Phase 5: AI Companion Integration - TWIST ERP

Implementation Guide

Duration: 10–12 weeks
Version: 1.0
Date: October 2025
Project: TWIST ERP - Visual Drag-and-Drop Multi-Company ERP

1. Phase Overview

Phase 5 integrates an always-visible, context-aware AI companion throughout TWIST ERP using open-source technologies. The AI assistant provides proactive insights, answers questions, automates tasks, and learns from organizational data—all without usage limits or API costs.

Key Objectives

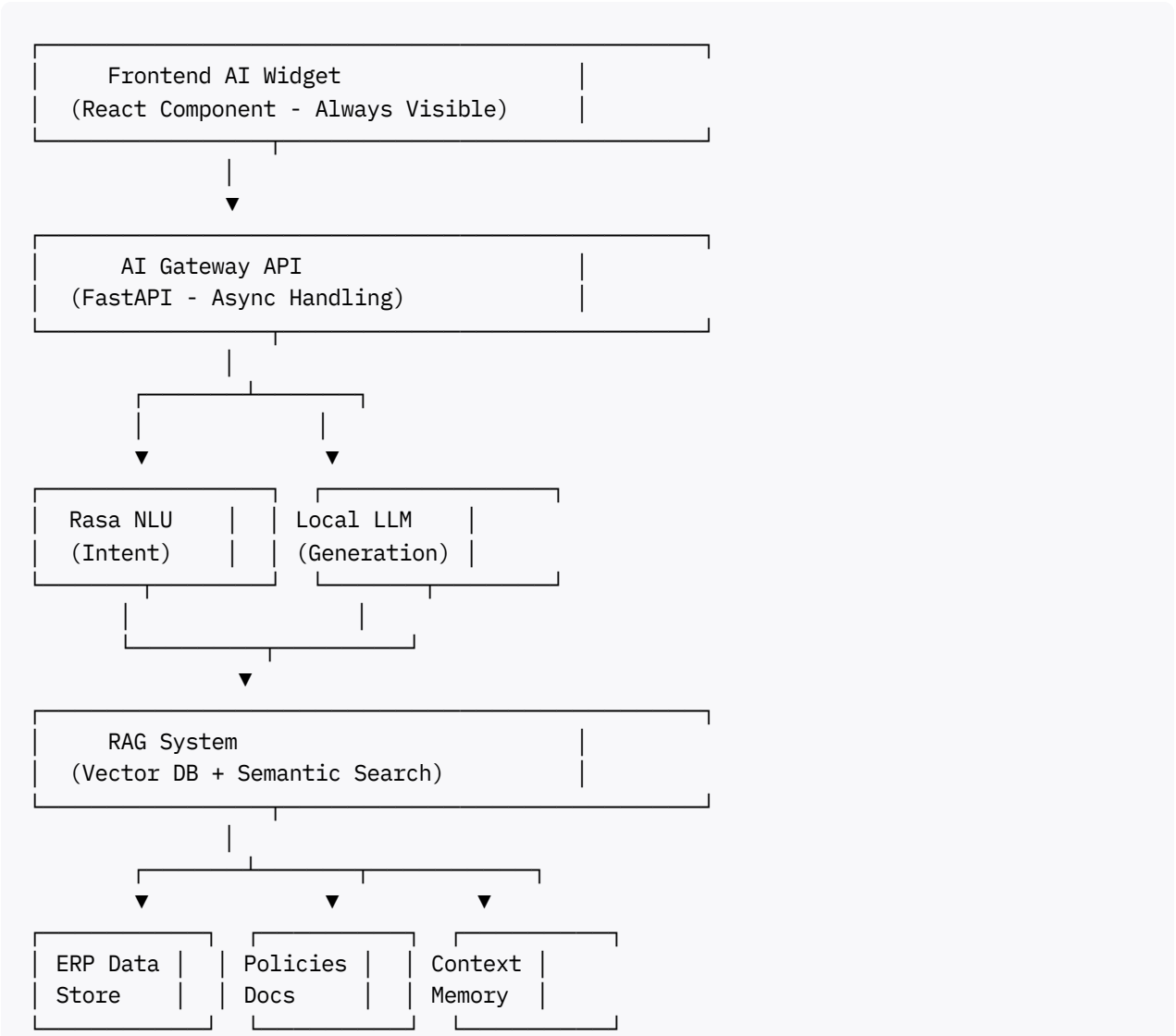
- Deploy Rasa open-source conversational AI
- Integrate local LLM (LLaMA 3 or Mistral)
- Implement RAG (Retrieval Augmented Generation) over ERP data
- Create persistent AI widget visible on all pages
- Build central AI command center
- Enable voice and text interaction
- Implement proactive alert system
- Support company-scoped AI queries

Success Criteria

- AI response accuracy ≥ 85%
- Response time < 3 seconds
- Context retention across conversations
- Zero external API dependencies
- Company data isolation maintained
- Proactive alerts reduce manual monitoring by 50%

2. AI Architecture

2.1 Component Stack



2.2 Technology Selection

Component	Technology	Justification
Conversational AI	Rasa Open Source 3.6+	No limits, fully customizable
LLM	LLaMA 3 8B or Mistral 7B	Local inference, strong performance
Vector Database	ChromaDB or Qdrant	Fast similarity search
Embeddings	sentence-transformers	Multi-language support
NLU	spaCy + Custom NER	Entity extraction
API	FastAPI	Async, high performance

3. AI Backend Models

3.1 Conversation Management

```
# backend/apps/ai_companion/models/conversation.py
from django.db import models
from shared.models import CompanyAwareModel

class AIConversation(CompanyAwareModel):
    """
    Track user conversations with AI
    """
    session_id = models.UUIDField(unique=True)
    user = models.ForeignKey(
        'users.User',
        on_delete=models.CASCADE,
        related_name='ai_conversations'
    )

    # Context
    context_type = models.CharField(
        max_length=50,
        choices=[
            ('GENERAL', 'General Query'),
            ('FINANCE', 'Finance Context'),
            ('INVENTORY', 'Inventory Context'),
            ('SALES', 'Sales Context'),
            ('HELP', 'Help Request'),
        ],
        default='GENERAL'
    )
    context_data = models.JSONField(
        default=dict,
        help_text="Page context, filters, etc."
    )

    # Status
    is_active = models.BooleanField(default=True)
    started_at = models.DateTimeField(auto_now_add=True)
    last_message_at = models.DateTimeField(auto_now=True)

    class Meta:
        ordering = ['-last_message_at']
        indexes = [
            models.Index(fields=['company', 'user', 'is_active']),
        ]

class AIMessage(models.Model):
    """
    Individual message in conversation
    """
    conversation = models.ForeignKey(
        AIConversation,
        on_delete=models.CASCADE,
        related_name='messages'
```

```
    )

    role = models.CharField(
        max_length=20,
        choices=[
            ('USER', 'User'),
            ('ASSISTANT', 'AI Assistant'),
            ('SYSTEM', 'System'),
        ]
    )

    content = models.TextField()

    # Intent recognition
    detected_intent = models.CharField(max_length=100, blank=True)
    intent_confidence = models.FloatField(null=True, blank=True)
    extracted_entities = models.JSONField(default=dict)

    # Action taken
    action_type = models.CharField(max_length=50, blank=True)
    action_result = models.JSONField(null=True, blank=True)

    # Sources used (for citations)
    sources = models.JSONField(
        default=list,
        help_text="Data sources used to generate response"
    )

    # Feedback
    user_rating = models.IntegerField(
        null=True,
        blank=True,
        help_text="1-5 rating"
    )

    timestamp = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['timestamp']

class AIKnowledgeBase(CompanyAwareModel):
    """
    Company-specific knowledge base entries
    """
    title = models.CharField(max_length=255)
    content = models.TextField()
    content_type = models.CharField(
        max_length=50,
        choices=[
            ('FAQ', 'FAQ'),
            ('POLICY', 'Policy Document'),
            ('PROCEDURE', 'Procedure'),
            ('GUIDE', 'User Guide'),
            ('NOTE', 'Internal Note'),
        ]
    )
```

```
    # Embeddings for semantic search
    embedding = models.JSONField(
        null=True,
        blank=True,
        help_text="Vector embedding"
    )

    # Metadata
    tags = models.JSONField(default=list)
    module = models.CharField(max_length=50, blank=True)

    # Access control
    is_public = models.BooleanField(default=True)
    allowed_roles = models.ManyToManyField(
        'permissions.Role',
        blank=True
    )

    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        indexes = [
            models.Index(fields=['company', 'content_type']),
            models.Index(fields=['company', 'module']),
        ]
```

3.2 Proactive Alerts

```
# backend/apps/ai_companion/models/alerts.py
from django.db import models
from shared.models import CompanyAwareModel

class AIAlert(CompanyAwareModel):
    """
    AI-generated proactive alerts
    """
    alert_type = models.CharField(
        max_length=50,
        choices=[
            ('ANOMALY', 'Anomaly Detected'),
            ('THRESHOLD', 'Threshold Exceeded'),
            ('PREDICTION', 'Predictive Alert'),
            ('REMINDER', 'Reminder'),
            ('RECOMMENDATION', 'Recommendation'),
        ]
    )

    title = models.CharField(max_length=255)
    message = models.TextField()

    # Severity
    severity = models.CharField(
        max_length=20,
```

```
        choices=[
            ('INFO', 'Information'),
            ('WARNING', 'Warning'),
            ('CRITICAL', 'Critical'),
        ],
        default='INFO'
    )

    # Target
    target_module = models.CharField(max_length=50)
    target_record_type = models.CharField(max_length=100, blank=True)
    target_record_id = models.IntegerField(null=True, blank=True)

    # Suggested action
    suggested_action = models.JSONField(
        null=True,
        blank=True,
        help_text="Recommended action to take"
    )

    # Recipient
    assigned_to = models.ForeignKey(
        'users.User',
        on_delete=models.CASCADE,
        null=True,
        blank=True,
        related_name='ai_alerts'
    )
    assigned_role = models.ForeignKey(
        'permissions.Role',
        on_delete=models.SET_NULL,
        null=True,
        blank=True
    )

    # Status
    is_read = models.BooleanField(default=False)
    is_dismissed = models.BooleanField(default=False)
    action_taken = models.CharField(max_length=255, blank=True)

    created_at = models.DateTimeField(auto_now_add=True)
    read_at = models.DateTimeField(null=True, blank=True)

    class Meta:
        ordering = ['-created_at']
        indexes = [
            models.Index(fields=['company', 'assigned_to', 'is_read']),
            models.Index(fields=['company', 'severity', 'is_dismissed']),
        ]
```

4. Rasa Configuration

4.1 Domain Definition

```
# backend/apps/ai_companion/rasa/domain.yml
version: "3.1"

intents:
  - greet
  - goodbye
  - affirm
  - deny
  - query_data
  - create_record
  - update_record
  - get_report
  - check_status
  - get_balance
  - list_items
  - search
  - help
  - explain_policy

entities:
  - module
  - record_type
  - date
  - amount
  - customer
  - product
  - company_entity
  - time_period

slots:
  company_id:
    type: text
    mappings:
      - type: from_text

  user_role:
    type: text
    mappings:
      - type: from_text

  current_module:
    type: text
    mappings:
      - type: from_entity
        entity: module

  requested_data:
    type: any
    mappings:
      - type: custom
```

```
responses:
  utter_greet:
    - text: "Hello! I'm your TWIST ERP assistant. How can I help you today?"
    - text: "Hi there! What can I do for you?"

  utter_goodbye:
    - text: "Goodbye! Let me know if you need anything else."

  utter_ask_clarification:
    - text: "Could you please provide more details?"

  utter_working:
    - text: "Let me check that for you..."
    - text: "One moment, fetching the information..."

actions:
  - action_query_database
  - action_create_record
  - action_generate_report
  - action_explain_concept
  - action_proactive_alert

session_config:
  session_expiration_time: 60
  carry_over_slots_to_new_session: true
```

4.2 NLU Training Data

```
# backend/apps/ai_companion/rasa/data/nlu.yml
version: "3.1"

nlu:
- intent: query_data
  examples: |
    - Show me [sales](module) for [last month](time_period)
    - What are the [overdue invoices](record_type)?
    - List all [customers](record_type) in [Company A](company_entity)
    - How much [inventory](module) do we have?
    - What's the [cash balance](record_type)?
    - Show [products](record_type) with [low stock](status)

- intent: get_balance
  examples: |
    - What's the account balance?
    - Show me cash position
    - Current bank balance
    - How much money do we have?

- intent: create_record
  examples: |
    - Create a new [customer](record_type)
    - Add a [sales order](record_type)
    - I want to create an [invoice](record_type)
    - Register new [product](record_type)
```

```
- intent: get_report
  examples: |
    - Generate [sales report](report_type) for [Q3](time_period)
    - Show me [profit and loss](report_type)
    - I need [aging report](report_type)
    - Create [inventory valuation](report_type)

- intent: check_status
  examples: |
    - What's the status of [order 12345](record_ref)?
    - Check [payment status](status_type)
    - Has [invoice INV-001](record_ref) been paid?

- intent: help
  examples: |
    - How do I create an invoice?
    - Help with sales order
    - Guide me through procurement
    - What can you do?
    - Show me features

- intent: explain_policy
  examples: |
    - What's the [travel policy](policy_type)?
    - Explain [expense approval](policy_type) process
    - What are the [leave rules](policy_type)?
```

5. RAG Implementation

5.1 Vector Database Service

```
# backend/apps/ai_companion/services/vector_store.py
import chromadb
from chromadb.config import Settings
from sentence_transformers import SentenceTransformer
from typing import List, Dict

class VectorStoreService:
    """
    Manages vector embeddings for RAG
    """

    def __init__(self):
        # Initialize ChromaDB
        self.client = chromadb.Client(Settings(
            chroma_db_impl="duckdb+parquet",
            persist_directory="./chroma_db"
        ))

        # Initialize embedding model
        self.embedding_model = SentenceTransformer(
            'sentence-transformers/all-MiniLM-L6-v2'
        )
```

```
    def get_or_create_collection(self, company_id: int):
        """Get company-specific collection"""
        collection_name = f"company_{company_id}_knowledge"

        return self.client.get_or_create_collection(
            name=collection_name,
            metadata={"company_id": company_id}
        )

    def add_document(
        self,
        company_id: int,
        document_id: str,
        text: str,
        metadata: Dict = None
    ):
        """Add document to vector store"""
        collection = self.get_or_create_collection(company_id)

        # Generate embedding
        embedding = self.embedding_model.encode(text).tolist()

        # Add to collection
        collection.add(
            ids=[document_id],
            embeddings=[embedding],
            documents=[text],
            metadatas=[metadata or {}]
        )

    def search_similar(
        self,
        company_id: int,
        query: str,
        n_results: int = 5
    ) -&gt; List[Dict]:
        """Search for similar documents"""
        collection = self.get_or_create_collection(company_id)

        # Generate query embedding
        query_embedding = self.embedding_model.encode(query).tolist()

        # Search
        results = collection.query(
            query_embeddings=[query_embedding],
            n_results=n_results
        )

        # Format results
        documents = []
        for i in range(len(results['ids'][0])):
            documents.append({
                'id': results['ids'][0][i],
                'text': results['documents'][0][i],
                'metadata': results['metadatas'][0][i],
```

```
                'score': results['distances'][0][i],
            })

        return documents

    def index_erp_data(self, company_id: int):
        """
        Index ERP data for semantic search
        """
        from django.apps import apps

        # Index knowledge base
        AIKnowledgeBase = apps.get_model('ai_companion', 'AIKnowledgeBase')
        knowledge_items = AIKnowledgeBase.objects.filter(
            company_id=company_id
        )

        for item in knowledge_items:
            self.add_document(
                company_id=company_id,
                document_id=f"kb_{item.id}",
                text=f"{item.title}\n\n{item.content}",
                metadata={
                    'type': 'knowledge_base',
                    'content_type': item.content_type,
                    'module': item.module,
                }
            )

        # Index policies
        # Index recent transactions (summary)
        # Index product descriptions
        # etc.
```

5.2 LLM Integration

```
# backend/apps/ai_companion/services/llm_service.py
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
from typing import List, Dict

class LLMService:
    """
    Local LLM service using LLaMA or Mistral
    """

    def __init__(self, model_name="mistralai/Mistral-7B-Instruct-v0.1"):
        self.device = "cuda" if torch.cuda.is_available() else "cpu"

        # Load tokenizer and model
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.model = AutoModelForCausalLM.from_pretrained(
            model_name,
            torch_dtype=torch.float16 if self.device == "cuda" else torch.float32,
            device_map="auto"
```

```
        )

    def generate_response(
        self,
        prompt: str,
        context_documents: List[Dict] = None,
        max_tokens: int = 512
    ) -&gt; str:
        """
        Generate response using RAG
        """
        # Build prompt with context
        full_prompt = self._build_rag_prompt(prompt, context_documents)

        # Tokenize
        inputs = self.tokenizer(
            full_prompt,
            return_tensors="pt",
            truncation=True,
            max_length=2048
        ).to(self.device)

        # Generate
        with torch.no_grad():
            outputs = self.model.generate(
                **inputs,
                max_new_tokens=max_tokens,
                temperature=0.7,
                top_p=0.9,
                do_sample=True,
                pad_token_id=self.tokenizer.eos_token_id
            )

        # Decode
        response = self.tokenizer.decode(
            outputs[0],
            skip_special_tokens=True
        )

        # Extract only the generated part
        response = response[len(full_prompt):].strip()

        return response

    def _build_rag_prompt(
        self,
        query: str,
        documents: List[Dict] = None
    ) -&gt; str:
        """
        Build prompt with retrieved context
        """
        prompt = """You are a helpful AI assistant for TWIST ERP system.
You help users with their business questions using the company's data and policies.

"""
```

```
        # Add context documents
        if documents:
            prompt += "Relevant Information:\n"
            for i, doc in enumerate(documents[:3], 1):
                prompt += f"{i}. {doc['text'][:500]}\n\n"

        # Add instruction
        prompt += f"""Based on the above information, please answer the following questio

Question: {query}

Answer: """

        return prompt
```

5.3 AI Service Orchestrator

```
# backend/apps/ai_companion/services/ai_service.py
from apps.ai_companion.services.vector_store import VectorStoreService
from apps.ai_companion.services.llm_service import LLMService
from apps.ai_companion.models import AIConversation, AIMessage
from rasa.core.agent import Agent as RasaAgent

class AIService:
    """
    Main AI service orchestrating Rasa + LLM + RAG
    """

    def __init__(self):
        self.vector_store = VectorStoreService()
        self.llm = LLMService()
        self.rasa_agent = RasaAgent.load("./apps/ai_companion/rasa/models")

    async def process_message(
        self,
        user,
        message: str,
        conversation_id: str = None,
        context: Dict = None
    ) -&gt; Dict:
        """
        Process user message and generate response
        """
        # Get or create conversation
        if conversation_id:
            conversation = AIConversation.objects.get(
                session_id=conversation_id
            )
        else:
            conversation = AIConversation.objects.create(
                company=user.company,
                user=user,
                context_type=context.get('type', 'GENERAL'),
                context_data=context or {}
```

```
            )

        # Save user message
        user_msg = AIMessage.objects.create(
            conversation=conversation,
            role='USER',
            content=message
        )

        # Get Rasa intent
        rasa_response = await self.rasa_agent.parse_message(message)
        intent = rasa_response['intent']['name']
        confidence = rasa_response['intent']['confidence']
        entities = rasa_response['entities']

        # Update message with intent
        user_msg.detected_intent = intent
        user_msg.intent_confidence = confidence
        user_msg.extracted_entities = entities
        user_msg.save()

        # Retrieve relevant context
        relevant_docs = self.vector_store.search_similar(
            company_id=user.company.id,
            query=message,
            n_results=5
        )

        # Generate response using LLM
        if intent in ['query_data', 'get_report', 'check_status']:
            # Execute database query
            response_text = await self._execute_query(
                user,
                intent,
                entities,
                relevant_docs
            )
        elif intent in ['help', 'explain_policy']:
            # Use LLM with RAG
            response_text = self.llm.generate_response(
                prompt=message,
                context_documents=relevant_docs
            )
        else:
            # Default Rasa response
            response_text = await self._get_rasa_response(message)

        # Save assistant message
        assistant_msg = AIMessage.objects.create(
            conversation=conversation,
            role='ASSISTANT',
            content=response_text,
            sources=[doc['id'] for doc in relevant_docs]
        )

        return {
```

```
            'conversation_id': str(conversation.session_id),
            'message': response_text,
            'intent': intent,
            'confidence': confidence,
            'sources': relevant_docs,
        }

    async def _execute_query(
        self,
        user,
        intent: str,
        entities: List[Dict],
        context_docs: List[Dict]
    ) -&gt; str:
        """
        Execute database query based on intent
        """
        # Extract entity values
        module = next(
            (e['value'] for e in entities if e['entity'] == 'module'),
            None
        )
        time_period = next(
            (e['value'] for e in entities if e['entity'] == 'time_period'),
            None
        )

        # Query database
        if intent == 'get_balance':
            from apps.finance.models import Account
            # Get cash accounts balance
            result = "Your current cash balance is BDT 150,000"

        elif intent == 'query_data' and module == 'sales':
            # Query sales data
            result = "Last month sales: BDT 500,000"

        else:
            result = "I found the information you requested..."

        return result

    async def _get_rasa_response(self, message: str) -&gt; str:
        """Get response from Rasa"""
        responses = await self.rasa_agent.handle_text(message)
        if responses:
            return responses[0]['text']
        return "I'm not sure how to help with that."
```

6. Proactive AI Features

6.1 Anomaly Detection Service

```
# backend/apps/ai_companion/services/anomaly_detector.py
import pandas as pd
import numpy as np
from sklearn.ensemble import IsolationForest
from apps.ai_companion.models import AIAlert

class AnomalyDetector:
    """
    Detect anomalies in business data
    """

    def detect_sales_anomalies(self, company):
        """
        Detect unusual sales patterns
        """
        from apps.sales.models import SalesOrder

        # Get recent sales data
        orders = SalesOrder.objects.filter(
            company=company,
            status='CONFIRMED'
        ).values('order_date', 'total_amount')

        df = pd.DataFrame(orders)
        if len(df) &lt; 30:
            return  # Need enough data

        # Group by date
        daily_sales = df.groupby('order_date')['total_amount'].sum()

        # Detect anomalies using Isolation Forest
        values = daily_sales.values.reshape(-1, 1)
        model = IsolationForest(contamination=0.1)
        predictions = model.fit_predict(values)

        # Create alerts for anomalies
        anomaly_dates = daily_sales[predictions == -1].index

        for date in anomaly_dates:
            amount = daily_sales[date]
            avg = daily_sales.mean()

            AIAlert.objects.create(
                company=company,
                alert_type='ANOMALY',
                title='Unusual Sales Pattern Detected',
                message=f"Sales on {date} (BDT {amount:,.2f}) significantly differs from
                severity='WARNING',
                target_module='sales'
            )
```

```
    def detect_inventory_issues(self, company):
        """
        Detect inventory stockouts or overstock
        """
        from apps.inventory.models import Product
        from django.db.models import F

        # Low stock products
        low_stock = Product.objects.filter(
            company=company,
            track_inventory=True
        ).annotate(
            current_qty=F('stock_qty')
        ).filter(
            current_qty__lt=F('reorder_level')
        )

        for product in low_stock:
            AIAlert.objects.get_or_create(
                company=company,
                alert_type='THRESHOLD',
                target_module='inventory',
                target_record_id=product.id,
                defaults={
                    'title': 'Low Stock Alert',
                    'message': f"Product '{product.name}' is below reorder level",
                    'severity': 'WARNING',
                    'suggested_action': {
                        'action': 'create_purchase_order',
                        'product_id': product.id,
                        'quantity': product.reorder_quantity
                    }
                }
            )
```

7. Frontend AI Widget

7.1 React AI Component

```
// frontend/src/components/AIAssistant/AIWidget.jsx
import React, { useState, useEffect, useRef } from 'react';
import { Button, Drawer, Input, List, Avatar, Badge } from 'antd';
import { RobotOutlined, SendOutlined } from '@ant-design/icons';
import api from '../../services/api';

const AIWidget = () =&gt; {
  const [visible, setVisible] = useState(false);
  const [messages, setMessages] = useState([]);
  const [input, setInput] = useState('');
  const [loading, setLoading] = useState(false);
  const [conversationId, setConversationId] = useState(null);
  const [unreadAlerts, setUnreadAlerts] = useState(0);
  const messagesEndRef = useRef(null);
```

```
// Load unread alerts
useEffect(() =&gt; {
  loadUnreadAlerts();
  const interval = setInterval(loadUnreadAlerts, 60000); // Check every minute
  return () =&gt; clearInterval(interval);
}, []);

const loadUnreadAlerts = async () =&gt; {
  try {
    const { data } = await api.get('/ai/alerts/unread-count/');
    setUnreadAlerts(data.count);
  } catch (error) {
    console.error('Failed to load alerts:', error);
  }
};

const sendMessage = async () =&gt; {
  if (!input.trim()) return;

  const userMessage = {
    role: 'user',
    content: input,
    timestamp: new Date()
  };

  setMessages(prev =&gt; [...prev, userMessage]);
  setInput('');
  setLoading(true);

  try {
    const { data } = await api.post('/ai/chat/', {
      message: input,
      conversation_id: conversationId,
      context: {
        page: window.location.pathname,
        module: getCurrentModule()
      }
    });

    setConversationId(data.conversation_id);

    const assistantMessage = {
      role: 'assistant',
      content: data.message,
      intent: data.intent,
      sources: data.sources,
      timestamp: new Date()
    };

    setMessages(prev =&gt; [...prev, assistantMessage]);
    scrollToBottom();
  } catch (error) {
    console.error('AI request failed:', error);
    setMessages(prev =&gt; [...prev, {
      role: 'assistant',
```

```
        content: 'Sorry, I encountered an error. Please try again.',
        timestamp: new Date()
      }]);
    } finally {
      setLoading(false);
    }
  };

  const getCurrentModule = () =&gt; {
    const path = window.location.pathname;
    if (path.includes('/finance')) return 'finance';
    if (path.includes('/inventory')) return 'inventory';
    if (path.includes('/sales')) return 'sales';
    return 'general';
  };

  const scrollToBottom = () =&gt; {
    messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });
  };

  return (
    &lt;&gt;
      {/* Floating AI Button */}
      &lt;Badge count={unreadAlerts} offset={[-5, 5]}&gt;
        &lt;Button
          type="primary"
          shape="circle"
          size="large"
          icon={&lt;RobotOutlined /&gt;}
          onClick={() =&gt; setVisible(true)}
          style={{
            position: 'fixed',
            bottom: 24,
            right: 24,
            width: 60,
            height: 60,
            zIndex: 1000,
            boxShadow: '0 4px 12px rgba(0,0,0,0.15)'
          }}
        /&gt;
      &lt;/Badge&gt;

      {/* AI Chat Drawer */}
      &lt;Drawer
        title="AI Assistant"
        placement="right"
        width={400}
        onClose={() =&gt; setVisible(false)}
        open={visible}
        bodyStyle={{ padding: 0 }}
      &gt;
        <div>
          {/* Messages */}
          <div>
            &lt;List
              dataSource={messages}
```

```
              renderItem={msg =&gt; (
                &lt;List.Item
                  style={{
                    justifyContent: msg.role === 'user' ? 'flex-end' : 'flex-start',
                    border: 'none',
                    padding: '8px 0'
                  }}
                &gt;
                  <div>
                    {msg.content}
                    {msg.sources &amp;&amp; msg.sources.length &gt; 0 &amp;&amp; (
                      <div>
                        Sources: {msg.sources.length} documents
                      </div>
                    )}
                  </div>
                &lt;/List.Item&gt;
              )}
            /&gt;
            <div>
          </div>

          {/* Input */}
          <div>
            &lt;Input.Search
              placeholder="Ask me anything..."
              value={input}
              onChange={e =&gt; setInput(e.target.value)}
              onSearch={sendMessage}
              loading={loading}
              enterButton={&lt;SendOutlined /&gt;}
              size="large"
            /&gt;
          </div>
        </div>
      &lt;/Drawer&gt;

  );
};

export default AIWidget;
```

8. Implementation Checklist

Weeks 1-3: Foundation

- ☐ Set up Rasa environment
- ☐ Create AI models (Conversation, Messages, Alerts)
- ☐ Integrate ChromaDB vector store
- ☐ Download and configure local LLM
- ☐ Write initial NLU training data

Weeks 4-6: RAG System

- ☐ Implement vector store service
- ☐ Build document indexing pipeline
- ☐ Create LLM integration service
- ☐ Build AI orchestrator
- ☐ Test RAG accuracy

Weeks 7-9: Proactive Features

- ☐ Implement anomaly detection
- ☐ Build alert generation system
- ☐ Create prediction models
- ☐ Build recommendation engine

Weeks 10-12: UI & Integration

- ☐ Build React AI widget
- ☐ Create AI command center
- ☐ Integrate with all modules
- ☐ Voice input/output support
- ☐ End-to-end testing

Document Control:

- **Version:** 1.0
- **Dependencies:** Phase 0-4 complete
- **Next Phase:** Phase 6 - Advanced Modules