Aquapurite ERP - AI Enhancement Roadmap

Vision: AI-Enabled ERP for Consumer Durables

Transform Aquapurite ERP into a genuinely AI-powered platform that automates decisions, predicts outcomes, and provides intelligent recommendations across all business functions.

What Makes an ERP "AI-Enabled"?

An ERP can legitimately claim to be "AI-enabled" when it:

1. **Predicts** - Uses historical data to forecast future outcomes
2. **Recommends** - Suggests optimal actions based on patterns
3. **Automates** - Makes decisions without human intervention
4. **Learns** - Improves accuracy over time with more data
5. **Understands** - Processes natural language and unstructured data

---

Current AI Capabilities (Already Built)

Feature	Location	Status
Smart Reorder Suggestions	`/dashboard/insights/reorder`	✅ Active
Customer Churn Prediction	`/dashboard/insights/churn-risk`	✅ Active
Allocation Rules Engine	`/dashboard/logistics/allocation-rules`	✅ Active

Phase 1: Quick Wins (2-4 weeks each)

1.1 AI-Powered Demand Forecasting

Business Value: Reduce stockouts by 40%, reduce excess inventory by 25%

Implementation:

```python
# app/services/ai/demand_forecasting.py

from prophet import Prophet
import pandas as pd

class DemandForecastingService:
    """
    Predict future demand for products using time-series analysis
    """

    async def forecast_product_demand(
        self,
        product_id: str,
        horizon_days: int = 30
    ) -> dict:
        """
        Returns:
        {
            "product_id": "xxx",
            "forecasts": [
                {"date": "2026-01-20", "predicted_qty": 45, "lower": 38, "upper":
52},
                ...
            ],
            "trend": "increasing",
            "seasonality": "weekly_peak_friday",
            "recommended_reorder_qty": 150,
            "confidence": 0.87
        }
        """
        # 1. Get historical sales data
        sales_data = await self.get_sales_history(product_id)

        # 2. Prepare for Prophet
        df = pd.DataFrame({
            'ds': sales_data['date'],
            'y': sales_data['quantity']
        })

        # 3. Train model
        model = Prophet(
            yearly_seasonality=True,
            weekly_seasonality=True,
            daily_seasonality=False
        )
        model.fit(df)

        # 4. Make predictions
        future = model.make_future_dataframe(periods=horizon_days)
        forecast = model.predict(future)

        return self.format_forecast(forecast)
```

API Endpoint:

```
GET /api/v1/ai/forecast/demand/{product_id}?days=30
POST /api/v1/ai/forecast/demand/bulk
```

Frontend Page: `/dashboard/insights/demand-forecast`

1.2 Intelligent Invoice Processing (OCR + AI)

Business Value: Reduce manual data entry by 80%, process invoices in seconds

Implementation:

```
# app/services/ai/invoice_ocr.py

import pytesseract
from pdf2image import convert_from_path
import openai  # or anthropic

class InvoiceOCRService:
    """
    Extract data from scanned invoices/bills using OCR + LLM
    """

    async def process_invoice(self, file_path: str) -> dict:
        """
        Returns:
        {
            "vendor_name": "ABC Supplies",
            "vendor_gstin": "27AABCU9603R1ZM",
            "invoice_number": "INV-2026-001",
            "invoice_date": "2026-01-10",
            "items": [
                {"description": "RO Membrane 80 GPD", "qty": 10, "rate": 450,
"amount": 4500}
            ],
            "subtotal": 4500,
            "gst_amount": 810,
            "total": 5310,
            "confidence": 0.92,
            "matched_vendor_id": "uuid-xxx",  # Auto-matched from database
            "matched_products": [...]          # Auto-matched products
        }
        """
        # 1. Convert PDF to image
        images = convert_from_path(file_path)

        # 2. OCR extraction
        raw_text = pytesseract.image_to_string(images[0])

        # 3. Use LLM to structure the data
        structured_data = await self.extract_with_llm(raw_text)

        # 4. Match with existing vendors/products
        structured_data = await self.match_entities(structured_data)

        return structured_data

    async def extract_with_llm(self, raw_text: str) -> dict:
        """Use Claude/GPT to extract structured data from OCR text"""
        prompt = f"""
        Extract invoice details from this text:
        {raw_text}
        Return JSON with: vendor_name, gstin, invoice_number, date, items[], totals
        """
        # Call LLM API
        response = await self.llm_client.complete(prompt)
        return json.loads(response)
```

API Endpoint:

```
POST /api/v1/ai/ocr/invoice  (multipart/form-data)
```

Frontend Page: `/dashboard/procurement/invoice-upload`

1.3 Smart Payment Collection Prediction

Business Value: Improve cash flow visibility, prioritize collection calls

Implementation:

```
# app/services/ai/payment_prediction.py

from sklearn.ensemble import RandomForestClassifier
import pandas as pd

class PaymentPredictionService:
    """
    Predict when customers will pay their invoices
    """

    async def predict_payment(self, invoice_id: str) -> dict:
        """
        Returns:
        {
            "invoice_id": "xxx",
            "amount_due": 50000,
            "due_date": "2026-01-20",
            "predicted_payment_date": "2026-01-25",
            "delay_probability": 0.65,
            "delay_days_predicted": 5,
            "risk_category": "MEDIUM",
            "recommended_action": "Send reminder on due date",
            "factors": [
                {"factor": "Customer payment history", "impact": "negative",
"detail": "Avg 7 days late"},
                {"factor": "Invoice amount", "impact": "neutral", "detail": "Within
usual range"},
                {"factor": "Customer segment", "impact": "positive", "detail": "B2B
regular"}
            ]
        }
        """
        # Features used:
        # - Customer's historical payment behavior
        # - Invoice amount relative to average
        # - Day of week/month
        # - Customer segment
        # - Outstanding balance

        features = await self.extract_features(invoice_id)
        prediction = self.model.predict(features)

        return self.format_prediction(prediction)

    async def get_collection_priority_list(self) -> list:
        """
        Returns invoices sorted by collection priority
        """
        overdue_invoices = await self.get_overdue_invoices()

        for invoice in overdue_invoices:
            invoice['priority_score'] = await self.calculate_priority(invoice)

        return sorted(overdue_invoices, key=lambda x: x['priority_score'],
reverse=True)
```

API Endpoint:

```
GET /api/v1/ai/predict/payment/{invoice_id}
GET /api/v1/ai/predict/collection-priority
```

Frontend Page: `/dashboard/insights/collection-priority`

1.4 AI Chatbot for Internal Queries

Business Value: Instant answers to business questions, reduce report generation time

Implementation:

```
# app/services/ai/erp_chatbot.py

class ERPChatbotService:
    """
    Natural language interface to ERP data
    """

    async def query(self, user_question: str, user_id: str) -> dict:
        """
        Examples:
        - "What were our sales last month?"
        - "Show me pending POs from ABC Vendor"
        - "How many service calls are open in Delhi?"
        - "What's our best selling product?"

        Returns:
        {
            "answer": "Your sales last month were ₹45,23,000 across 234 invoices.",
            "data": {...},  # Structured data if applicable
            "visualization": "bar_chart",  # Suggested chart type
            "follow_up_suggestions": [
                "Compare with previous month",
                "Show product-wise breakdown",
                "Show region-wise breakdown"
            ]
        }
        """
        # 1. Understand intent
        intent = await self.classify_intent(user_question)

        # 2. Generate SQL or API call
        query = await self.generate_query(intent, user_question)

        # 3. Execute and get data
        data = await self.execute_query(query)

        # 4. Generate natural language response
        response = await self.generate_response(data, user_question)

        return response
```

API Endpoint:

```
POST /api/v1/ai/chat
WebSocket: /api/v1/ai/chat/stream
```

Frontend Component: Floating chat widget on all pages

1.5 Predictive Service Maintenance

Business Value: Proactive service calls, reduce customer complaints, increase AMC renewals

Implementation:

```
# app/services/ai/predictive_maintenance.py

class PredictiveMaintenanceService:
    """
    Predict when customer's water purifier needs service
    """

    async def predict_service_need(self, installation_id: str) -> dict:
        """
        Returns:
        {
            "installation_id": "xxx",
            "customer_name": "John Doe",
            "product": "Aquapurite RO 7000",
            "installed_date": "2025-06-15",
            "last_service_date": "2025-12-10",
            "predictions": [
                {
                    "component": "RO Membrane",
                    "predicted_failure_date": "2026-02-15",
                    "confidence": 0.85,
                    "urgency": "MEDIUM",
                    "recommended_action": "Schedule replacement in next service"
                },
                {
                    "component": "Sediment Filter",
                    "predicted_failure_date": "2026-01-25",
                    "confidence": 0.92,
                    "urgency": "HIGH",
                    "recommended_action": "Schedule immediate service call"
                }
            ],
            "overall_health_score": 72,
            "next_recommended_service": "2026-01-20"
        }
        """
        # Factors:
        # - Days since last service
        # - Water quality in area (TDS levels)
        # - Usage patterns (family size)
        # - Component age
        # - Historical failure patterns

        installation = await self.get_installation(installation_id)
        predictions = await self.run_prediction_model(installation)

        return predictions

    async def get_proactive_service_list(self) -> list:
        """
        Get list of installations that need proactive service
        """
        all_installations = await self.get_active_installations()

        service_needed = []
        for installation in all_installations:
            prediction = await self.predict_service_need(installation.id)
            if prediction['overall_health_score'] < 80:
                service_needed.append(prediction)

        return sorted(service_needed, key=lambda x: x['overall_health_score'])
```

API Endpoint:

```
GET /api/v1/ai/predict/maintenance/{installation_id}
GET /api/v1/ai/predict/proactive-service-list
```

Frontend Page: `/dashboard/insights/proactive-service`

Phase 2: Advanced AI Features (1-2 months each)

2.1 Vendor Auto-Negotiation Bot

- Analyze market prices
- Suggest negotiation points
- Auto-generate counter-offer emails

2.2 Dynamic Pricing Engine

- Real-time price optimization
- Competitor price monitoring
- Demand-based pricing

2.3 Fraud Detection System

- Unusual transaction patterns
- Employee expense anomalies
- Fake invoice detection

2.4 Automated Report Generation

- Natural language report requests
- Auto-generated insights
- Scheduled AI summaries

2.5 Voice-Enabled Operations

- Voice commands for warehouse
- Voice notes for service technicians
- Speech-to-text for call logs

Technical Architecture

AI Services Layer

```
┌─────────────────────────────────────────────────────────────┐
│                   FRONTEND (Next.js)                          │
│  ┌─────────────┐ ┌─────────────┐ ┌──────────────────────────┐│
│  │ AI Dashboard│ │ Chat Widget │ │ Prediction Components    ││
│  └─────────────┘ └─────────────┘ └──────────────────────────┘│
└─────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌─────────────────────────────────────────────────────────────┐
│                   API GATEWAY                                 │
│                /api/v1/ai/*                                 │
└─────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌─────────────────────────────────────────────────────────────┐
│                AI SERVICES LAYER                            │
│  ┌──────────────┐ ┌──────────────┐ ┌──────────────────────┐│
│  │   Demand     │ │   Payment    │ │    Predictive        ││
│  │  Forecasting │ │  Prediction  │ │   Maintenance        ││
│  └──────────────┘ └──────────────┘ └──────────────────────┘│
│  ┌──────────────┐ ┌──────────────┐ ┌──────────────────────┐│
│  │   Invoice    │ │   Chatbot    │ │    Allocation        ││
│  │     OCR      │ │   Service    │ │      Engine          ││
│  └──────────────┘ └──────────────┘ └──────────────────────┘│
└────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌────────────────────────────────────────────────────────────┐
│                  ML INFRASTRUCTURE                         │
│  ┌──────────────┐ ┌──────────────┐ ┌──────────────────────┐│
│  │   MLflow     │ │   Model      │ │    Feature           ││
│  │  (Tracking)  │ │   Registry   │ │      Store           ││
│  └──────────────┘ └──────────────┘ └──────────────────────┘│
└────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌────────────────────────────────────────────────────────────┐
│                    DATA LAYER                              │
│  ┌──────────────┐ ┌──────────────┐ ┌──────────────────────┐│
│  │  PostgreSQL  │ │    Redis     │ │   Vector DB          ││
│  │   (Supabase) │ │   (Cache)    │ │  (Embeddings)        ││
│  └──────────────┘ └──────────────┘ └──────────────────────┘│
└────────────────────────────────────────────────────────────┘
```

Required Libraries

```
# requirements-ai.txt

# Core ML
scikit-learn>=1.3.0
pandas>=2.0.0
numpy>=1.24.0

# Time Series
prophet>=1.1.4

# Deep Learning (optional)
torch>=2.0.0
transformers>=4.30.0

# NLP
spacy>=3.6.0
langchain>=0.1.0

# LLM Integration
anthropic>=0.18.0  # Claude API
openai>=1.0.0      # GPT API
```

```
# OCR
pytesseract>=0.3.10
pdf2image>=1.16.0

# ML Ops
mlflow>=2.8.0

# Vector Search
chromadb>=0.4.0
```

Implementation Priority Matrix

Feature	Impact	Effort	Priority
Demand Forecasting	High	Medium	⭐⭐⭐⭐⭐
AI Chatbot	High	Medium	⭐⭐⭐⭐⭐
Invoice OCR	High	Low	⭐⭐⭐⭐⭐
Payment Prediction	Medium	Low	⭐⭐⭐⭐
Predictive Maintenance	High	Medium	⭐⭐⭐⭐
Smart Allocation	Medium	Medium	⭐⭐⭐
Fraud Detection	Medium	High	⭐⭐⭐
Voice Commands	Low	High	⭐⭐

Marketing Claims (Legitimate)

With these features implemented, you can legitimately claim:

Taglines:

- **"AI-Powered ERP for Consumer Durables"**
- **"Intelligent Automation for Your Business"**
- **"Predict. Automate. Grow."**

Feature Claims:

1. ✅ "AI-powered demand forecasting reduces stockouts by 40%"
2. ✅ "Intelligent invoice processing saves 10 hours/week"
3. ✅ "Predictive analytics for cash flow management"
4. ✅ "Smart allocation engine optimizes logistics costs"
5. ✅ "Proactive maintenance alerts increase customer satisfaction"
6. ✅ "Natural language queries for instant business insights"

Certifications to Pursue:

- ISO 27001 (Data Security)
- SOC 2 Type II (for enterprise clients)

- AI/ML Best Practices documentation

Quick Start: First AI Feature

To start immediately, here's the simplest high-impact feature:

Smart Reorder Alerts (Enhanced)

```python
# app/services/ai/smart_reorder.py

class SmartReorderService:
    """
    Enhanced reorder suggestions with AI predictions
    """

    async def get_reorder_suggestions(self) -> list:
        products = await self.get_products_below_reorder_level()

        suggestions = []
        for product in products:
            # Get demand forecast
            forecast = await self.demand_service.forecast(product.id, days=30)

            # Calculate optimal order quantity
            optimal_qty = self.calculate_eoq(
                annual_demand=forecast['annual_demand'],
                ordering_cost=product.ordering_cost or 500,
                holding_cost_rate=0.25
            )

            suggestions.append({
                'product': product,
                'current_stock': product.available_quantity,
                'forecasted_demand_30d': forecast['total_qty'],
                'days_of_stock_remaining': forecast['days_until_stockout'],
                'suggested_order_qty': optimal_qty,
                'urgency': self.calculate_urgency(forecast),
                'confidence': forecast['confidence']
            })

        return sorted(suggestions, key=lambda x: x['days_of_stock_remaining'])
```

Next Steps

1. **Choose first 3 features** from Phase 1 to implement
2. **Set up AI infrastructure** (MLflow for tracking, model storage)
3. **Create AI endpoints** in FastAPI
4. **Build frontend components** for AI insights
5. **Train initial models** on existing data

6. **Deploy and monitor** performance

Document Version: 1.0 Created: January 2026 For: Aquapurite ERP Development Team