Screen Printing NW Chatbot - Complete Implementation Guide

PROJECT OVERVIEW

You need to build a production-ready chatbot that EXACTLY replicates an existing Voiceflow Screen Printing Assistant. This is not just any chatbot - it must match the original word-for-word in terms of messages, flow, logic, and user experience.

Core Requirements:

- Use Python 3.11+, FastAPI, LangGraph for state machine
- Implement finite state machine (FSM) with explicit states and transitions
- Support session persistence across HTTP requests using Redis
- Include RAG system for FAQ answers using Chroma vector database
- Send confirmation emails for orders
- Handle graceful fallbacks and error recovery
- Support global interrupts (human escalation, end conversation)

TECHNICAL ARCHITECTURE

Stack:

- FastAPI (web framework and API endpoints)
- LangGraph (state machine orchestration)
- Pydantic (data models and validation)
- OpenAI GPT-4o-mini (for intent classification and FAQ)
- Chroma (vector database for RAG/FAQ system)
- Redis (session state persistence)
- SMTP/SendGrid (email delivery)
- Docker (containerization)

System Flow:

User Input → FastAPI Webhook → Session Manager → LangGraph FSM → Node Processing → Update State → Response

DATA MODELS

SessionState Model

Create a comprehensive Pydantic model with these fields:

- `session_id`: Unique identifier for the conversation
- `state`: Current FSM state (enum with all possible states)
- `contact`: Nested model with first_name, last_name, email, phone
- `order`: Nested model with order_type, budget_range, method, apparel_type, items[], has_logo, logo_url
- `last_utterance`: User's most recent input
- `conversation_history`: List of all messages with roles and timestamps
- `metadata`: Dictionary for temporary state variables and flags
- `retry_count`: Track failed attempts for fallback logic
- `created_at`, `updated_at`: Timestamps

Order Models

- `OrderItem`: size and quantity pairs
- `OrderSpec`: All order attributes and list of OrderItems
- `Contact`: All contact information fields

FINITE STATE MACHINE DESIGN

States (ConversationState Enum)

1. **WELCOME** - Initial greeting, auto-transition to MAIN_MENU
2. **MAIN_MENU** - Classify user intent and route to appropriate flow
3. **FAQ_CLASSIFY** - Determine if user has concrete question
4. **FAQ** - Answer questions using RAG system with follow-up loop
5. **ORDER_CONTACT** - Collect contact info step-by-step (first name → last name → email → phone)
6. **ORDER_ATTRS** - Collect order attributes (type → budget → method → apparel)
7. **SIZE_LOOP** - Collect size/quantity pairs with "add more" loop
8. **LOGO** - Ask about logo upload with yes/no branch
9. **SUMMARY** - Generate summary, send email, show confirmation
10. **HUMAN** - Provide contact information immediately
11. **END** - Farewell message and terminate
12. **FALLBACK** - Handle unrecognized input, retry logic

Transitions and Routing Logic

Each state should have conditional routing based on:

- Intent classification results
- Data validation status
- User yes/no responses
- Global interrupts (human/end from any state)
- Error conditions and retry counts

NODE IMPLEMENTATIONS

Welcome Node

- Display exact greeting message
- Automatically transition to MAIN_MENU
- Reset retry counter

Main Menu Classification Node

- Use LLM to classify user intent into: PlaceOrder, HasProductQuestion, WantsHuman, EndConversation
- Include confidence scoring - if below threshold, use fallback
- Store classification results in metadata
- Route based on classified intent

FAQ System Nodes

FAQ_CLASSIFY Node:

- Use LLM to determine if user has concrete, answerable question
- If not concrete, ask user to be more specific
- If concrete, proceed to FAQ agent

FAQ_AGENT Node:

- Use RAG system to search knowledge base
- Generate answer with confidence scoring
- If confident answer found: provide answer + ask for follow-up
- If not confident: apologize + offer human escalation
- Support follow-up question loop

Order Flow Nodes

ORDER_CONTACT Node:

- Sequential collection: first name → last name → email → phone
- Validate each field before proceeding to next
- Email validation using regex/email-validator library
- Phone validation using phonenumbers library
- Retry invalid inputs with helpful error messages

ORDER_ATTRS Node:

- Sequential collection: order type → budget → method → apparel type
- Use LLM/keyword classification for each attribute
- Validate responses and retry if unclear
- Store validated attributes in order model

SIZE_LOOP Node:

- Collect size → quantity pair
- Add to order items list
- Ask "Do you want to add another size?"
- Loop if yes, proceed to logo if no
- Validate size options (XS, S, M, L, XL, XXL, XXXL)
- Validate quantity as positive integer

LOGO Node:

- Ask yes/no about logo upload
- If yes: provide upload instructions and link
- If no: acknowledge and proceed
- Store boolean flag in order

SUMMARY Node:

- Generate formatted order summary
- Send confirmation email to customer
- Send notification email to business
- Display success message with next steps

Support Nodes

HUMAN Node:

- Immediately provide phone number, email, and business hours
- "Phone: 425.303.3381, Email: info@screenprintingnw.com, Hours: Mon-Fri 8AM-5PM"
- End conversation after providing info

END Node:

- Display goodbye message: "It was nice talking to you. I hope to see you again soon. Goodbye!"
- Mark conversation as ended
- Clean up session if needed

FALLBACK Node:

- Display fallback message asking for clarification
- Increment retry counter
- After 3 retries, offer human escalation
- Route back to MAIN_MENU for new attempt

RAG/FAQ SYSTEM SPECIFICATION

Knowledge Base Setup

- Ingest documents from provided PDFs/markdown files
- Split into 800-1200 token chunks with 100 token overlap
- Use OpenAI text-embedding-3-large for embeddings
- Store in Chroma persistent collection

Answer Generation Process

1. Generate embedding for user question
2. Search vector DB for top 5 most similar chunks
3. Filter results below similarity threshold (0.8)
4. Use retrieved chunks as context for LLM answer generation
5. Calculate confidence score based on similarity and answer quality
6. Return empty response if confidence below 0.7

Answer Quality Controls

- System prompt: Only use provided context, don't fabricate
- If uncertain, return empty string to trigger "not found" flow
- Include source attribution when possible
- Provide contact info when answer not found

EMAIL SYSTEM SPECIFICATION

Customer Confirmation Email

Subject: "Your Screen Printing Order Details - Screen Printing NW"

Content Template:

- Professional HTML template with company branding
- Order summary with all collected information
- Contact details, order attributes, items list, logo status
- Next steps: "Our team will review within 24 hours"
- Logo upload instructions if applicable
- Contact information for questions

Business Notification Email

Subject: "New Order from [Customer Name]"

Content:

- All order details for internal processing
- Customer contact information
- Order specifications and requirements
- Timestamp and session details

Email Configuration

- Support both SMTP and transactional email services (SendGrid/Mailgun)
- HTML email with plain text fallback
- Error handling with graceful degradation
- Retry logic for failed sends

INTENT CLASSIFICATION SYSTEM

Primary Intents

- **PlaceOrder**: Keywords like "order", "quote", "print", "shirts"
- **HasProductQuestion**: "question", "how much", "what", "can you"
- **WantsHuman**: "human", "person", "representative", "call"
- **EndConversation**: "bye", "goodbye", "quit", "done", "exit"
- **Yes/No**: For confirmation dialogs in flows

Classification Strategy

- Use hybrid approach: keyword patterns + LLM classification
- LLM system prompt: Return exactly one intent from predefined list
- Include confidence scoring
- Fallback to keyword matching if LLM fails
- Context-aware classification based on current state

Global Interrupts

- "WantsHuman" and "EndConversation" should work from ANY state
- Override current flow and route immediately
- Preserve context for potential return (human escalation)
- Clean up partial data appropriately

VALIDATION AND ERROR HANDLING

Input Validation

- **Email**: RFC-compliant regex or email-validator library
- **Phone**: Use phonenumbers library for international support
- **Names**: Length limits, basic character validation
- **Sizes**: Strict enum matching (XS, S, M, L, XL, XXL, XXXL)
- **Quantities**: Positive integers only

Error Recovery Strategies

- Polite error messages explaining what's expected
- Provide examples of valid input formats

- Maximum retry attempts before escalation
- Never leave user in dead-end state

Resilience Features

- Graceful degradation when external services fail
- Retry logic with exponential backoff
- Circuit breaker pattern for failing services
- Comprehensive error logging

SESSION MANAGEMENT

State Persistence

- Store complete SessionState in Redis with TTL
- Session key: `chatbot:session:{session_id}`
- Serialize using Pydantic JSON export
- Handle Redis connection failures gracefully

Session Lifecycle

- Create new session on first interaction
- Update state after each node execution
- Extend TTL on each interaction
- Clean up expired sessions
- Support session recovery after interruption

Re-entrancy Safety

- All operations should be idempotent
- Handle concurrent access to same session
- Atomic updates to session state
- Prevent race conditions in multi-step flows

API DESIGN

Chat Endpoint: POST /chat

Request:

```
{
   "session_id": "uuid4-string",
   "message": "user input text",
   "metadata": {}
}
```

Response:

```
{
   "messages": [
      {
         "role": "assistant",
         "content": "bot response",
         "timestamp": "ISO datetime"
      }
   ],
   "session_id": "uuid4-string",
   "state": "current_state_name",
   "needs_input": true,
   "conversation_ended": false,
   "metadata": {}
}
```

Health Check: GET /health

- Return service status
- Check database connections
- Verify external service availability

Optional File Upload: POST /upload/logo

- Accept multipart file uploads
- Validate file types (JPG, PNG, PDF, AI)
- Return upload URL for order association
- Alternative: provide Google Form link

EXACT PROMPTS AND MESSAGES

User-Facing Messages (Must Match Original)

- **Welcome**: "Hello! Welcome to Screen Printing NW. How can I help you today?"
- **Fallback**: "Sorry, I didn't get that. Do you want to place an order, ask a product question, talk to a human, or end the conversation?"
- **Human Contact**: "You can reach us at phone 425.303.3381, email info@screenprintingnw.com, or visit our website. Our hours are Monday through Friday, 8 AM to 5 PM."
- **Goodbye**: "It was nice talking to you. I hope to see you again soon. Goodbye!"

Order Flow Prompts

- **First Name**: "I'd be happy to help you place an order! Let's start with your first name."
- **Last Name**: "Great! And what's your last name?"

- **Email**: "Perfect! What's the best email address to reach you?"
- **Phone**: "And what's your phone number?"
- **Order Type**: "What type of order is this for? Is it for business, personal use, an event, or a team?"
- **Budget**: "What's your budget range for this project?"
- **Method**: "Would you prefer screen printing or embroidery?"
- **Apparel**: "What type of apparel are you looking to customize?"
- **Size**: "What size would you like?"
- **Quantity**: "How many pieces in that size?"
- **More Sizes**: "Would you like to add another size?"
- **Logo**: "Do you have a logo or design that you'd like us to print?"

LLM System Prompts

- **Intent Classification**: "Return exactly one of: {PlaceOrder, HasProductQuestion, WantsHuman, EndConversation, Yes, No, Fallback}"
- **FAQ Classification**: "Decide TRUE/FALSE if the user's message contains a concrete, answerable question about screen printing, products, or services."
- **FAQ Answer**: "Answer the question using only the provided context. If you cannot answer confidently, respond with an empty string."

TESTING STRATEGY

Unit Tests Required

- Test each node function individually
- Mock external services (OpenAI, email, Redis)
- Validate state transitions
- Test input validation functions
- Test classification helpers

Integration Tests Required

- End-to-end conversation flows
- Database persistence
- Email delivery
- Error recovery scenarios
- Global interrupt handling

Test Scenarios (Happy Paths)

1. **Complete Order Flow**: Welcome → order → contact → attributes → sizes → logo → summary → email
2. **FAQ Flow**: Welcome → question → answer → follow-up → menu
3. **Human Escalation**: From any state → immediate contact info
4. **Graceful Exit**: From any state → goodbye

Test Scenarios (Edge Cases)

- Invalid email/phone formats
- Classification confidence below threshold
- External service failures (OpenAI, email, Redis)
- Session timeout and recovery
- Rapid consecutive requests
- Malformed input handling

DEPLOYMENT SPECIFICATION

Environment Configuration

- Use environment variables for all secrets
- Separate configs for dev/staging/prod
- Docker containerization required
- Health check endpoints for monitoring

Required Environment Variables

OPENAI_API_KEY=sk-...
REDIS_URL=redis://localhost:6379
VECTOR_DB_DIR=/app/data/vector_db
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=your-email
SMTP_PASSWORD=your-app-password
MAIL_FROM=noreply@screenprintingnw.com
BUSINESS_EMAIL=info@screenprintingnw.com

Production Requirements

- HTTPS termination
- Rate limiting (requests per session/IP)
- Request/response logging
- Error tracking and alerting
- Database backup strategy
- Monitoring and metrics

DELIVERABLES CHECKLIST

Code Structure

- FastAPI application with proper project structure
- Pydantic models for all data structures
- LangGraph state machine implementation

- Node implementations for all states
- Service classes for external integrations
- Comprehensive error handling

Documentation

- README with setup instructions
- API documentation (auto-generated by FastAPI)
- Environment setup guide (.env.example)
- Docker deployment guide
- Testing instructions

Testing

- Unit tests for core functions
- Integration tests for full flows
- Postman collection for API testing
- Test data and scenarios

Deployment Assets

- Dockerfile and docker-compose.yml
- Environment configuration templates
- Health check endpoints
- Logging configuration

DEVELOPMENT WORKFLOW

Phase 1: Core Infrastructure (Days 1-2)

- Project setup and dependencies
- Data models and session management
- Basic FastAPI app with health checks
- Redis integration and session persistence

Phase 2: State Machine Framework (Days 2-3)

- LangGraph setup and basic FSM
- Node structure and routing logic
- Welcome, main menu, and fallback nodes
- Intent classification system

Phase 3: Order Flow (Days 3-4)

- Contact collection nodes
- Order attributes collection
- Size loop implementation

- Validation and error handling

Phase 4: FAQ System (Days 4-5)

- RAG service implementation
- Knowledge base ingestion
- FAQ classification and answering
- Confidence scoring and fallbacks

Phase 5: Email and Polish (Days 5-6)

- Email service implementation
- Template creation and rendering
- Order summary generation
- End-to-end testing

Phase 6: Testing and Deployment (Day 6-7)

- Comprehensive testing
- Documentation completion
- Docker setup and deployment
- Performance optimization

SUCCESS CRITERIA

Functional Requirements

- ✅ Exact message matching with original Voiceflow bot
- ✅ All conversation flows work end-to-end
- ✅ Session persistence across requests
- ✅ Email delivery and confirmation
- ✅ FAQ system with proper fallbacks
- ✅ Global interrupts from any state

Technical Requirements

- ✅ Production-ready error handling
- ✅ Proper logging and monitoring
- ✅ Input validation and security
- ✅ Docker deployment ready
- ✅ API documentation complete
- ✅ Test coverage >80%

Performance Requirements

- Response time <2 seconds for simple queries
- FAQ search <3 seconds

- Email delivery <10 seconds
- Support 100+ concurrent sessions
- Session data persistence for 24+ hours

This comprehensive specification provides everything needed to build the exact chatbot system you need. Each section contains the precise requirements, constraints, and implementation details necessary for a developer to create a production-ready application that exactly matches your existing Voiceflow assistant.