**Use Case: Customer Support in E-Commerce**

**Problem: High Cost and Slow Response in Customer Service**

E-commerce companies often struggle with high volumes of repetitive customer inquiries (e.g., order tracking, return policies, delivery delays). Human agents get overwhelmed, leading to:

* Long response times (avg. >12 hours)
* High support staffing costs
* Low customer satisfaction (CSAT scores below 70%)

**Proposed AI Agent: Smart Support Concierge**

An **AI-powered customer support agent** embedded in the company's website and mobile app that:

* Answers FAQs instantly via natural language
* Checks order status via API integration with backend systems
* Guides users through returns and refunds
* Escalates complex queries to human agents with full conversation history

**Interaction Channels**: Chat widget (web & mobile), optional voice interface

**Measurable Impact**

1. **⏱ Time Saved**: Average response time reduced from 12 hours to under 30 seconds
2. **💸 Cost Savings**: 40–60% reduction in support center costs by deflecting up to 80% of Tier 1 queries
3. **😊 Improved CSAT**: Customer satisfaction increases from 68% to 85% due to faster resolution and 24/7 availability

**Bonus: Interaction Sketch**

* Users open the chat widget labeled “Ask about your order”
* They type: *“Where is my package?”*
* The agent replies: *“Your order #12345 is currently out for delivery and expected by 6 PM today.”*
* Buttons below allow follow-up: “Track Live”, “Report Issue”, “Return Item”

Here's a structured Lovable.dev prompt using the **C.L.E.A.R.** principles to build the Smart Support Concierge AI agent:

**📦 AI Customer Support Agent – Prompt for Lovable.dev**

**Context**

You are an AI developer building a customer support chatbot for an e-commerce website using Lovable.dev. The chatbot must handle common support tasks and integrate with backend systems.

**Task**

Create an AI-powered chat interface embedded on the homepage. The chatbot should:

* Answer FAQs (shipping, returns, refund policies)
* Provide real-time order tracking via Supabase API
* Escalate unresolved issues to human agents with conversation history
* Support follow-up actions through quick-reply buttons

**Guidelines**

* Use React and Tailwind CSS for the chat UI
* Connect to a Supabase database to retrieve order information (e.g., order ID, status, delivery estimate)
* Use OpenAI or Anthropic for conversational handling
* Pre-train with FAQs and escalation triggers
* Keep the chat window lightweight and mobile-responsive

**Constraints**

* Do not store any sensitive user information in the frontend
* Must handle API errors gracefully
* Must respond in <2 seconds for real-time queries
* The chatbot UI must not cover any critical UI elements on mobile

Here's how we can break this into **incremental Lovable.dev prompts** for a clean, step-by-step implementation:

**🧩 Step 1: Build the Chat UI**

**Prompt**:

sql

Create a floating chat widget for the homepage using React and Tailwind CSS. The chat bubble should be fixed to the bottom right, expandable into a panel with:

- A header titled “Smart Support”

- A message window with scroll

- An input box with a send button

- Support for displaying bot and user messages

Style it with a minimal, mobile-friendly look. Ensure it doesn’t overlap with core page content.

**🔌 Step 2: Add Supabase Integration for Order Tracking**

**Prompt**:

pgsql

Connect the chat interface to a Supabase backend with an "orders" table. The table includes: order\_id, user\_email, status, expected\_delivery.

When a user types “Where is my order?” followed by their order ID or email, query Supabase and return the order status and delivery estimate.

Handle API errors and show a fallback message like: “Sorry, I couldn’t find your order.”

**🧠 Step 3: Add AI Logic for FAQ + Escalation**

**Prompt**:

pgsql

Integrate OpenAI to handle customer messages that are not related to orders. Include a predefined set of FAQs (returns, refunds, shipping times).

If the user types a message the model can’t confidently answer, respond with: “I’ll connect you to a human agent now,” and trigger an `escalateToHuman()` function (placeholder).

Maintain a message history array so the full conversation can be sent during escalation.

**🎛️ Step 4: Add Quick Reply Buttons**

**Prompt**:

sql

Enhance the chat UI with quick-reply buttons after certain messages. For example:

- After order status: [Track Live] [Report Issue]

- After refund info: [Start Return] [Talk to Human]

Handle button clicks with matching function calls and update the conversation view accord