful migration

**PostgreSQL Schema Design**

sql

*-- Migration sessions*

CREATE TABLE migration\_sessions (

id UUID PRIMARY KEY,

project\_name VARCHAR(255),

start\_time TIMESTAMP,

end\_time TIMESTAMP,

status VARCHAR(50),

metadata JSONB

);

*-- Agent activities*

CREATE TABLE agent\_activities (

id UUID PRIMARY KEY,

session\_id UUID REFERENCES migration\_sessions(id),

agent\_name VARCHAR(100),

activity\_type VARCHAR(100),

start\_time TIMESTAMP,

end\_time TIMESTAMP,

status VARCHAR(50),

input\_data JSONB,

output\_data JSONB,

error\_message TEXT

);

*-- Human feedback*

CREATE TABLE human\_feedback (

id UUID PRIMARY KEY,

session\_id UUID REFERENCES migration\_sessions(id),

feedback\_type VARCHAR(100),

content TEXT,

severity VARCHAR(20),

resolved BOOLEAN DEFAULT FALSE,

created\_at TIMESTAMP

);

*-- Performance metrics*

CREATE TABLE performance\_metrics (

id UUID PRIMARY KEY,

session\_id UUID REFERENCES migration\_sessions(id),

metric\_name VARCHAR(100),

metric\_value NUMERIC,

recorded\_at TIMESTAMP

);

**Workflow Configuration Guidelines**

**n8n Workflow Structure**

1. **Trigger Node**: HTTP webhook or manual trigger for starting migration
2. **Agent Nodes**: Custom n8n nodes for each agent with appropriate connectors
3. **Decision Nodes**: Logic gates for workflow branching and human approval
4. **Database Nodes**: PostgreSQL and Vector DB integration nodes
5. **Git Nodes**: Repository management and deployment nodes
6. **Notification Nodes**: Email/Slack notifications for human intervention

**Error Handling Strategy**

* Implement comprehensive error catching at each agent level
* Design rollback mechanisms for partial failures
* Create notification systems for human intervention requirements
* Maintain detailed error logs for debugging and improvement

**Performance Optimization**

* Implement parallel processing where possible
* Use caching for frequently accessed data
* Optimize vector database queries
* Monitor and alert on performance degradation

**Success Metrics and KPIs**

* Migration success rate and completion time
* Code quality improvements (technical debt reduction)
* Performance improvements (load time, bundle size)
* Human feedback scores and satisfaction
* Post-migration bug rates and maintenance overhead

**Security Considerations**

* Secure handling of source code and sensitive data
* Access control for human validation interfaces
* Audit logging for compliance requirements
* Secure Git repository and deployment pipelines

This multi-agent system provides a comprehensive, automated approach to AngularJS-to-React migration while maintaining quality, human oversight, and continuous improvement capabilities.

# Perplexity

**Comprehensive N8N Workflow Creation Prompt for AngularJS to React Migration System**

**Overview and System Architecture**

**Create a sophisticated multi-agent n8n workflow system designed to automate the complete migration of legacy AngularJS applications to modern React applications****[[1]](#fn1)****[[2]](#fn2). This system leverages an 8-agent architecture with PostgreSQL logging, vector database integration, and human-in-the-loop validation points to ensure high-quality, controlled migrations****[[3]](#fn3)****[[4]](#fn4).**

**Core System Requirements**

**Infrastructure Components**

* **PostgreSQL Database**: For comprehensive logging of migration sessions, agent activities, human feedback, and performance analytics[[3]](#fn3)
* **Vector Database**: For code storage, chunking, and semantic retrieval using embeddings[[5]](#fn5)[[6]](#fn6)
* **Git Integration**: Automated repository management and deployment workflows[[7]](#fn7)[[8]](#fn8)
* **MCP Communication**: Agent-to-agent messaging using MCP Trigger and Client nodes for seamless workflow orchestration[[2]](#fn2)

**Agent Communication Architecture**

**Implement a supervisor-agent pattern where:**

* MCP Trigger nodes act as agent listeners for inter-workflow communication[[2]](#fn2)
* MCP Client nodes enable cross-workflow messaging and coordination[[2]](#fn2)
* Sequential execution with conditional branching based on agent outputs
* Comprehensive error handling with retry logic and human intervention points[[4]](#fn4)[[9]](#fn9)

**Detailed Agent Specifications**

**Agent 1: Codebase Ingestion Agent**

**Create an n8n workflow that:**

**Triggers**: HTTP webhook or file upload trigger for AngularJS project submission

**Core Functions**:

* Validate AngularJS project structure and version compatibility using JavaScript code nodes
* Extract and catalog all source files (JS, HTML, CSS, JSON, configuration files)
* Implement recursive file system operations to handle complex project structures
* Generate project metadata including dependency mappings and file relationships

**Vector Database Integration**:

* Chunk code files into semantic segments for optimal storage[[5]](#fn5)
* Store code chunks in vector database with embeddings for later retrieval
* Create namespace organization for different project components
* Implement data cleanup mechanisms for completed migrations[[5]](#fn5)

**Database Operations**:

* Insert comprehensive project metadata into PostgreSQL migration\_sessions table[[3]](#fn3)
* Log ingestion metrics, file counts, and processing times
* Create unique session identifiers for tracking throughout the migration process

**Output**: Structured project data object and confirmation of vector database storage completion

**Agent 2: Analysis and Documentation Agent**

**Build a comprehensive analysis workflow featuring:**

**Triggers**: MCP Trigger activated by Agent 1 completion[[2]](#fn2)

**Analysis Engine**:

* Query vector database using similarity search for relevant code chunks[[5]](#fn5)
* Deploy multiple AI nodes (OpenAI/Anthropic) to analyze:
  + Dependency graphs and component relationships[[10]](#fn10)[[11]](#fn11)
  + Architecture patterns and design decisions[[12]](#fn12)[[13]](#fn13)
  + Code quality metrics and technical debt assessment[[14]](#fn14)[[15]](#fn15)
  + User story extraction from existing functionality[[16]](#fn16)[[17]](#fn17)

**Documentation Generation**:

* Create detailed user stories following "As a [role], I want [goal], so that [benefit]" format[[17]](#fn17)
* Generate dependency graphs showing component interdependencies[[10]](#fn10)[[11]](#fn11)
* Produce style guides and UI component inventories
* Document existing business logic and feature specifications

**Database Storage**: Store analysis results in PostgreSQL with JSONB fields for flexible metadata[[3]](#fn3)

**Human Review Integration**: Implement form node for stakeholder review and feedback collection[[4]](#fn4)[[18]](#fn18)

**Agent 3: Migration Strategy and Planning Agent**

**Design a strategic planning workflow with:**

**Input Processing**: Receive structured analysis data from Agent 2 via MCP messaging[[2]](#fn2)

**AI Strategy Generation**:

* Multiple AI agents recommending optimal technology stacks for React migration[[19]](#fn19)[[13]](#fn13)
* Analysis of migration approaches (incremental vs. complete rewrite)[[12]](#fn12)[[20]](#fn20)
* Risk assessment with timeline estimation and resource requirements[[19]](#fn19)

**Human Consultation Framework**:

* Interactive form sequences for technology preference gathering[[4]](#fn4)[[9]](#fn9)
* State management library selection (Redux, Zustand, Context API)[[13]](#fn13)
* Styling framework decisions (CSS-in-JS, Tailwind, styled-components)[[13]](#fn13)
* Build tool and deployment strategy choices[[13]](#fn13)

**Plan Documentation**:

* Generate comprehensive migration strategy documents
* Create implementation roadmaps with milestone definitions
* Produce risk mitigation strategies and rollback procedures

**Database Updates**: Store strategy decisions and human preferences in structured format[[3]](#fn3)

**Agent 4: React Development Agent**

**Create an advanced development workflow that:**

**Code Generation Engine**:

* AI-powered conversion of AngularJS components to React functional components[[12]](#fn12)[[20]](#fn20)
* Automatic translation of AngularJS directives to React custom hooks[[21]](#fn21)[[22]](#fn22)
* State management implementation using selected libraries[[13]](#fn13)
* Routing system conversion from ui-router to React Router[[13]](#fn13)

**Project Initialization**:

* Automated React project setup with chosen technology stack
* Configuration of build tools (Webpack, Vite, etc.)
* Installation and configuration of selected dependencies
* Setup of development environment and tooling

**Batch Processing Capabilities**:

* Handle multiple component conversions simultaneously
* Implement parallel processing for improved performance
* Maintain component relationship mappings during conversion

**Quality Assurance**:

* Automated code quality checks using ESLint and Prettier[[23]](#fn23)[[15]](#fn15)
* TypeScript integration and type safety validation
* Component prop validation and interface generation

**Git Integration**: Commit generated code with proper branching strategy and descriptive commit messages[[7]](#fn7)[[8]](#fn8)

**Progress Tracking**: Real-time updates to PostgreSQL with development status and metrics[[3]](#fn3)

**Agent 5: Quality Assurance Agent**

**Build a comprehensive QA workflow featuring:**

**Automated Testing Suite**:

* Unit test generation and execution for React components[[22]](#fn22)
* Integration testing for component interactions
* Accessibility audits using axe-core or similar tools[[22]](#fn22)
* Performance testing and bundle size analysis

**Visual Comparison System**:

* Screenshot comparison between original AngularJS and new React applications
* Visual regression testing to ensure UI consistency
* Cross-browser compatibility verification

**Functional Validation**:

* Automated verification of all user stories and acceptance criteria[[16]](#fn16)
* Feature parity checking between original and migrated applications
* Data flow validation and API integration testing

**Performance Analysis**:

* Load time comparisons between AngularJS and React versions[[19]](#fn19)[[12]](#fn12)
* Bundle size optimization recommendations
* Runtime performance metrics and optimization suggestions

**Reporting System**:

* Generate detailed quality assessment reports with metrics and recommendations
* Create actionable feedback for Agent 4 improvements
* Maintain quality gates for deployment readiness

**Feedback Loop**: Structured communication back to Agent 4 for iterative improvements[[2]](#fn2)

**Agent 6: Human Validation Agent**

**Create a sophisticated human testing workflow with:**

**Deployment Automation**:

* Automated deployment to staging environment using CI/CD pipelines[[24]](#fn24)[[8]](#fn8)
* Environment configuration and database seeding
* SSL certificate and domain setup for testing

**Feedback Collection System**:

* Multi-step forms with rich media support for comprehensive testing feedback[[4]](#fn4)[[9]](#fn9)
* Screenshot and video upload capabilities for issue documentation
* Structured feedback categories (functionality, UI/UX, performance)
* Priority and severity classification for reported issues

**Issue Management**:

* AI-powered categorization and prioritization of feedback[[4]](#fn4)
* Integration with project management tools (Jira, GitHub Issues)[[7]](#fn7)
* Automatic routing of critical issues for immediate attention

**Approval Workflow**:

* Multi-level approval system with conditional routing based on feedback[[4]](#fn4)[[9]](#fn9)
* Stakeholder notification system via email/Slack[[7]](#fn7)[[8]](#fn8)
* Version control and approval tracking

**Communication Hub**:

* Automated notifications for testing milestones and deadlines
* Real-time status updates for all stakeholders
* Integration with team communication platforms

**Resolution Tracking**: Monitor feedback resolution progress and closure validation[[4]](#fn4)

**Agent 7: Deployment and Git Management Agent**

**Design a comprehensive deployment workflow that:**

**Repository Management**:

* Create and configure new Git repositories with branch protection rules[[7]](#fn7)[[8]](#fn8)
* Setup proper Git workflows (GitFlow, GitHub Flow) based on team preferences
* Configure repository settings, permissions, and access controls

**CI/CD Pipeline Setup**:

* Automated build pipeline configuration using GitHub Actions or similar[[7]](#fn7)[[8]](#fn8)
* Deployment pipeline setup for staging and production environments[[24]](#fn24)
* Integration with cloud platforms (AWS, Vercel, Netlify) for hosting

**Environment Management**:

* Automated deployment to multiple environments with proper configuration[[24]](#fn24)
* Environment variable management and secrets handling
* Database migration and seeding for production deployment

**Monitoring and Alerting**:

* Application performance monitoring setup (New Relic, DataDog)
* Error tracking integration (Sentry, Bugsnag)
* Uptime monitoring and alerting configuration

**Documentation Generation**:

* Automated deployment guides and setup instructions
* API documentation generation and hosting
* User guides and feature documentation

**Archive Management**: Store migration artifacts and comprehensive documentation for future reference[[7]](#fn7)

**Agent 8: Monitoring and Orchestration Agent**

**Build a comprehensive monitoring workflow that:**

**Workflow Coordination**:

* Master orchestration of all agent interactions using MCP Client nodes[[2]](#fn2)
* Sequential and parallel workflow execution management
* Dynamic routing based on agent outputs and conditions

**Activity Logging**:

* Comprehensive logging of all agent activities to PostgreSQL[[3]](#fn3)[[25]](#fn25)
* Performance metrics tracking (execution times, resource usage)
* Error logging with stack traces and context information

**Analytics Dashboard**:

* Real-time status dashboards showing migration progress
* Performance analytics and bottleneck identification
* Success rate tracking and improvement recommendations

**Error Handling System**:

* Automatic detection of workflow failures and issues[[25]](#fn25)
* Intelligent retry mechanisms with exponential backoff
* Escalation procedures for critical failures requiring human intervention

**Continuous Improvement Engine**:

* Analysis of completed migrations for process optimization
* Pattern recognition for common issues and automated solutions
* Performance optimization recommendations based on historical data

**Technical Implementation Patterns**

**Database Schema Design**

**PostgreSQL Integration Pattern**:

-- Migration Sessions Table  
CREATE TABLE migration\_sessions (  
 session\_id UUID PRIMARY KEY,  
 project\_name VARCHAR(255) NOT NULL,  
 created\_at TIMESTAMP DEFAULT NOW(),  
 status VARCHAR(50) DEFAULT 'active',  
 metadata JSONB,  
 configuration JSONB  
);  
  
-- Agent Activities Table  
CREATE TABLE agent\_activities (  
 activity\_id UUID PRIMARY KEY,  
 session\_id UUID REFERENCES migration\_sessions(session\_id),  
 agent\_name VARCHAR(100) NOT NULL,  
 activity\_type VARCHAR(100) NOT NULL,  
 started\_at TIMESTAMP DEFAULT NOW(),  
 completed\_at TIMESTAMP,  
 status VARCHAR(50) DEFAULT 'running',  
 input\_data JSONB,  
 output\_data JSONB,  
 error\_details JSONB  
);  
  
-- Human Feedback Table  
CREATE TABLE human\_feedback (  
 feedback\_id UUID PRIMARY KEY,  
 session\_id UUID REFERENCES migration\_sessions(session\_id),  
 feedback\_type VARCHAR(100) NOT NULL,  
 feedback\_data JSONB,  
 provided\_by VARCHAR(255),  
 created\_at TIMESTAMP DEFAULT NOW(),  
 processed BOOLEAN DEFAULT FALSE  
);

**Vector Database Configuration**

**Implement efficient vector storage using:**

* Pinecone or Chroma for production deployments[[5]](#fn5)
* Proper chunking strategies (1000-2000 characters per chunk)
* Semantic embeddings using OpenAI text-embedding-3-small[[5]](#fn5)
* Namespace organization for project isolation[[5]](#fn5)
* Automatic cleanup mechanisms post-migration[[5]](#fn5)

**Multi-Agent Communication Protocol**

**MCP Implementation Pattern**:

* Standardized message formats across all agents[[2]](#fn2)
* Event-driven architecture with clear message routing
* Failure recovery mechanisms for communication errors
* Message queuing for high-volume processing
* Dead letter queues for failed message handling

**Human-in-the-Loop Integration**

**Approval and Validation Points**:

* Form nodes with structured data collection[[4]](#fn4)[[9]](#fn9)
* Conditional routing based on approval status
* Timeout handling for delayed human responses[[4]](#fn4)
* Multi-stakeholder approval workflows with role-based permissions
* Integration with external approval systems (Slack, Microsoft Teams)[[9]](#fn9)

**Security and Compliance**

**Credential Management**

* Environment variables for all sensitive configurations[[25]](#fn25)
* OAuth 2.0 integration for third-party services
* API key rotation and management procedures
* Role-based access control for different workflow stages

**Data Protection**

* Encryption at rest for vector database storage[[5]](#fn5)
* Secure transmission protocols for inter-agent communication
* Data retention policies and automatic cleanup procedures[[5]](#fn5)
* Compliance with data protection regulations (GDPR, CCPA)

**Performance Optimization**

**Scalability Considerations**

* Horizontal scaling capabilities for high-volume migrations
* Load balancing for concurrent workflow executions
* Resource monitoring and automatic scaling triggers
* Caching strategies for frequently accessed data

**Monitoring and Alerting**

* Real-time performance dashboards[[3]](#fn3)
* SLA monitoring and alerting thresholds
* Capacity planning based on historical usage patterns
* Cost optimization recommendations for cloud resources

This comprehensive prompt provides the foundation for creating a production-ready AngularJS to React migration system using n8n, incorporating industry best practices for multi-agent workflows, database integration, human validation, and deployment automation[[1]](#fn1)[[2]](#fn2)[[3]](#fn3)[[4]](#fn4).

⁂

1. <https://www.youtube.com/watch?v=o2Pubq36Pao>

1. <https://community.n8n.io/t/exploring-multi-agent-patterns-in-n8n-using-mcp-triggers-clients-without-webhooks/114944>

1. <https://n8n.io/workflows/4464-export-ai-agent-conversation-logs-from-postgres-to-google-sheets/>

1. <https://n8n.io/workflows/2907-a-very-simple-human-in-the-loop-email-response-system-using-ai-and-imap/>

1. <https://www.youtube.com/watch?v=VItv8MJ9xBs>

1. <https://docs.n8n.io/integrations/builtin/cluster-nodes/root-nodes/n8n-nodes-langchain.vectorstoreinmemory/>

1. <https://docs.github.com/en/issues/planning-and-tracking-with-projects/automating-your-project/automating-projects-using-actions>

1. <https://github.blog/developer-skills/github/a-beginners-guide-to-ci-cd-and-automation-on-github/>

1. <https://www.youtube.com/watch?v=z-gDDajSSNE>

1. <https://vulncheck.com/blog/understanding-software-dependency-graphs>

1. <https://en.wikipedia.org/wiki/Dependency_graph>

1. <https://buttercms.com/blog/migrating-from-angularjs-to-react/>

1. <https://www.softcrylic.com/blogs/the-top-six-points-to-consider-when-migrating-from-angularjs-to-reactjs/>

1. <https://dzone.com/articles/continuous-code-quality-and-automated-code-review>

1. <https://www.awesomecodereviews.com/automation/automated-code-reviews/>

1. <https://projectcopilot.co/posts/implementing-user-stories-with-ai-powered-code-generation/>

1. <https://github.com/melahn/user-story-generator>

1. <https://n8n.io/workflows/3725-wordpress-content-automation-machine-with-human-in-the-loop-and-deep-research/>

1. <https://software.iquasar.com/blog/angular-js-to-react-migration-roi-for-your-business/>

1. <https://www.babbel.com/en/magazine/how-we-moved-from-angularjs-to-react>

1. <https://dev.to/fpaghar/react-component-design-patterns-part-1-5f0g>

1. <https://www.uxpin.com/studio/blog/integrating-react-components-with-design-patterns/>

1. <https://best-practice-and-impact.github.io/qa-of-code-guidance/continuous_integration.html>

1. <https://newsletter.techworld-with-milan.com/p/what-are-deployment-patterns>
2. <https://docs.n8n.io/hosting/logging-monitoring/logging/>