# Jules System Architecture & How It Works

## Overview

Jules is an AI-powered fashion and dating advisor that acts like a confident, stylish older sister who helps men with style, dating, and life advice. The system combines natural language processing, product search capabilities, conversational AI, and comprehensive logging to provide personalized recommendations and track system performance.

## System Architecture

### Frontend (React/Next.js)

* **Location**: jules-frontend/
* **Port**: 3000
* **Technology**: Next.js with TypeScript
* **Purpose**: User interface for chatting with Jules

### Backend (Node.js/Express)

* **Location**: jules-backend/
* **Port**: 4000
* **Technology**: Express.js with MongoDB
* **Purpose**: Handles all business logic, AI processing, data storage, and logging

### Database (MongoDB)

* **Purpose**: Stores user data, conversation history, user preferences, and system logs
* **Models**: User, Conversation, ChatLog

## How It Works - Step by Step

### 1. User Sends a Message

User types: "I need help finding a leather jacket"

### 2. Frontend Processing

* Frontend captures the message
* Sends it to backend via API call: POST /api/chat
* Includes user authentication token and user ID

### 3. Backend Receives Request

* **Route**: /api/chat → chatController.handleChat()
* **Authentication**: Verifies user token
* **Validation**: Checks for required fields (message, userId)

### 4. Intent Classification & Context Analysis

The system analyzes the message to determine:

**Gender Detection:** - Scans message for gender indicators (“I’m a man”, “male”, “guy”) - Updates user preferences accordingly - Defaults to male if no gender specified

**Message Type Classification:** - **Image Request**: “show me pics”, “what does it look like” - **Product Request**: “show me leather jackets”, “find me shoes” - **Link Request**: “show me links”, “where can I buy” - **Advice Request**: “dating advice”, “ghosting situation” - **Simple Question**: “hi”, “thanks”, “bye”

### 5. Conversation Context Management

* Retrieves user’s conversation history from MongoDB
* Maintains context across multiple messages
* Uses last 10 messages for context
* Ensures natural conversation flow

### 6. AI Processing (OpenAI GPT-3.5-turbo)

**System Prompt Generation:** - Creates gender-specific personality prompt - Includes conversation rules and constraints - Sets tone: confident, direct, flirty, helpful

**Message Processing:** - Sends conversation history + current message to OpenAI - Uses dynamic token limits based on message type: - Simple questions: 2000 tokens - Advice questions: 3000 tokens - Product requests: 2000 tokens - Long conversations: 3000 tokens

### 7. Response Generation

**AI generates response with these rules:** - ✅ Natural, conversational tone - ✅ Gender-appropriate advice (men’s fashion only) - ✅ Specific, actionable suggestions - ✅ Follow-up questions for context - ❌ No AI-style closers (“let me know how I can help”) - ❌ No numbered lists or bullet points - ❌ No motivational language (“You got this!”)

### 8. Product Search (When Applicable)

**Triggers for product search:** - User asks for specific clothing items - User asks for shopping links - User mentions brands (Nike, Lululemon, etc.)

**Search Process:** 1. Extracts brand and product keywords from message 2. Generates search query: “Nike men’s sneakers buy shop” 3. Uses Google Custom Search API 4. Filters results: - ✅ Shopping/product sites only - ❌ Excludes women’s clothing - ❌ Excludes social media/blog sites 5. Returns up to 3 product cards with: - Title, link, image, price, description

### 9. Response Processing

**Post-processing steps:** 1. **Strip Closers**: Removes AI-style endings 2. **Clean Formatting**: Removes numbered lists 3. **Extract Product Links**: Converts markdown links to structured data 4. **Final Response**: Returns clean, natural response

### 10. Logging & Analytics

**Every interaction is logged for analysis:** - **ChatLog Model**: Stores user messages, AI responses, and intent classification - **Intent Detection**: Automatically categorizes each interaction: - dating\_advice: Relationship, ghosting, dating questions - style\_advice: Fashion, clothing, outfit questions - product\_request: Shopping, product search requests - image\_request: Visual content requests - greeting: Simple greetings and acknowledgments - general\_conversation: Other interactions - **Error Logging**: Captures failed requests and system errors - **Performance Tracking**: Monitors response times and token usage

### 11. Frontend Display

**Response rendering:** - Displays Jules’s text response - Shows product cards (if any) with images and links - Maintains conversation history - Auto-scrolls to latest message

## Logging System

### Log Storage & Structure

**Database Model**: ChatLog

{  
 userId: String, // User identifier  
 message: String, // Original user message  
 reply: String, // Jules's response  
 intent: String, // Classified intent (optional)  
 timestamp: Date // When interaction occurred  
}

### Log Creation Process

**Every chat interaction triggers logging:** 1. **Message Processing**: After AI generates response 2. **Intent Classification**: Automatic categorization based on content 3. **Log Creation**: Stored in MongoDB with full context 4. **Error Handling**: Logging failures don’t break user experience

**Intent Classification Logic:**

// Dating advice detection  
if (/(ghost|date|relationship|breakup|text|message|call|ignore|respond|feel|hurt|confused|frustrated|angry|sad|upset|anxious|nervous|worried|stressed|overthink|doubt|trust|love|like|crush|feelings|emotion)/i.test(message)) {  
 intent = 'dating\_advice';  
}  
// Style advice detection  
else if (/(shorts|shoes|jacket|shirt|jeans|pants|sneakers|boots|suit|blazer|tie|belt|watch|accessory|outfit|clothing|apparel|fashion|dress|wear|brand|ten thousand|lululemon|nike|adidas|brooks|asics|levi|uniqlo|jcrew|target|amazon|schott|allsaints|leather)/i.test(message)) {  
 intent = 'style\_advice';  
}  
// Image request detection  
else if (/(pic|pics|picture|pictures|image|images|visual|visuals|what\s\*does\s\*it\s\*look\s\*like|outfit\s\*examples?|can\s\*i\s\*see\s\*(it|them)|example\s\*of|examples\s\*of)/i.test(message)) {  
 intent = 'image\_request';  
}  
// Greeting detection  
else if (/(hi|hello|hey|thanks|thank you|bye|goodbye|yes|no|ok|okay)/i.test(message)) {  
 intent = 'greeting';  
}  
else {  
 intent = 'general\_conversation';  
}

### Log Access & Export

**Admin Routes**: /api/logs - **GET /api/logs/export**: Download all logs as CSV file - **GET /api/logs**: Paginated log viewing for admin dashboard

**CSV Export Features:** - **Headers**: userId, message, reply, intent, timestamp - **Filtering**: Admin users see all logs, regular users see only their own - **Formatting**: Properly escaped CSV with quotes and commas - **File Name**: jules\_chat\_logs.csv

**Log Retrieval Features:** - **Pagination**: Configurable page size (default: 50) - **User Filtering**: Filter logs by specific user ID - **Sorting**: Most recent logs first - **Admin Only**: Regular users cannot access logs

### Log Analysis Use Cases

**System Performance:** - Track most common user intents - Monitor response quality and length - Identify failed product searches - Analyze conversation patterns

**User Experience:** - Understand user needs and pain points - Identify popular fashion/dating topics - Track product search success rates - Monitor system reliability

**Business Intelligence:** - Popular brands and products - Common dating scenarios - User engagement patterns - System usage trends

## Key Features & Capabilities

### Conversation Memory

* Stores all messages in MongoDB
* Maintains context across sessions
* Remembers user preferences (gender, style preferences)

### Product Integration

* Real-time product search via Google API
* Brand-specific recommendations
* Shopping links with images and prices
* Filters for men’s clothing only

### Personality Consistency

* Gender-specific responses
* Consistent tone: confident, direct, flirty
* Natural conversation flow
* No AI-style language

### Comprehensive Logging

* Every interaction logged with context
* Automatic intent classification
* Error tracking and debugging
* Performance monitoring
* Admin export capabilities

### Error Handling

* Graceful handling of API failures
* Fallback responses when search fails
* User-friendly error messages
* Session management for anonymous users
* Logging failures don’t break user experience

## Data Flow Example

User: "I need a leather jacket for winter"  
 ↓  
Frontend: POST /api/chat {message: "I need a leather jacket for winter", userId: "123"}  
 ↓  
Backend:   
 - Validates request  
 - Detects gender context (male)  
 - Retrieves conversation history  
 - Classifies as product request  
 ↓  
OpenAI:   
 - Receives conversation context + current message  
 - Generates fashion advice response  
 ↓  
Product Search:  
 - Extracts "leather jacket"  
 - Searches Google for "men's leather jacket buy shop"  
 - Filters results for shopping sites  
 ↓  
Response Processing:  
 - Strips AI closers  
 - Formats product cards  
 ↓  
Logging:  
 - Creates ChatLog entry with intent: "style\_advice"  
 - Stores user message, AI response, and metadata  
 ↓  
Frontend:   
 - Displays Jules's advice  
 - Shows 3 leather jacket product cards

## Technical Stack

### Frontend

* **Framework**: Next.js 14 with TypeScript
* **Styling**: Tailwind CSS
* **State Management**: React hooks
* **Authentication**: JWT tokens stored in localStorage

### Backend

* **Framework**: Express.js
* **Database**: MongoDB with Mongoose
* **AI**: OpenAI GPT-3.5-turbo
* **Search**: Google Custom Search API
* **Authentication**: Passport.js with JWT
* **Logging**: Custom ChatLog system with intent classification

### Infrastructure

* **Frontend Hosting**: Vercel
* **Backend Hosting**: Render
* **Database**: MongoDB Atlas
* **Environment**: Production with environment variables

## Security & Privacy

### Authentication

* JWT-based authentication
* Google OAuth integration
* Session management with MongoDB
* Protected API routes

### Data Protection

* User conversations stored securely
* No sensitive data in logs
* Environment variables for API keys
* CORS configuration for production
* Admin-only access to system logs

## Scalability Considerations

### Performance

* Dynamic token limits based on message type
* Conversation history limited to last 10 messages
* Product search results limited to 3 items
* Request timeouts (15 seconds)
* Efficient logging with minimal overhead

### Monitoring

* Comprehensive logging throughout the system
* Error tracking and debugging
* Health check endpoints
* Deployment version tracking
* Intent classification for analytics
* Performance metrics collection

## Future Enhancements

### Planned Features

* Image generation capabilities
* Voice interaction
* Advanced product filtering
* Personalized style profiles
* Event and class recommendations
* Advanced analytics dashboard

### Technical Improvements

* Caching for product searches
* Real-time notifications
* Advanced conversation analytics
* Multi-language support
* Machine learning for intent classification
* Automated log analysis and insights

This document provides a comprehensive overview of how Jules works from a technical and user experience perspective. The system is designed to be conversational, helpful, and maintain a consistent personality while providing practical fashion and dating advice to men, with full logging and analytics capabilities for continuous improvement.