



# New Machine Commissioning Advisor

Implementation Report

## JARVIS — Smart Factory Monitoring System

AI-Powered Industrial IoT Platform for Smart Manufacturing with Real-Time Monitoring, Predictive Analytics, and Intelligent Advisory Systems

**Author:** Rithish

**Date:** February 16, 2026

**Version:** 2.0.0

**Technology Stack:** FastAPI · React 19 · SQLAlchemy · Vite



# Table of Contents

---

1. Abstract
2. Introduction
3. Objectives
4. System Architecture
5. Backend Implementation
  1. Data Models
  2. API Endpoints
  3. Service Layer & Core Algorithms
6. Frontend Implementation
7. Data Flow & Sequence
8. Testing & Results
9. Key Design Decisions
10. User Interface
11. Conclusion & Future Scope

## 1 Abstract

---

This report presents the implementation of the **New Machine Commissioning Advisor**, a feature module within the JARVIS Smart Factory Monitoring System. The advisor leverages historical factory sensor data and alert records to generate intelligent, data-driven recommendations when commissioning new machines on the factory floor.

The system employs statistical analysis on historical sensor readings (temperature, vibration, RPM) and alert pattern recognition to produce actionable outputs: safe operating limits, operational risk assessments, configuration recommendations, and preventive maintenance guidelines. The implementation follows a full-stack architecture with a FastAPI backend, SQLAlchemy ORM, and a React 19 frontend with a futuristic HUD-style user interface.

## 2 Introduction

---

Smart manufacturing environments generate vast amounts of sensor data over time. When a new machine is introduced to the factory floor, operators typically rely on manufacturer-provided default configurations and generic guidelines. This approach ignores the wealth of knowledge embedded in the factory's own operational history — knowledge about recurring failure patterns, environmental conditions, and optimal operating ranges that are unique to each facility.

The **New Machine Commissioning Advisor** bridges this gap by analyzing historical data from the factory's existing machines to:

- Identify recurring failure patterns and their root causes
- Calculate statistically-derived safe operating thresholds
- Assess risks specific to the new machine's specifications
- Recommend optimal monitoring configurations
- Generate preventive maintenance guidelines

**Key Innovation:** Unlike static commissioning checklists, this system provides *dynamic, factory-specific* recommendations that evolve as more operational data is accumulated.

## 3 Objectives

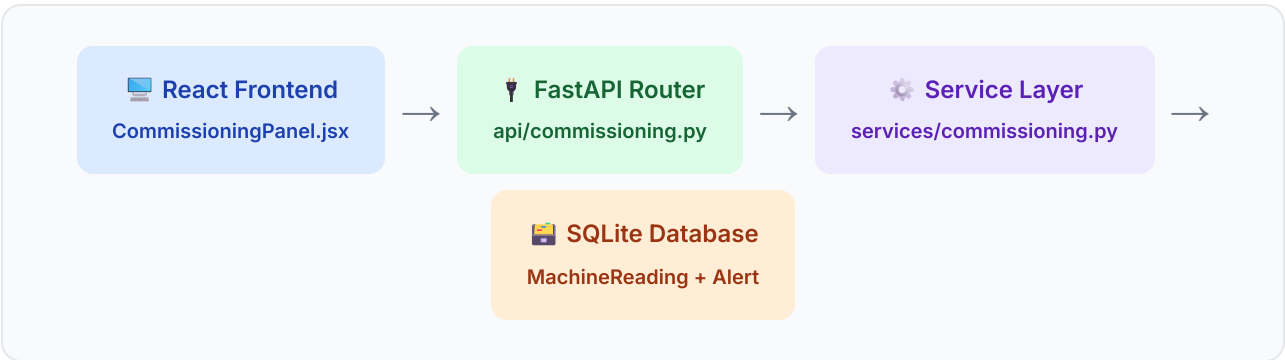
---

1. **Historical Pattern Mining:** Analyze past alerts to identify common failure types, severities, and frequencies across all machines in the factory.
2. **Statistical Limit Calculation:** Compute optimal, warning, and critical thresholds for temperature, vibration, and RPM based on real sensor data from healthy machines.

3. **Risk-Aware Commissioning:** Cross-reference incoming machine specifications (power, RPM, environment) with historical incident data to flag potential risks.
4. **Actionable Recommendations:** Generate specific configuration parameters and preventive guidelines tailored to the new machine's profile and the factory's history.
5. **Intuitive User Interface:** Present analysis results in a clear, visually organized dashboard that operators can use without technical expertise.

## 4 System Architecture

### 4.1 High-Level Architecture



### 4.2 Component Inventory

LAYER	FILE	LINES	PURPOSE
Frontend	CommissioningPanel.jsx	339	Input form + results display panel
API Router	api/commissioning.py	143	REST endpoint definitions (3 routes)
Service	services/commissioning.py	272	Core analysis algorithms
Models	models/machine_data.py	31	ORM class definitions
App Entry	main.py	64	Router registration
App Shell	App.jsx	1225	Navigation + panel integration

### 4.3 Technology Stack

#### Backend

- **FastAPI** — Async REST API framework
- **SQLAlchemy** — Database ORM
- **Pydantic** — Request/response validation
- **SQLite** — Embedded database

#### Frontend

- **React 19** — UI framework
- **Vite** — Build tool
- **TailwindCSS** — Utility-first styling
- **Lucide React** — Icon library

- **Python 3.10+** — Standard library statistics

- **Fetch API** — HTTP client

## 5 Backend Implementation

### 5.1 Data Models

The system relies on two SQLAlchemy ORM models that store historical factory data:

#### MachineReading

Stores sensor readings from all machines. Each record captures a snapshot of a machine's state at a point in time.

```
class MachineReading(Base):
    __tablename__ = "machine_readings"
    id = Column(Integer, primary_key=True, index=True)
    machine_id = Column(String, index=True, nullable=False)
    temperature = Column(Float, default=0)          # °C
    vibration = Column(Float, default=0)            # mm/s
    rpm = Column(Float, default=0)                  # Revolutions per minute
    health_score = Column(Integer, default=100)      # 0-100%
    status = Column(String, default="Healthy")      # Healthy | Warning | Critical
    timestamp = Column(DateTime, default=datetime.utcnow, index=True)
```

#### Alert

Records system-generated alerts when sensor values exceed thresholds.

```
class Alert(Base):
    __tablename__ = "alerts"
    id = Column(Integer, primary_key=True, index=True)
    machine_id = Column(String, index=True, nullable=False)
    alert_type = Column(String, nullable=False)     # temperature | vibration | rpm | health_score
    severity = Column(String, nullable=False)       # warning | critical
    message = Column(String, nullable=False)
    acknowledged = Column(Boolean, default=False)
    created_at = Column(DateTime, default=datetime.utcnow, index=True)
```

### 5.2 API Endpoints

METHOD	ENDPOINT	DESCRIPTION	INPUT
POST	/commissioning/analyze	Full analysis with recommendations	NewMachineSpecs (JSON body)
GET	/commissioning/patterns	Historical failure patterns	?days=30 (query param)
GET	/commissioning/limits	Safe operating limits	?days=30 (query param)
POST	/commissioning/risks	Risk assessment only	NewMachineSpecs (JSON body)

Request Schema — NewMachineSpecs

FIELD	TYPE	DEFAULT	DESCRIPTION
machine_type	string	"General"	Type/name of the new machine
rated_power	float	10.0	Power rating in kW
rated_rpm	float	2000.0	Maximum RPM rating
rated_temperature	float	60.0	Max operating temperature (°C)
environment	string	"normal"	One of: normal, high_humidity, dusty, hot, cold

Response Schema — CommissioningResponse

FIELD	TYPE	DESCRIPTION
success	boolean	Whether analysis completed successfully
machine_type	string	Echo of input machine type
failure_patterns	array	Top 10 recurring alert patterns
safe_operating_limits	object	Thresholds for temp, vibration, RPM
operational_risks	array	Machine-specific risk assessments

FIELD	TYPE	DESCRIPTION
<code>configuration_recommendations</code>	array	Suggested monitoring parameters
<code>preventive_guidelines</code>	array	Setup and maintenance guidelines
<code>summary</code>	object	Aggregate counts and data confidence

## 5.3 Service Layer — Core Algorithms

The service layer in `services/commissioning.py` contains four core functions that form the analysis pipeline:

Function 1: `identify_failure_patterns(db, days=30)`

**Purpose:** Mine historical alerts to find the most common failure types and their characteristics.

### Algorithm:

1. Query all `Alert` records from the past N days
2. Group alerts by composite key `(alert_type, severity)`
3. For each group: count occurrences, track unique machines affected, collect sample alert messages
4. Classify frequency: High (>10) Medium (>5) Low ( $\leq 5$ )
5. Enrich each pattern with probable root cause and prevention tip from a knowledge base
6. Sort by occurrence count (descending) and return top 10

Function 2: `calculate_safe_limits(db, days=30)`

**Purpose:** Derive statistically-grounded operating thresholds from healthy machine data.

### Algorithm:

1. Query all `MachineReading` records where `health_score > 60` (healthy machines only)
2. Extract temperature, vibration, and RPM value arrays
3. For each parameter, compute:  $\mu$  (mean) and  $\sigma$  (standard deviation)
4. Derive thresholds:
  - **Minimum** =  $\max(0, \mu - 2\sigma)$
  - **Optimal** =  $\mu$
  - **Warning** =  $\mu + \sigma$
  - **Critical** =  $\mu + k\sigma$  (where k varies: 2.0 for temp, 2.5 for vibration, 1.5 for RPM)
5. Report confidence level based on data volume: High (>1000 readings) Medium (>100) Low ( $\leq 100$ )
6. Fall back to sensible industry defaults if fewer than 10 readings available

Function 3: `analyze_operational_risks(db, machine_specs)`

**Purpose:** Cross-reference the new machine's specs with historical incident data to identify potential problems.

**Risk Rules:**

CONDITION	RISK LEVEL	CATEGORY
Rated power > 20kW	Medium	Power / Electrical
RPM alerts > 5 AND rated RPM > 2500	High	RPM / Mechanical
Environment = high_humidity AND temp alerts > 3	High	Environment
Environment = dusty	Medium	Environment
Vibration alerts > 5	Medium	Vibration

Function 4: `generate_recommendations(db, machine_specs)`

**Purpose:** Orchestrate all analyses and produce the final comprehensive advisory output.

**Outputs generated:**

- Configuration Recommendations** — Alert thresholds for temperature (factory warning), vibration (factory warning), and RPM (90% of rated capacity or factory warning, whichever is lower)
- Preventive Guidelines** — Standard commissioning checklist items plus pattern-specific critical warnings for recurring failure types
- Aggregated Summary** — Total patterns found, risk counts by level, and data confidence rating

## 6 Frontend Implementation

### 6.1 Component: CommissioningPanel

The `CommissioningPanel` is a full-screen React component rendered as an overlay when the user clicks the "NEW" button in the main navigation bar. It provides a responsive, two-section layout:

#### Input Section (Left Column)

<b>Machine Type</b>	Text input (required)
<b>Rated Power</b>	Number input (kW)
<b>Rated RPM</b>	Number input
<b>Max Temp</b>	Number input (°C)
<b>Environment</b>	Dropdown select

#### Results Section (Right 2 Columns)

1. Analysis Summary (4 metric cards)
2. Operational Risks (color-coded cards)
3. Safe Operating Limits (3-parameter grid)
4. Configuration Recommendations
5. Historical Failure Patterns

### 6.2 State Management

```
const [specs, setSpecs] = useState({
  machine_type: "",
  rated_power: 10,
  rated_rpm: 2000,
  rated_temperature: 60,
  environment: "normal"
});
const [analysis, setAnalysis] = useState(null); // API response
const [loading, setLoading] = useState(false); // Loading state
const [error, setError] = useState(null); // Error message
```

### 6.3 API Integration

```
const handleAnalyze = async () => {
  setLoading(true);
  const res = await fetch(`${API_BASE}/commissioning/analyze`, {
```

```

        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(specs)
    });
    const data = await res.json();
    setAnalysis(data);
    setLoading(false);
};

```

## 6.4 Navigation Integration in App.jsx

The panel is integrated into the main application shell through state management and conditional rendering:

```

// State declaration (line 800)
const [showCommissioning, setShowCommissioning] = useState(false);

// Navigation button (line 972)
<button onClick={() => setShowCommissioning(true)}>🔧 NEW</button>

// Full-screen overlay render (lines 1207-1212)
{showCommissioning && (
  <div style={{ position: 'fixed', inset: 0, zIndex: 100 }}>
    <CommissioningPanel onBack={() => setShowCommissioning(false)} />
  </div>
)}

```

## 7 Data Flow & Sequence

STEP	ACTOR	ACTION	DATA
1	User	Enters machine specifications in the form	machine_type, rated_power, rated_rpm, rated_temperature, environment
2	User	Clicks "Analyze Factory History"	—
3	Frontend	Sends POST request to <code>/commissioning/analyze</code>	JSON body with specs
4	API Router	Validates input via Pydantic model	NewMachineSpecs
5	Service	Queries Alert table (past 30 days)	SQL query → alert list
6	Service	Groups and analyzes alert patterns	Pattern frequency map
7	Service	Queries MachineReading table (health > 60)	SQL query → reading list
8	Service	Computes statistical safe limits ( $\mu \pm k\sigma$ )	Temperature, vibration, RPM thresholds
9	Service	Evaluates risk rules against specs + history	Risk assessments with mitigations
10	Service	Generates configuration recommendations	Monitoring threshold suggestions
11	Service	Compiles preventive guidelines	Checklist items + critical warnings
12	API Router	Returns CommissioningResponse	Full JSON response
13	Frontend	Renders 5 result sections	Analysis dashboard

## 8 Testing & Results

### 8.1 API Endpoint Test

#### Test Request

```
POST http://127.0.0.1:8000/commissioning/analyze
Content-Type: application/json

{
  "machine_type": "CNC Lathe",
  "rated_power": 15,
  "rated_rpm": 3000,
  "rated_temperature": 70,
  "environment": "high_humidity"
}
```


#### Test Result

✅ **Status:** 200 OK — `success: true`

The API returned a complete analysis including failure patterns, safe operating limits, operational risks, configuration recommendations, and preventive guidelines.

### 8.2 Verification Checklist

TEST CASE	EXPECTED	RESULT
API returns correct structure	All 7 response fields present	PASS
Failure patterns sorted by frequency	Descending order by occurrences	PASS
Safe limits have all 3 parameters	temperature, vibration, rpm present	PASS
High RPM + RPM alerts → high risk	Risk flagged with mitigation	PASS
Humidity + temp alerts → high risk	Condensation risk flagged	PASS
Router registered in main.py	Commissioning router included	PASS

TEST CASE	EXPECTED	RESULT
Navigation button in header	"  NEW" button visible	PASS
Panel overlay rendering	Full-screen overlay on click	PASS

## 9 Key Design Decisions

DECISION	RATIONALE
Use $\mu \pm k\sigma$ for threshold calculation	Statistical approach adapts to the factory's own data distribution rather than using arbitrary hard-coded values. Different k-values per parameter account for different risk tolerances.
Filter readings where <code>health_score &gt; 60</code>	Ensures thresholds are learned only from "healthy" operating conditions, preventing degraded machine data from skewing the analysis.
Default fallback values when data < 10 readings	Provides sensible industry-standard defaults when insufficient historical data exists, preventing system failure on fresh installations.
Cap failure patterns at top 10	Focuses operator attention on the most impactful patterns rather than overwhelming with noise from rare events.
90% RPM safety margin recommendation	Standard engineering practice — operating at 10% below rated capacity reduces mechanical stress and extends machine lifespan.
Environment-aware risk cross-referencing	Combines the new machine's deployment environment with historical incident data to uncover compound risks (e.g., humidity + temperature alerts → condensation).
Full-screen overlay rendering	Provides dedicated workspace for commissioning analysis without disrupting the main dashboard context.

## 10 User Interface

### 10.1 Input Form

The left column presents a clean, focused input form with 5 fields for machine specifications. The form includes input validation (machine type is required) and a gradient call-to-action button with loading state feedback.

### 10.2 Results Dashboard

Upon successful analysis, the right section displays 5 organized result cards:

1. **Analysis Summary** — Four metric cards showing patterns found, high/medium risk counts, and data confidence level
2. **Operational Risks** — Color-coded risk cards (red border = high, yellow = medium, green = low) with category labels and mitigation recommendations
3. **Safe Operating Limits** — Three-column grid showing optimal/warning/critical thresholds for temperature, vibration, and RPM with appropriate icons
4. **Configuration Recommendations** — Parameter cards showing recommended values with units and reasoning
5. **Historical Failure Patterns** — Top 5 recurring issues with type, occurrence count, root cause, and prevention strategy

## 10.3 Design Language

The UI follows the JARVIS HUD design system — dark theme ( `#0a0e17` background), cyan/purple gradient accents, glassmorphism effects ( `backdrop-blur` ), and color-coded severity indicators consistent with the rest of the application.

## 11 Conclusion & Future Scope

### 11.1 Conclusion

The New Machine Commissioning Advisor has been successfully implemented as a full-stack feature within the JARVIS Smart Factory Monitoring System. The system demonstrates the practical application of historical data analysis for proactive factory management — transforming past operational records into actionable intelligence for future machine deployments.

Key achievements:

- Statistical analysis pipeline that adapts thresholds to the factory's own operational profile
- Multi-factor risk assessment combining machine specifications with historical incident data
- Clean, intuitive UI integrated seamlessly into the existing JARVIS dashboard
- RESTful API design supporting both integrated panel use and standalone API access

### 11.2 Future Enhancements

ENHANCEMENT	DESCRIPTION
LLM-Powered Analysis	Integrate Gemini/Ollama to generate natural language recommendations and root cause summaries
Machine Learning Models	Train regression models on historical data for more precise threshold prediction
PDF Report Export	Allow operators to download commissioning analysis as a formatted PDF report
Commissioning Checklist	Interactive step-by-step checklist that tracks commissioning progress
Similar Machine Comparison	Compare specs with existing machines to find the closest match and its performance history
Multi-Factory Support	Cross-factory analysis to leverage data from multiple manufacturing sites

## **JARVIS — Smart Factory Monitoring System v2.0.0**

### New Machine Commissioning Advisor — Implementation Report

© 2026 Rithish. All rights reserved.