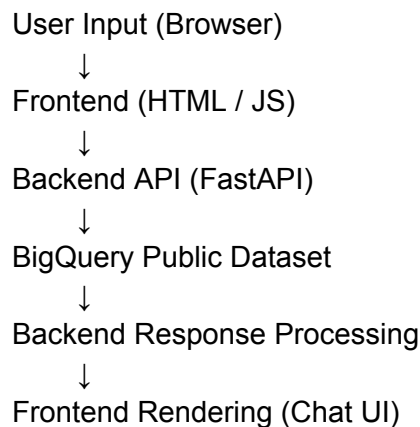


# Data Flow — E-commerce Chatbot

## 1. Overview

This document describes how data moves through the E-commerce Chatbot system, from user input to rendered product results. The system follows a **request-response flow** with clear separation between presentation, logic, and data layers.

## 2. High-Level Data Flow



## 3. Frontend → Backend Flow

### Step 1: User Input

- User types a free-text message (e.g., “Show me winter jackets under \$100”)
- Message is captured from the chat input field

### Step 2: Request Dispatch

Frontend sends an HTTP request:

GET /chat?message=<encoded text>

### Payload

- Natural language text only
- No session or user ID required

## 4. Backend Data Processing Flow

### Step 3: Message Normalization

- Convert text to lowercase
- Remove punctuation where necessary
- Prepare for pattern matching

### Step 4: Constraint Extraction

Structured data is derived from text:

Constraint	Source
Price	\$, under, over keywords
Department	gift recipient keywords
Size	small / medium / large
Category	product type keywords

Extracted constraints are stored in memory for the request lifecycle.

## 5. Query Construction

## Step 5: Dynamic SQL Assembly

Base query:

```
SELECT name, category, brand, retail_price
```

```
FROM products
```

```
WHERE 1=1
```

Optional filters appended:

- Price condition
- Department
- Size (via name matching)
- Category keyword

All values are passed as **parameterized inputs**.

## 6. Backend → BigQuery Flow

### Step 6: Query Execution

- FastAPI executes the SQL query using the BigQuery client
- BigQuery scans the public dataset
- Results returned as rows

### Step 7: Result Mapping

- Rows converted into JSON-serializable objects
- Optional enrichment applied (mock stock, sizes)

## 7. Backend Response Flow

### Step 8: Response Decision

#### **If results found**

```
{  
  "reply": "Here are jackets under $100.",  
  "products": [...]  
}
```

#### **If no results**

```
{  
  "reply": "Sorry, I couldn't find any matching items.",  
  "quick_replies": ["Increase budget", "Show similar items"]  
}
```

## **8. Backend → Frontend Flow**

### **Step 9: JSON Response**

- Backend returns structured JSON
- HTTP 200 response

## **9. Frontend Rendering Flow**

### **Step 10: Message Rendering**

- Bot reply shown as chat bubble
- Products rendered as horizontal carousel
- Quick replies rendered as buttons

### **Step 11: User Interaction**

- Clicking a quick reply sends a new message
- Carousel scroll preserves chat context

## 10. Error & Empty State Flow

Scenario	Handling
No results	Friendly explanation + suggestions
Invalid query	Default product search
Backend error	Silent failure with fallback reply

## 11. Why This Data Flow

- Stateless backend = easy scaling
- Minimal payloads = fast responses
- Conversational UI = reduced cognitive load
- Data-driven responses remain explainable

## 12. Future Data Flow Extensions

- Redis cache for popular queries
- User context persistence
- LLM-based semantic parsing
- Recommendation embeddings
- Event tracking for analytics