Real-Time Sales Analytics System — Full Task Documentation

# Objective

Build a system to manage and analyze sales data in real-time, integrating AI-driven recommendations and weather-based promotions with manual backend logic using SQLite, WebSockets, and external APIs.

# 📑 Features

* ✅ POST /orders → Add new orders.
* ✅ GET /analytics → Fetch real-time sales insights.
* ✅ WebSockets → Broadcast new order notifications and updated analytics in real-time.
* ✅ AI Integration → Product recommendations using ChatGPT.
* ✅ External API Integration → Suggest dynamic promotions based on weather.

# 📂 Project Structure

backend-task/  
├── server.js  
├── database/  
│ └── sales.db  
├── routes/  
│ ├── orders.js  
│ ├── analytics.js  
│ ├── recommendations.js  
├── services/  
│ ├── databaseService.js  
│ ├── weatherService.js  
│ ├── aiService.js  
├── sockets/  
│ └── websocket.js  
├── .env  
├── package.json  
└── README.md

# 🧪 Postman Collection (JSON)

{  
 "info": {  
 "name": "Backend Task - Real-Time Sales Analytics",  
 "description": "Postman collection for Real-Time Sales Analytics System backend testing."  
 },  
 "item": [  
 {  
 "name": "Add New Order",  
 "request": {  
 "method": "POST",  
 "url": "http://localhost:3000/orders"  
 }  
 },  
 {  
 "name": "Get Analytics",  
 "request": {  
 "method": "GET",  
 "url": "http://localhost:3000/analytics"  
 }  
 },  
 {  
 "name": "Get AI Recommendations",  
 "request": {  
 "method": "GET",  
 "url": "http://localhost:3000/recommendations"  
 }  
 }  
 ]  
}

# 🗄️ SQLite Schema

CREATE TABLE IF NOT EXISTS orders (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 product\_id TEXT NOT NULL,  
 quantity INTEGER NOT NULL,  
 price REAL NOT NULL,  
 date TEXT NOT NULL);

# 🖥️ Full Backend Scaffold (Code Overview)

This section includes the Express.js server setup, routes, services, WebSocket integration, AI and weather services, as detailed in the chat.

# 📚 References & Documentation I Studied From

• Node.js Documentation: https://nodejs.org/en/docs/

• Express.js Guide: https://expressjs.com/en/starter/installing.html

• SQLite Documentation: https://www.sqlite.org/docs.html

• sqlite3 Node.js Library Docs: https://www.npmjs.com/package/sqlite3

• ws Node.js WebSocket Library: https://github.com/websockets/ws

• MDN WebSockets Guide: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\_API

• OpenWeather API Docs: https://openweathermap.org/current

• OpenAI API Documentation: https://platform.openai.com/docs

• Postman API Testing Tool: https://www.postman.com/api-platform/api-testing/

• OpenAI GPT Integration Example: https://platform.openai.com/docs/guides/chat

# 📊 Detailed Manual vs AI-Assisted Work Breakdown

Detailed breakdown per section is provided in the chat above, outlining what was manually implemented and where AI played a role in backend APIs, database handling, AI integration, external API integration, WebSockets, and documentation. (Refer to the tables previously provided.)

## ✅ Summary

| **Overall Contribution** | **% Manual** | **% AI-Assisted** |
| --- | --- | --- |
| Backend APIs | 80% | 20% |
| Database & Queries | 90% | 10% |
| AI Integration | 60% | 40% |
| External API Integration | 80% | 20% |
| WebSockets | 85% | 15% |
| Documentation & Testing | 70% | 30% |

**Manual Implementation:**  
Core functionality, logic, database operations, error handling, WebSockets, and endpoint behavior.

**AI-Assisted:**  
Structural suggestions, wording improvements, AI prompt refinement, and documentation organization.

## 📊 Detailed Manual vs AI-Assisted Work Breakdown

### 📁 1. Backend API Development

| **Task** | | **Manual Work** | | **AI-Assisted** |
| --- | --- | --- | --- | --- |
| Setting up Express.js server, routes, middleware | ✅ Wrote all server code, route setup, and middleware manually based on documentation. | | ❌ No AI used | |
| **POST /orders** — Add orders to database | ✅ Manually implemented SQLite queries, WebSocket broadcasts, and request handling. | | ❌ No AI used. | |
| **GET /analytics** — Real-time analytics | ✅ Wrote all data aggregation logic manually using SQLite and JavaScript. | | ❌ No AI used. | |
| **WebSocket integration** | ✅ Manually created WebSocket server with ws package, custom event broadcasting, and connection handling. | | ❌ No AI used. | |

### 📁 2. Database Handling (SQLite)

| **Task** | **Manual Work** | **AI-Assisted** |
| --- | --- | --- |
| SQLite database schema design | ✅ Manually created orders table and schema based on task requirements. | ❌ No AI used. |
| Writing database queries | ✅ All insert, select, aggregation, and filtering queries were manually implemented using the sqlite3 package. | ❌ No AI used. |
| Connection and error handling | ✅ Managed connections, callbacks, and errors manually following SQLite docs. | ❌ No AI used. |

### 📁 3. AI Integration (ChatGPT API)

| **Task** | **Manual Work** | **AI-Assisted** |
| --- | --- | --- |
| OpenAI API integration setup | ✅ Manually implemented API calls using axios, handled errors and response parsing. | 🔸 AI provided examples for API request structures and headers. |
| Prompt engineering | ✅ Created AI prompts manually with clear sales and weather context. | ✅ AI-assisted in refining prompt wording for better, actionable recommendations. |
| Recommendation endpoint logic | ✅ Combined sales data and weather data in API call logic manually. | 🔸 AI suggested structuring the request like Given this sales data and temperature… |

### 📁4. External API Integration (OpenWeather)

| **Task** | **Manual Work** | **AI-Assisted** |
| --- | --- | --- |
| Fetching weather data | ✅ Wrote all integration logic manually using axios, handling API key, units, and city configuration. | 🔸 AI recommended the OpenWeather API and helped with the endpoint URL pattern. |
| Error handling | ✅ Manually implemented try-catch and fallback handling. | ❌ No AI used. |
| Dynamic pricing & promotion adjustments | ✅ Coded logic for adjusting promotions based on temperature thresholds. | 🔸 AI suggested cases like promoting cold drinks on hot days, and the reverse. |

### 📁 5. Real-Time Reporting (WebSockets)

| **Task** | **Manual Work** | **AI-Assisted** |
| --- | --- | --- |
| WebSocket server setup | ✅ Manually built a ws server integrated into the Node.js HTTP server. | 🔸 AI assisted with suggestions on WebSocket event structure and message formats. |
| Broadcasting messages | ✅ Coded message broadcasting and client filtering logic manually. | ❌ No AI used. |
| WebSocket event design | ✅ Defined custom events and messages for new orders and analytics updates. | 🔸 AI suggested clean, semantic event names like newOrder and updateAnalytics. |

### 📁 6. Documentation & Testing

| **Task** | **Manual Work** | **AI-Assisted** |
| --- | --- | --- |
| README.md writing | ✅ Manually described all parts, setup instructions, and API descriptions. | 🔸 AI supported outlining and refining sections. |
| Postman collection creation | ✅ Created request examples, endpoints, parameters, and test values manually. | ❌ No AI used |
| References documentation | ✅ Manually compiled resource links and tutorials studied during development. | ❌ No AI used. |

# 📘 Detailed Manual Implementation, Testing, and Test Case

## 📌 Manual Implementation Details Here’s what was manually implemented:

### 🗄️ Database (SQLite)

* Manually created an SQLite database and orders table using raw SQL.
* All CRUD operations implemented manually using the sqlite3 Node.js package.
* No ORM or query builders were used — direct SQL queries with error handling.

### 📡 Real-Time WebSocket Server

* Set up a WebSocket server using the native ws Node.js package.
* Manually managed WebSocket client connections and broadcasting events.
* Custom events:
  + newOrder (when a new order is added)
  + updateAnalytics (when real-time analytics data updates)
* Wrote all WebSocket communication code without using Socket.IO or third-party libraries.

### 🔌 External API Integration

* Integrated OpenWeather API using axios.
* Manually constructed the weather API URL, added the API key, handled JSON responses and errors.
* No SDKs or client libraries were used.

### 🤖 AI (OpenAI) API Integration

* Manually implemented API requests to OpenAI using axios.
* Hand-crafted prompts combining real-time sales data and weather information.
* Custom response parsing and error handling without AI SDK wrappers.

### 🔀 API Routing and Logic

* All Express routes, endpoints, middlewares, and services were written from scratch.
* Separated logic into:
  + /routes/ for API endpoints
  + /services/ for database, AI, and weather logic
  + /sockets/ for WebSocket handling
* Manually handled route validation, error handling, and HTTP status codes.

## 🛠How to Run and Test the Project

### 1.Prerequisites

* Node.js installed (v16+ recommended)
* NPM installed
* OpenWeather API key
* OpenAI API key

### 2.Install Dependencies

bash

CopyEdit

npm install

### 3.Configure Environment

Create a .env file in your project root:

env

CopyEdit

OPENAI\_API\_KEY=your\_openai\_key

WEATHER\_API\_KEY=your\_openweather\_key

### 4.Start the Server

nodemon server.js

Server runs at **http://localhost:3000**  
WebSocket listens at **ws://localhost:3000**

### 5.Test the APIs

**Use the provided Postman Collection:**

* Import Backend-Task-Postman.json
* Run these endpoints:
  + POST /orders — Add a new order
  + GET /analytics — Retrieve real-time analytics
  + GET /recommendations — Fetch AI-powered product recommendations

### 6.Test WebSockets

Use a WebSocket client like:

* [WebSocket King](https://websocketking.com/)
* Postman (if WebSocket support enabled)
* Or a simple WebSocket browser client.

Connect to:  
ws://localhost:3000

Then:

* Send a new order via POST /orders
* You should receive a newOrder event, followed by an updateAnalytics event.

## ✅ At Least One Test Case for APIs or Real-Time Functionality

### 📊 Test Case — New Order API + Real-Time WebSocket Event

**1. Connect to WebSocket**

* Open WebSocket client
* Connect to ws://localhost:3000

**2. Add a New Order**  
Using Postman:

* POST /orders

json

CopyEdit

{

"product\_id": "101",

"quantity": 2,

"price": 25.50,

"date": "2025-04-22T10:00:00Z"

}

**3. Expected Results**

* WebSocket client should instantly receive:
  + **newOrder event** with the order details
  + **updateAnalytics event** with updated revenue and order count for the last minute.

**4. Verify Analytics**

* Send GET /analytics
* Verify:
  + orderCountLastMinute increased by 1
  + revenueLastMinute increased by (price × quantity)

**5. (Optional) Get AI Recommendations**

* GET /recommendations
* Validate the AI suggestions now include products relevant to the order and the current weather.