Ops Manager Architecture Document

# 1. Overview

The Ops Manager module is designed to streamline and centralize core operational functions. It simplifies routine tasks such as automating workarounds, monitoring their execution, reprocessing transactions, and supporting issue resolution through a built-in triage agent. The platform also integrates AI-powered tools for sentiment analysis, anomaly detection, and predictive analytics—enabling proactive and data-driven decision-making.

Designed with flexibility in mind, the module allows for easy integration of additional features and AI capabilities over time, ensuring continuous improvement in operational efficiency and adaptability to evolving business needs.

# 2. Features

Key features of the Ops Manager module include:

**1. Manage Workaround Automation Workflow via UI**

This feature replaces manual SQL scripting with a user-friendly interface to define, manage, and track workaround steps.

* **Dynamic Form Builder**: Allows users to configure job steps with fields like job name, step sequence, command type (SQL/script), and error handling.
* **Execution Tracker**: Displays job run history with timestamps, status, and failure reasons.
* **Audit Logging**: Maintains a history of changes for compliance and traceability.
* **Validation Engine**: Ensures correct syntax and logical flow before saving configurations.

**2. Sentiment Insights (AI**

This feature analyzes user feedback from various HIX portal pages to provide actionable insights into user satisfaction and pain points.

* **NLP Models**: Uses model BERT to classify feedback as Positive, Neutral, or Negative.
* **Page-Level Sentiment**: Aggregates sentiment scores by portal page to identify UX issues.
* **Trend Analysis**: Visualizes sentiment trends over time to monitor improvements or regressions.

**3. Anomaly Viewer**

Automatically detects unusual patterns in batch job executions to proactively flag potential issues.

* **Execution Time Monitoring**: Flags jobs that run significantly longer or shorter than expected.
* **Result Deviation Detection**: Identifies mismatches between expected and actual outcomes.
* **Model-Based Detection**: Uses statistical models (e.g., Z-score, Isolation Forest) to detect outliers.
* **Alerting System**: Notifies users of anomalies via dashboard indicators or optional email alerts.

**4. Predictive Analysis**

Forecasts future operational metrics to support planning and resource allocation.

* **Notice Volume Estimation**: Projects the number of notices to be generated.
* **MMIS Transaction Prediction**: Estimates transaction counts based on historical trends.
* **Modeling Techniques**: Uses linear regression models, or XGBoost

**5. Reprocessing transactions**

Enables users to regenerate transactions to resolve failed or incomplete service requests.

* **Self-Service Trigger**: UI-based trigger to re-initiate transactions without backend intervention.
* **Validation Checks**: Ensures prerequisites are met before regeneration.
* **Audit Trail**: Logs all regeneration attempts with timestamps and user details.

**6. Triage Agent**

The triage agent automates issue detection, classification, and prioritization by monitoring logs and system events. It assists with root cause analysis, recommends remediation steps, and logs all actions for audit. Integrated with AI, it supports anomaly detection, predictive insights, and learns from past incidents to improve future response.

# 3. System Architecture

The architecture consists of three core layers:

**1. Frontend Layer – React.js (Agent Portal UI)**

This is the user-facing interface integrated into the HIX Agent Portal. It allows operations team to interact with the system through intuitive dashboards and forms.

**Key Technologies**

* **React.js**: Component-based UI framework
* **Recharts**: For visualizing KPIs, sentiment trends, and anomaly graphs

**Modules**

* **Workaround Entry UI**: Form-based interface to define and manage batch job steps
* **Execution Tracker**: Displays batch job run history with filters and status indicators
* **Sentiment Dashboard**: Visualizes user feedback sentiment across HIX pages
* **Anomaly Viewer**: Highlights abnormal job executions or durations

**2. Backend Layer – Spring Boot (HIX API Layer)**

This layer acts as the orchestrator between the frontend and AI services. It handles business logic, validation, and data persistence.

**🔧 Key Technologies**

* **Spring Boot**: Java-based backend framework
* **REST APIs**: Exposed for frontend and AI service communication
* **Open API/Swagger**: Use a design-first approach to define the API in a .yml file, and then use OpenAPI Generator to generate the implementation code

**🧩 Responsibilities**

* **Workflow Management**: Validates and stores workaround configurations
* **Execution Logging**: Tracks batch job runs and stores results
* **AI Integration**: Sends data to AI microservices and receives enriched insights
* **Audit Logging**: Maintains history of changes and actions

**3. AI Microservices Layer – Python Flask**

This layer provides intelligent services to enhance operational decision-making using machine learning and natural language processing.

**🔧 Key Technologies**

* **Flask**: Lightweight Python web framework for REST APIs
* **scikit-learn**: For anomaly detection and predictive models
* **transformers (HuggingFace)**: For sentiment analysis using BERT model.
* **pandas/numpy**: For data manipulation

**🧠 Microservices**

* **Sentiment Analysis Service**:
  + Input: User feedback text
  + Output: Sentiment label (Positive, Neutral, Negative)
  + Models: VADER, BERT, or DistilBERT
* **Anomaly Detection Service**:
  + Input: Batch job execution logs
  + Output: Flags for unusual durations or failures
  + Models: Isolation Forest, Z-score, or LSTM
* **Predictive Analysis Service**:
  + Input: Historical batch job data
  + Output: Forecast of notice count and MMIS transactions
  + Models: Linear Regression, ARIMA, or XGBoost

**🧩 Deployment**

* Containerized using Docker
* Internal network access only
* Scalable via Kubernetes (optional)

# 4. AI Services

The AI microservices are implemented using Python and Flask. The services include:  
- Sentiment Analysis: Uses NLP models (e.g., BERT, VADER) to analyze user feedback.  
- Anomaly Detection: Uses statistical and ML models to detect unusual patterns in batch job execution.  
- Predictive Analysis: Uses regression models to forecast batch job metrics.

# 5. Python Modules Used

* Flask - for building REST APIs
* scikit-learn - for machine learning models
* pandas - for data manipulation
* numpy - for numerical operations
* matplotlib / seaborn - for visualizations
* nltk / transformers - for sentiment analysis

# 6. Data Flow Diagram

The following diagram illustrates the data flow between the UI, backend, and AI services.

# A diagram of a computer AI-generated content may be incorrect.

# 7. Roadmap

1. Phase 1: Sentiment Analysis + Manual Workaround Logging UI
2. Phase 2: Anomaly Detection
3. Phase 3: Integrated ML-driven Decision Support
4. Phase 4: AI-Powered Triage & Monitoring Agents

# 8. Technology Stack

* Frontend: React.js, Recharts
* Backend: Spring Boot, REST APIs
* AI Services: Python, Flask, scikit-learn, transformers
* Deployment: Docker, Kubernetes (optional)