SIT314 – Distinction Task 4.2D

# Project Status Report: Smart Warehouse Inventory & Delivery Management System

Student: Ocean (s223503101)

Date: 1 September 2025

## 1. Introduction

The project requires the design and implementation of an IoT-based scalable solution that demonstrates event-driven processing, cloud integration, and the ability to handle dynamic workloads. The chosen project is the Smart Warehouse Inventory and Delivery Management System, which will monitor warehouse inventory levels through simulated sensors, process events in real-time using Node-RED, and update an inventory microservice that manages stock data. In its final form, the project will integrate with cloud services (AWS DynamoDB and EC2), extend into order and delivery microservices, and demonstrate scalability through testing and simulation. This report provides a status update on the project’s current progress, highlights completed components, outlines challenges encountered, and specifies pending work required to reach the final deliverable.

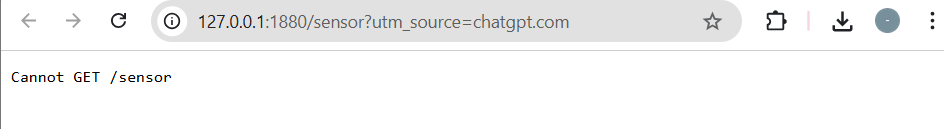
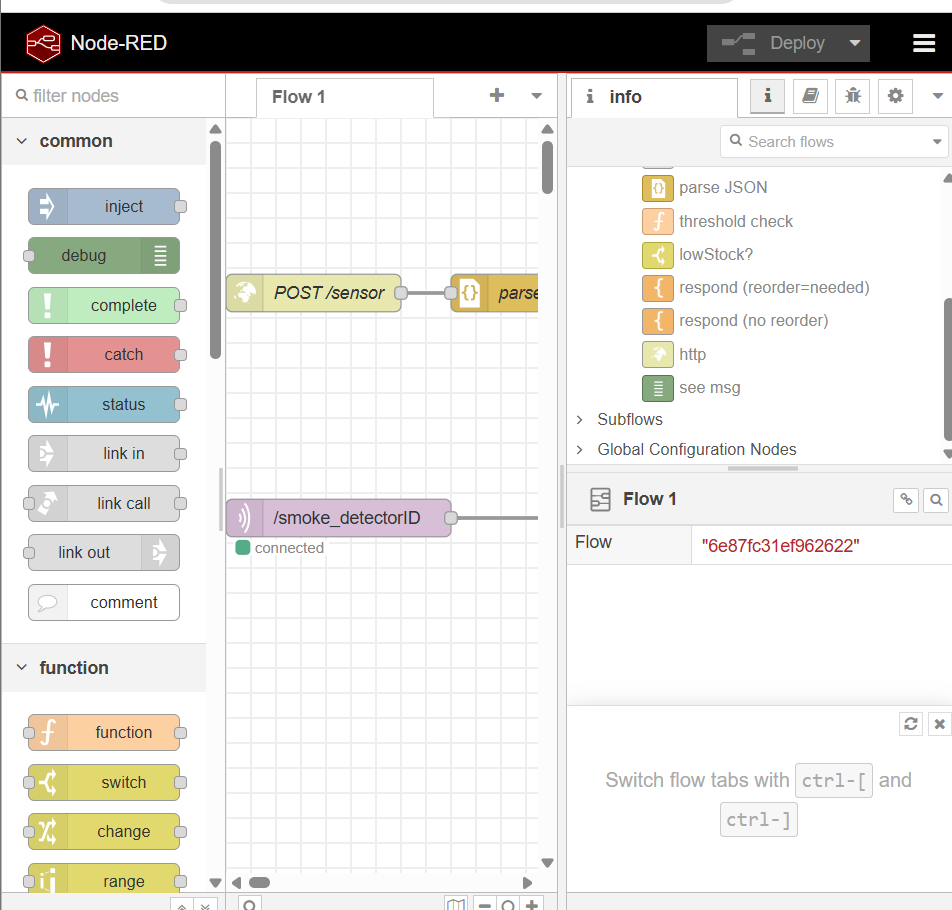
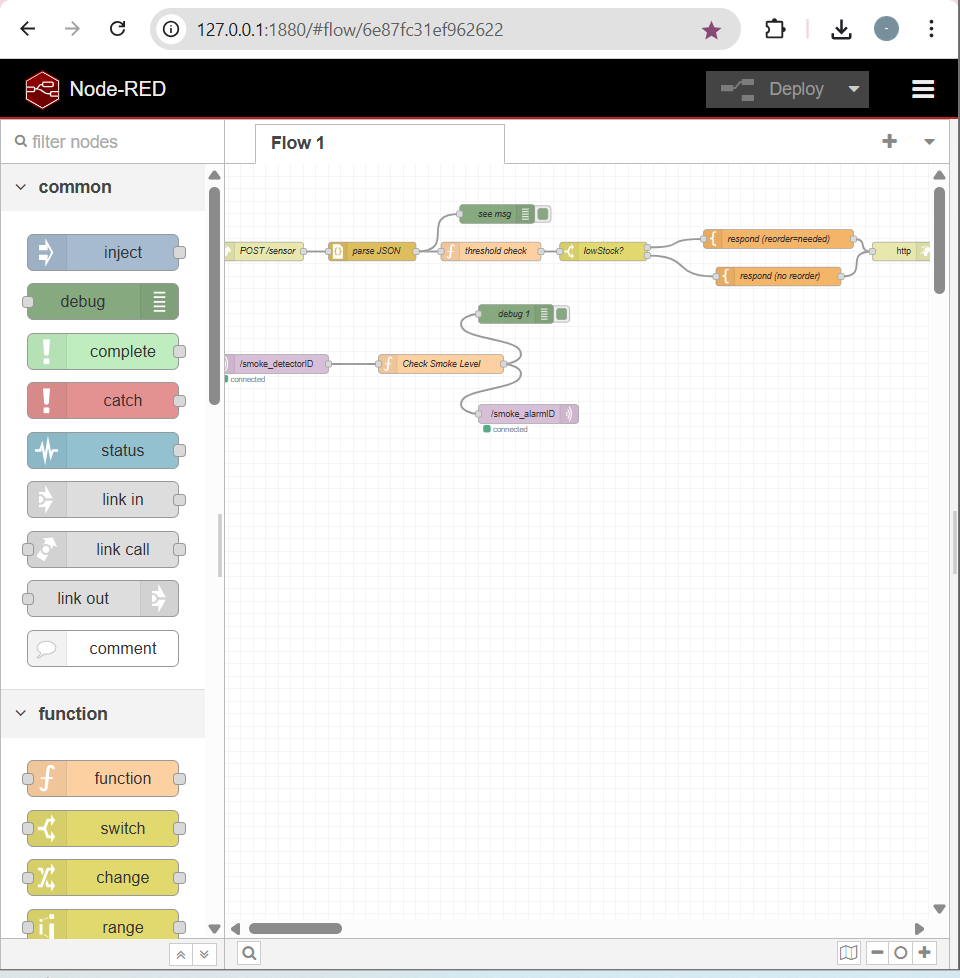
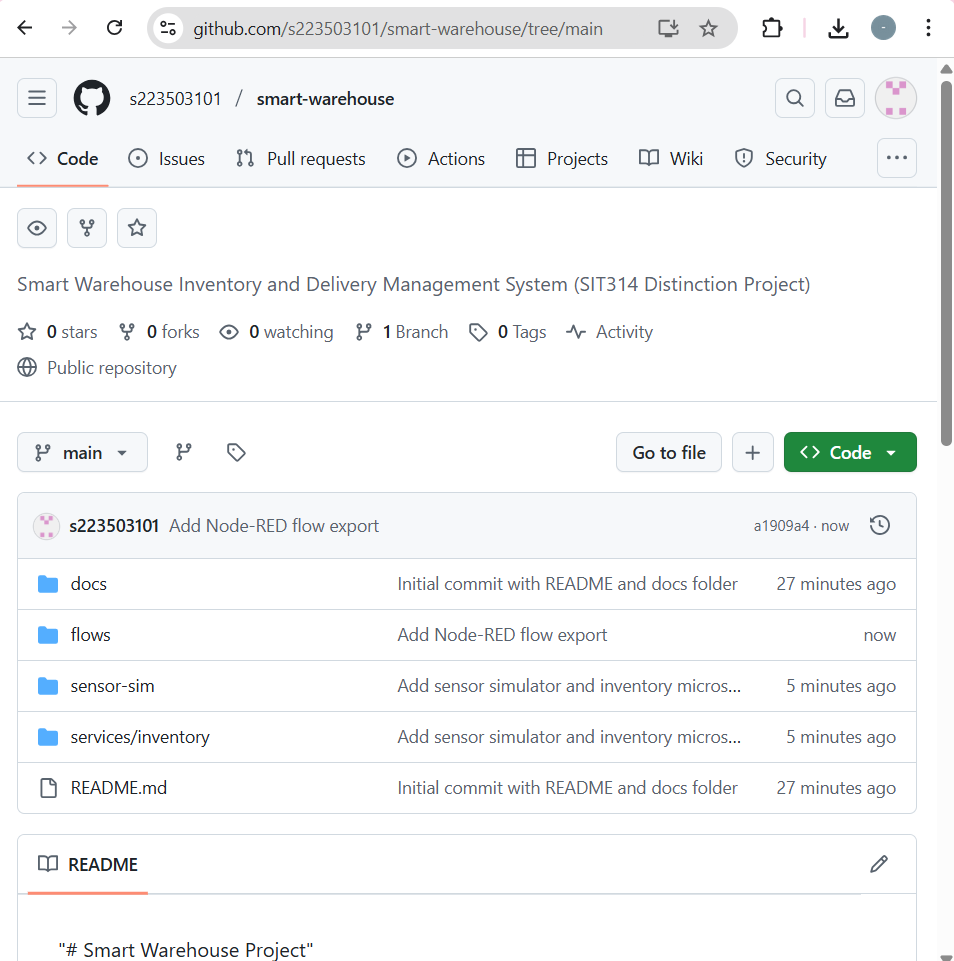
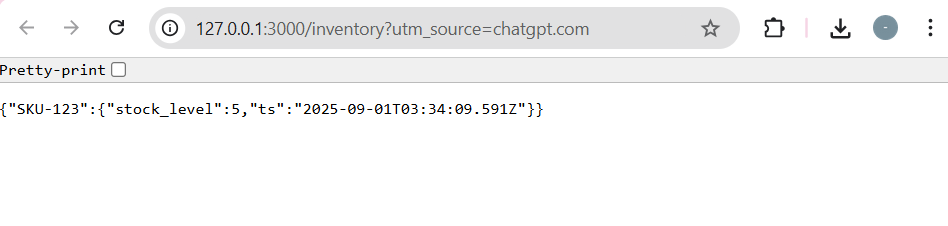
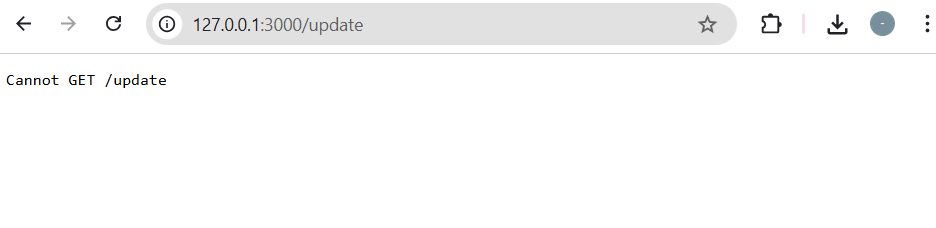
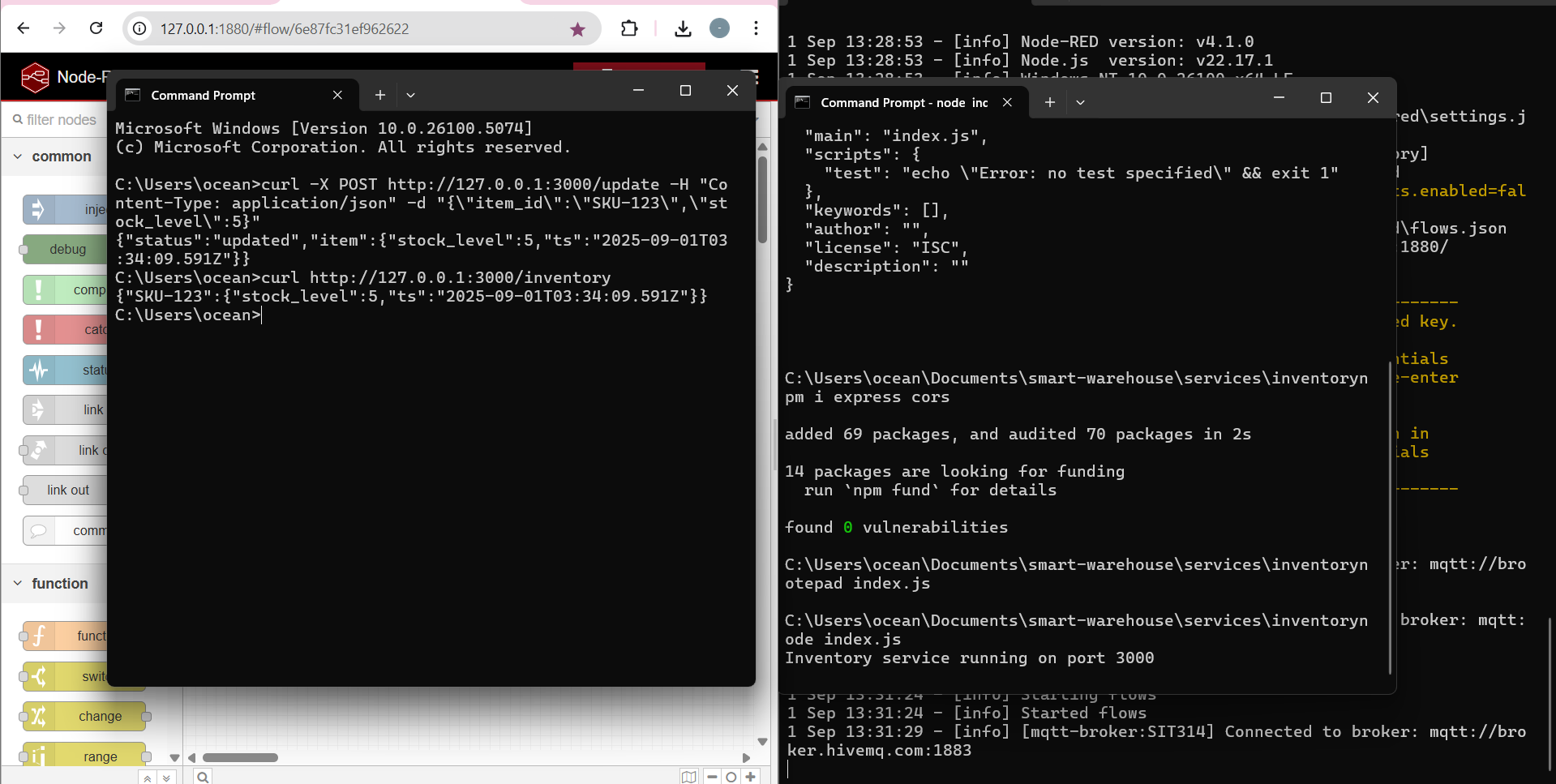
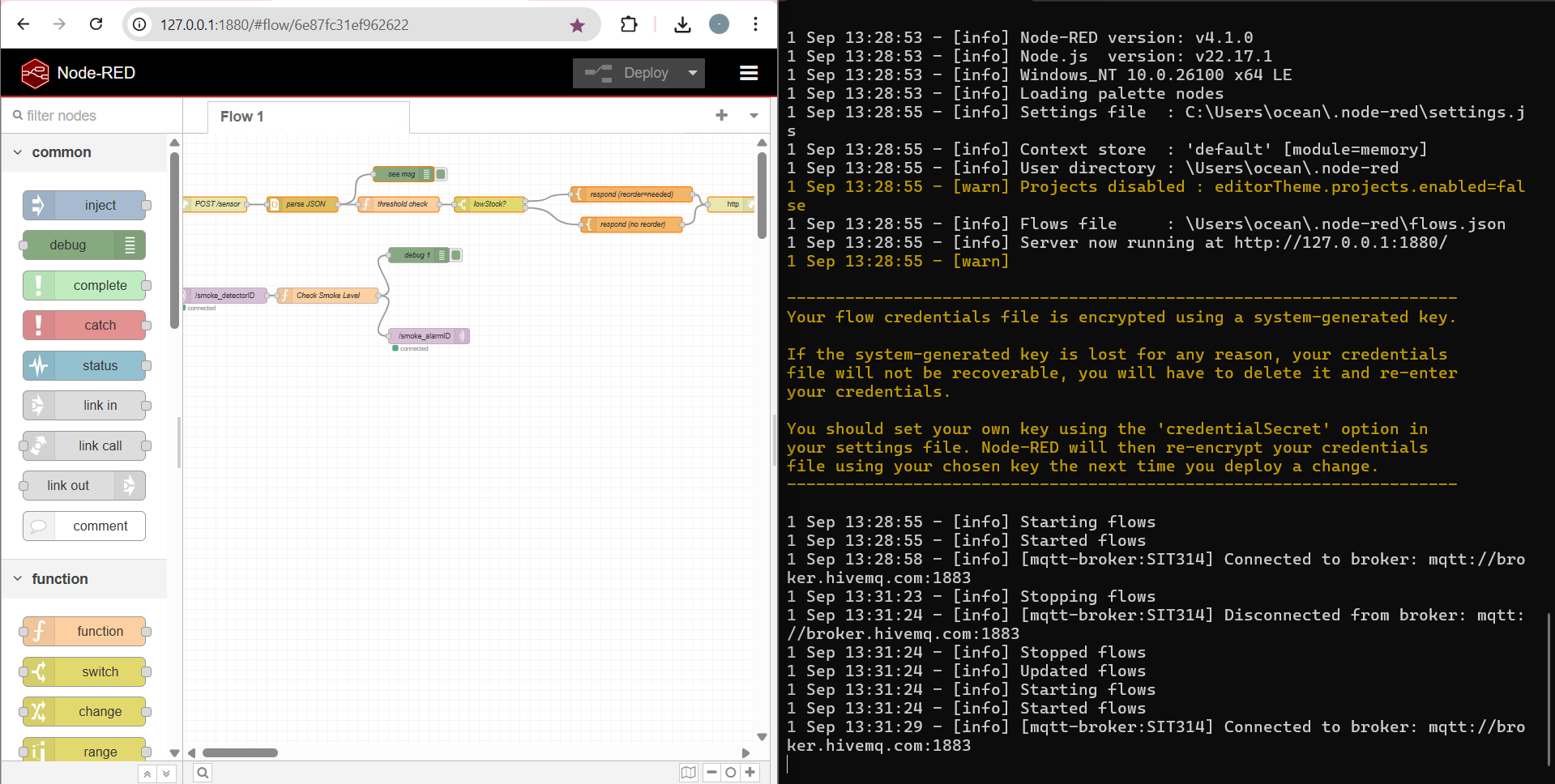
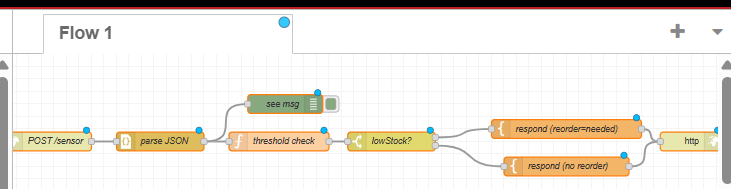
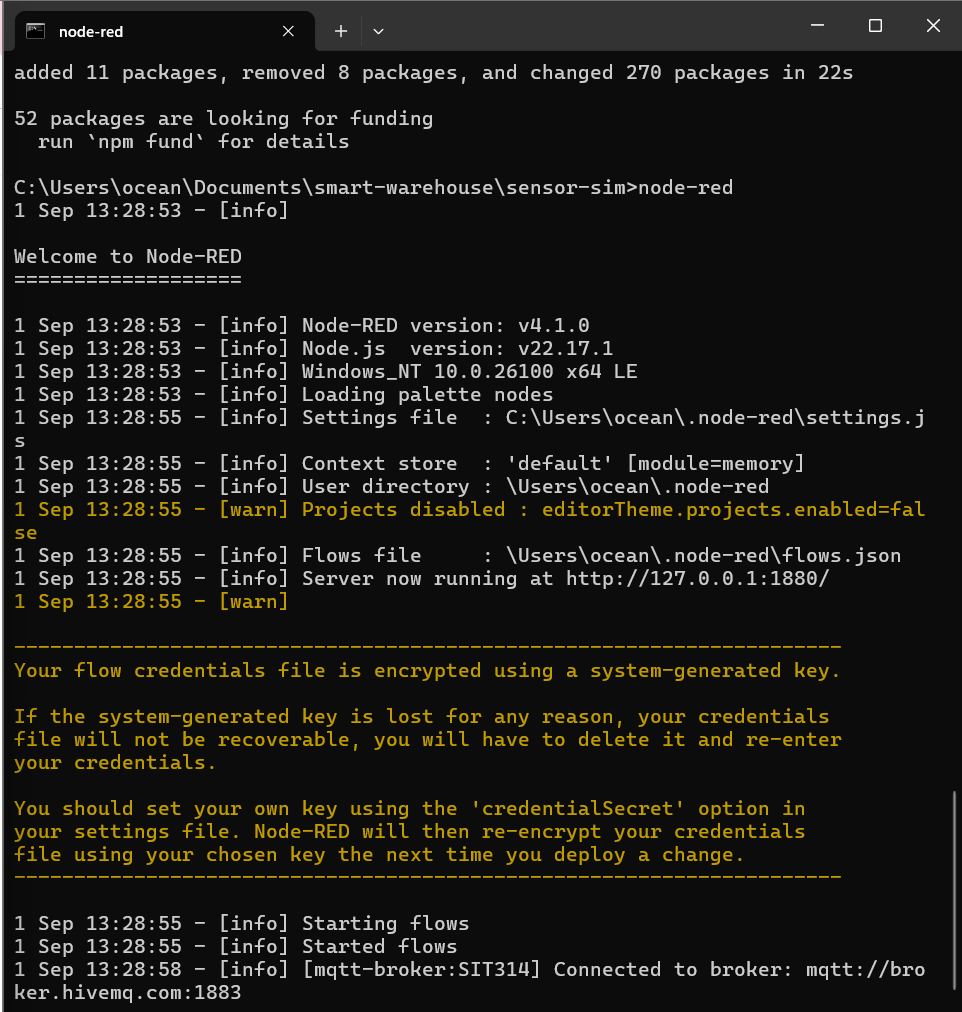
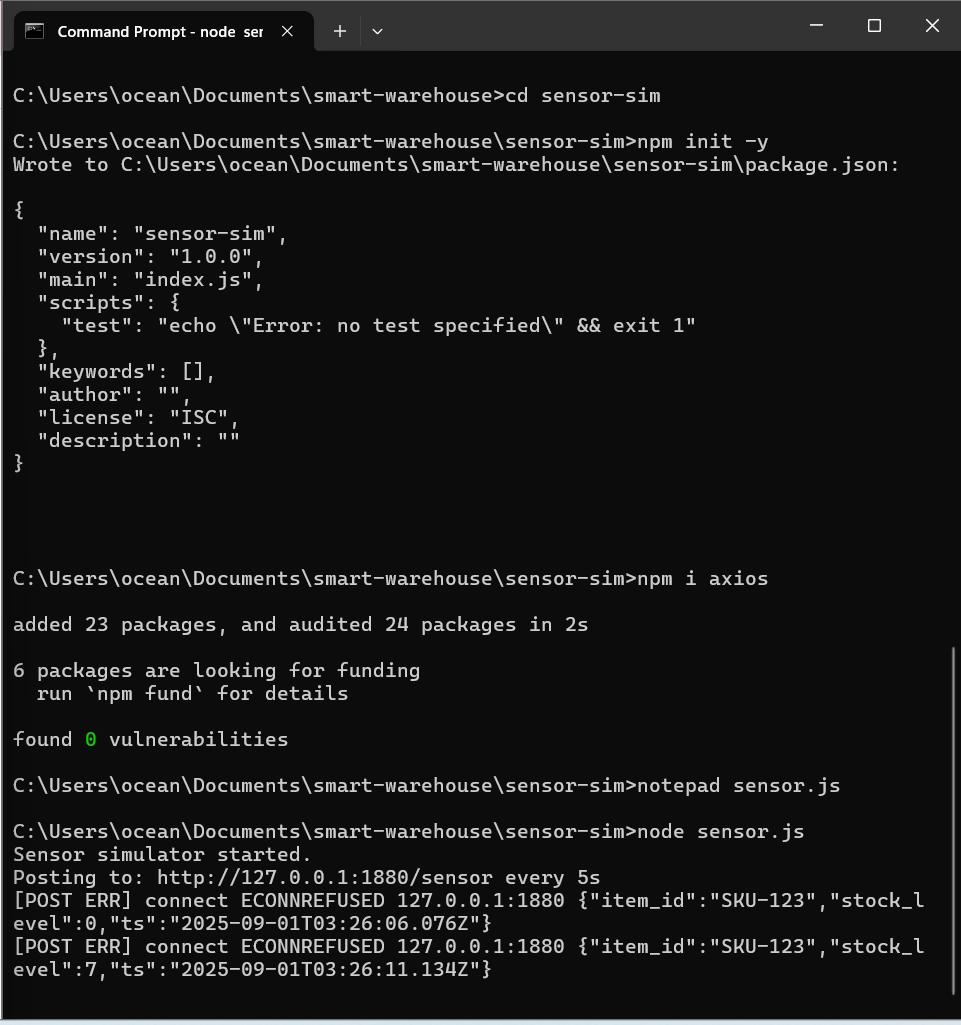
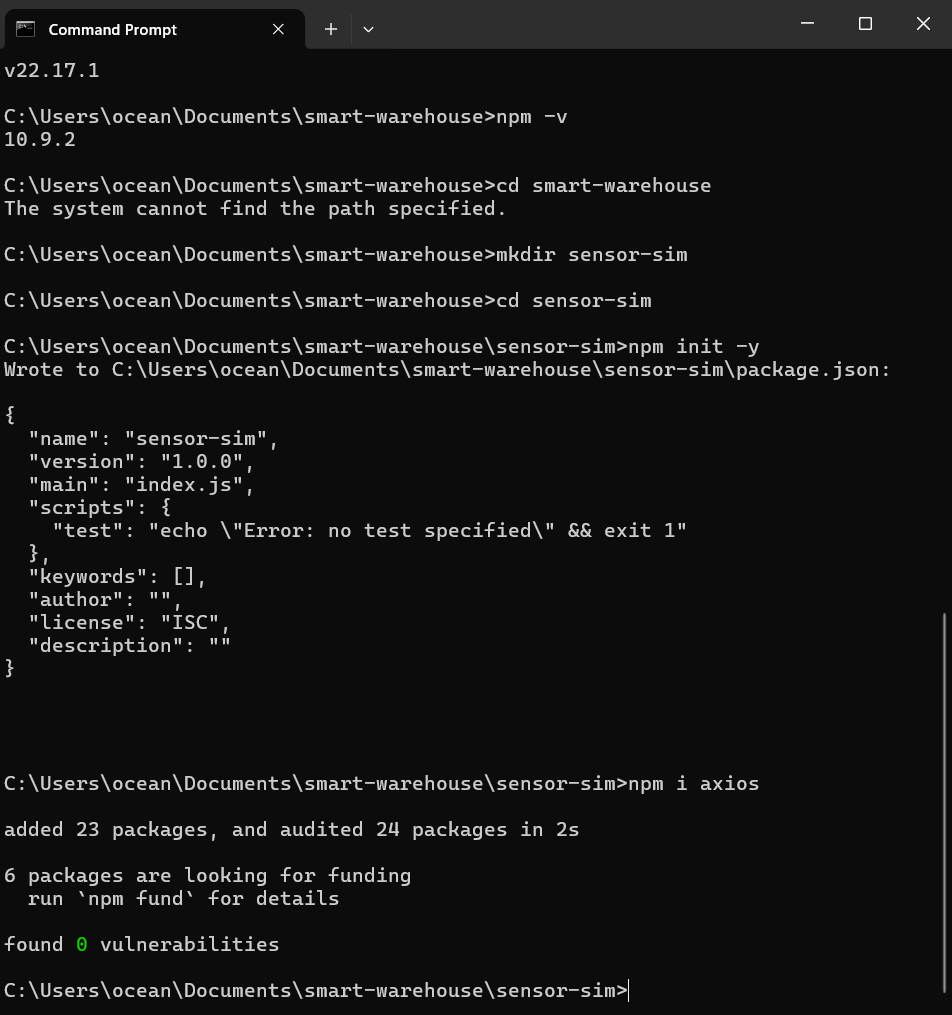
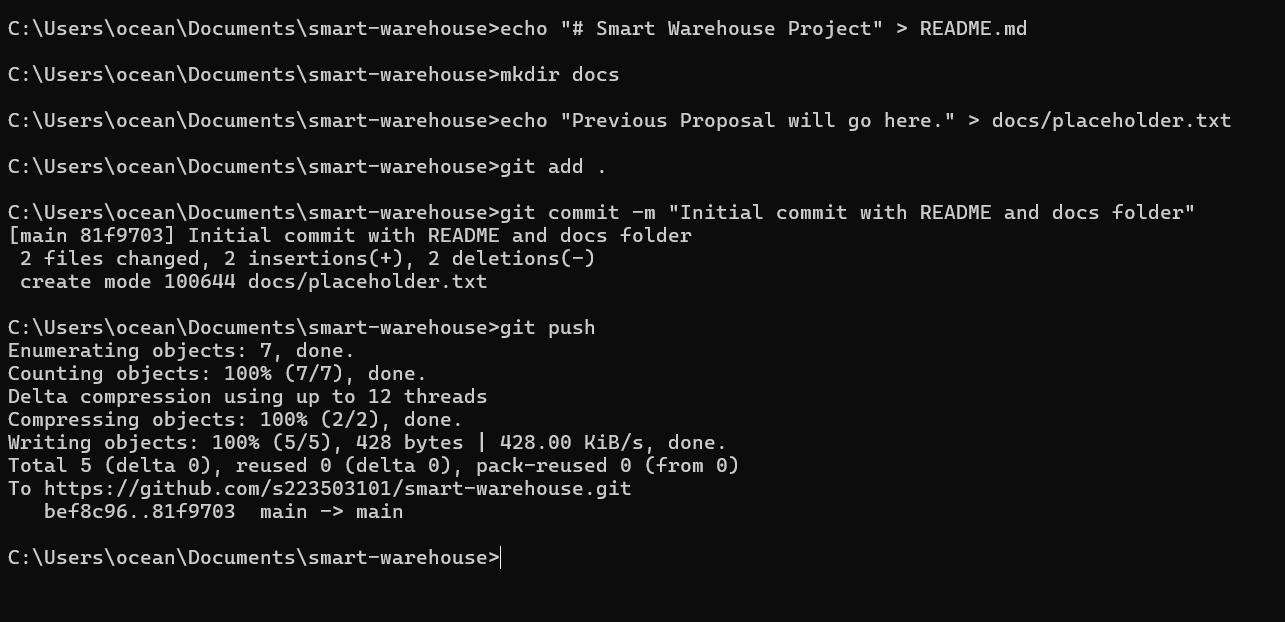
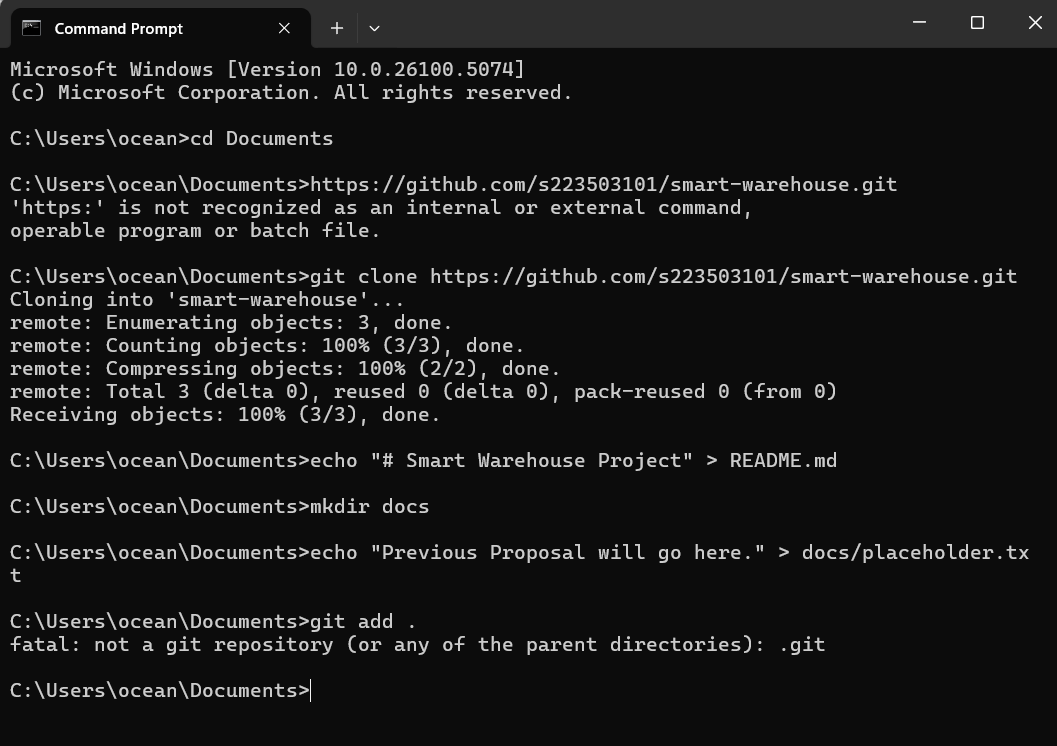
## 2. Project Objectives

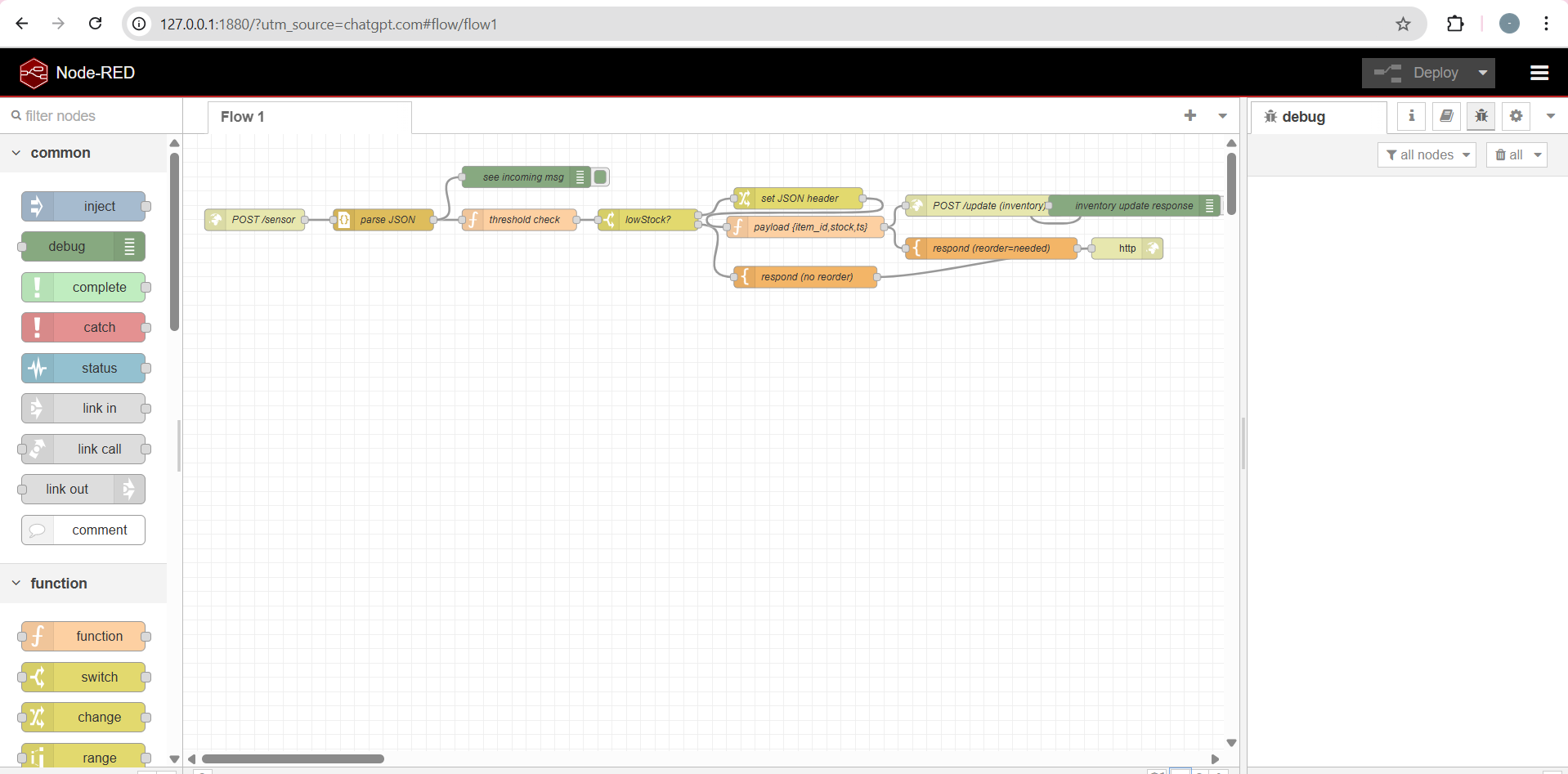
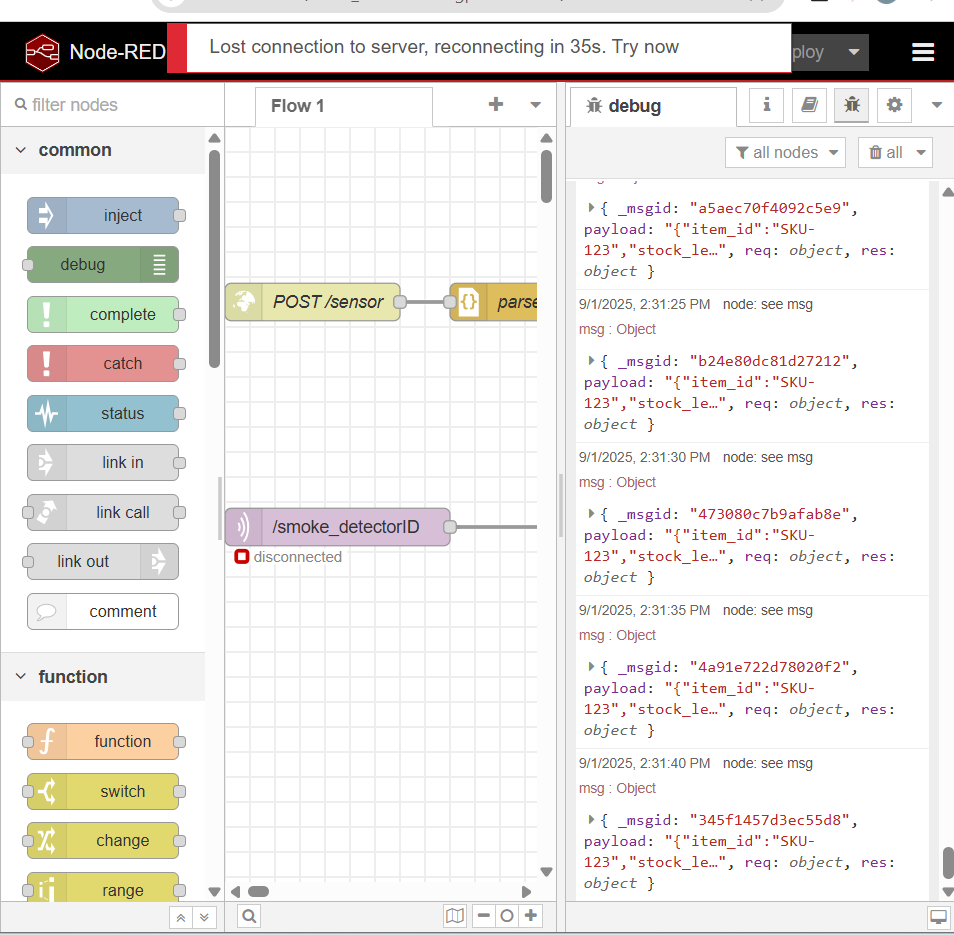
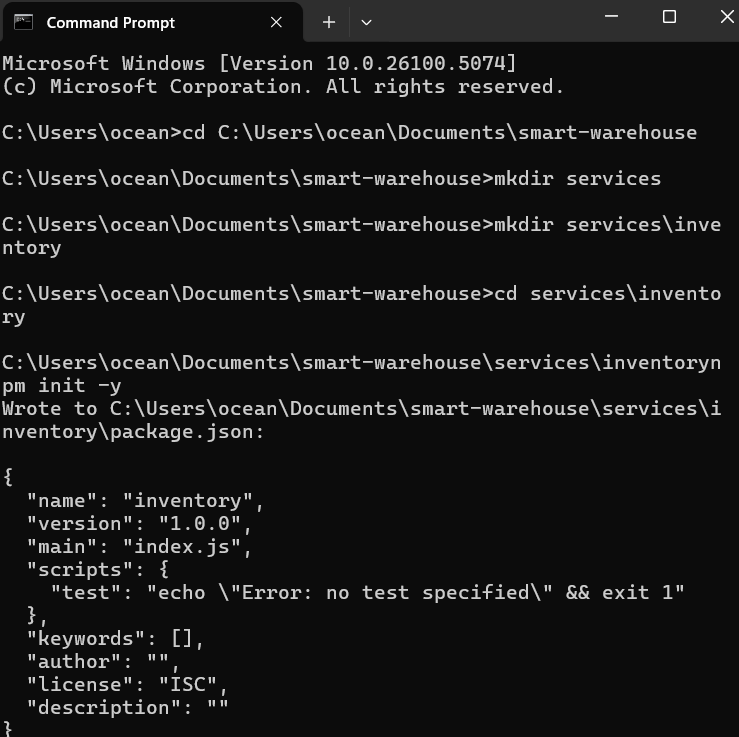
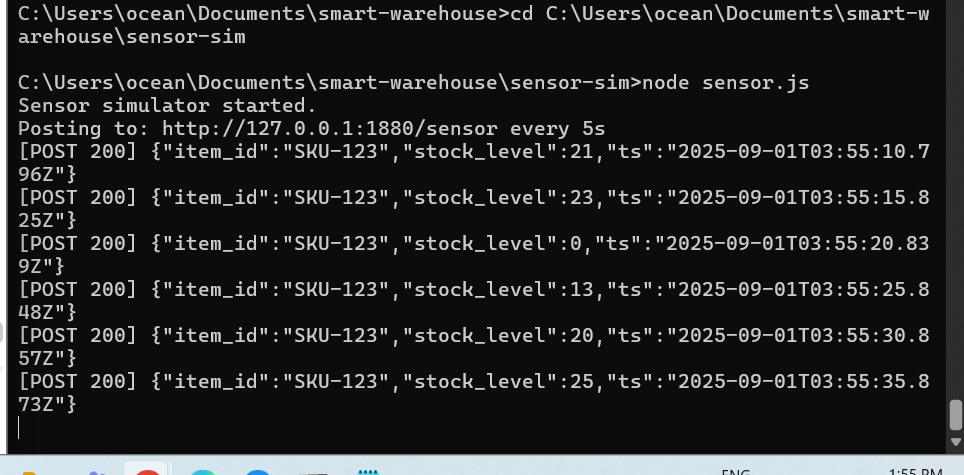
- Sensor Simulation: Implement simulated IoT sensors to generate inventory data.  
- Event Processing: Use Node-RED to apply rules, thresholds, and trigger reorder events.  
- Inventory Management: Build a microservice to store, update, and retrieve inventory records.  
- Cloud Integration (pending): Deploy to AWS infrastructure, using DynamoDB for persistence and EC2/Lambda for scalability.  
- Extended Microservices (pending): Add services for order handling and delivery scheduling.  
- Testing and Scalability (pending): Conduct functional, integration, and load testing with multiple simulated sensors.

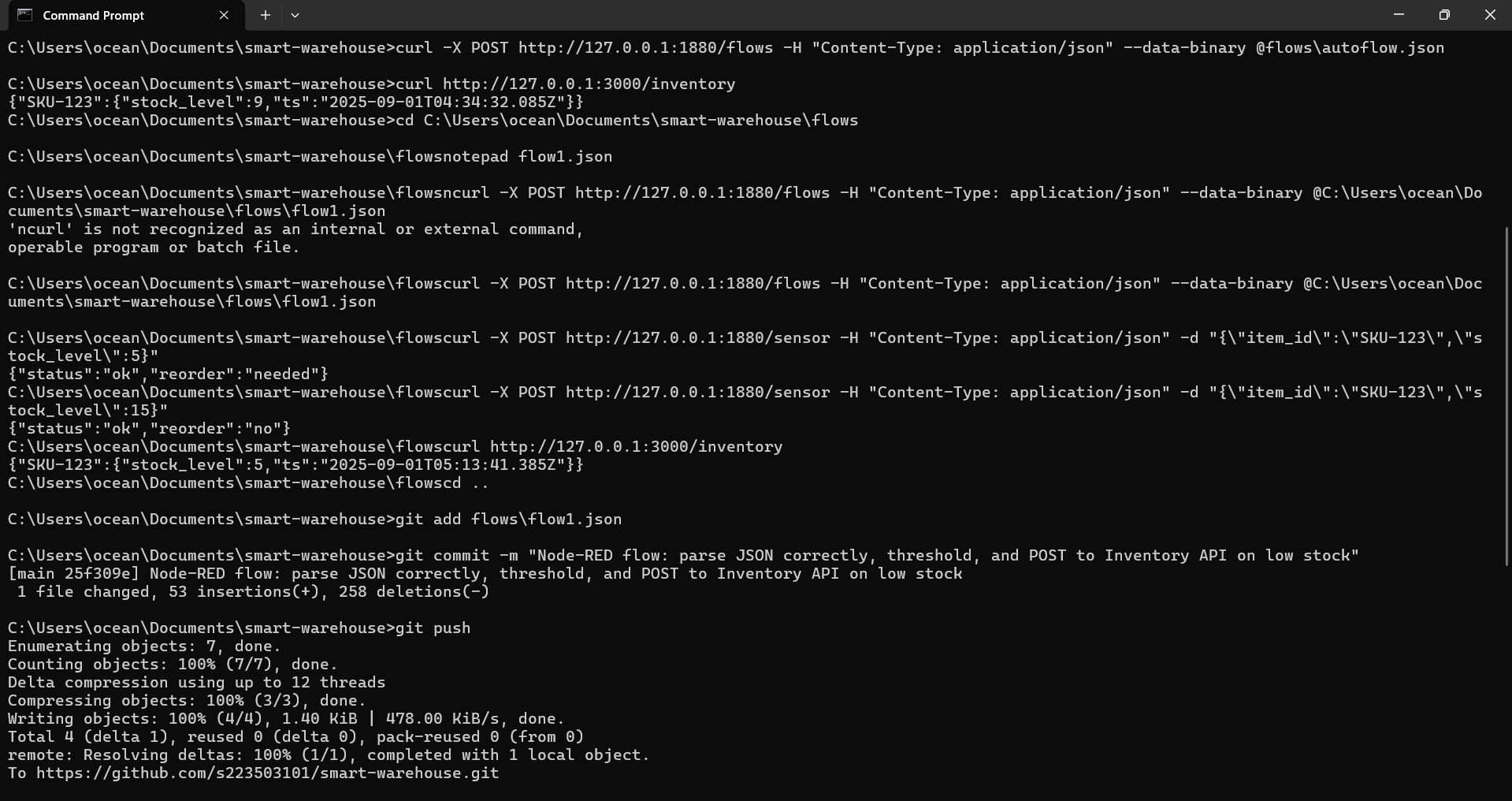
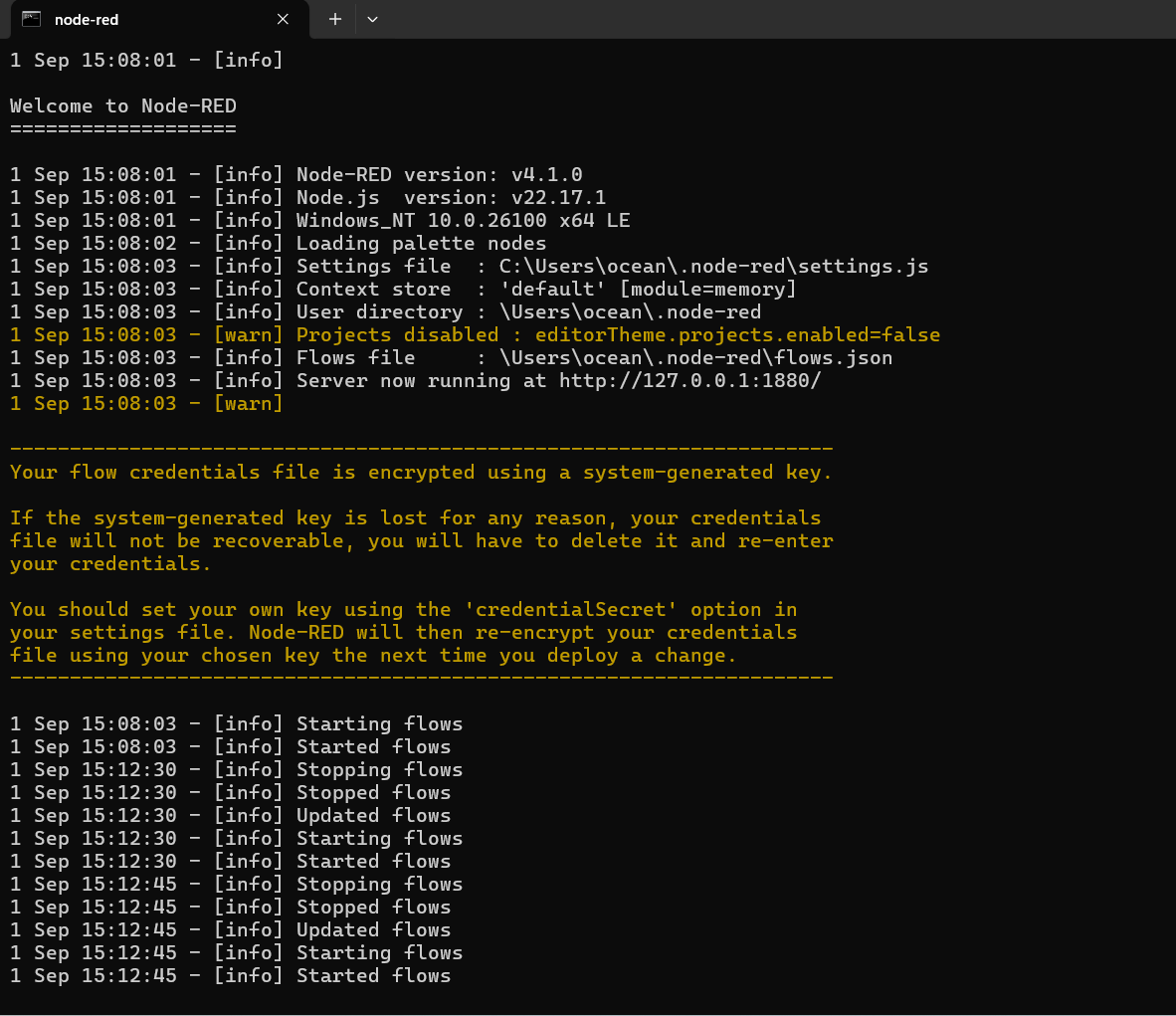
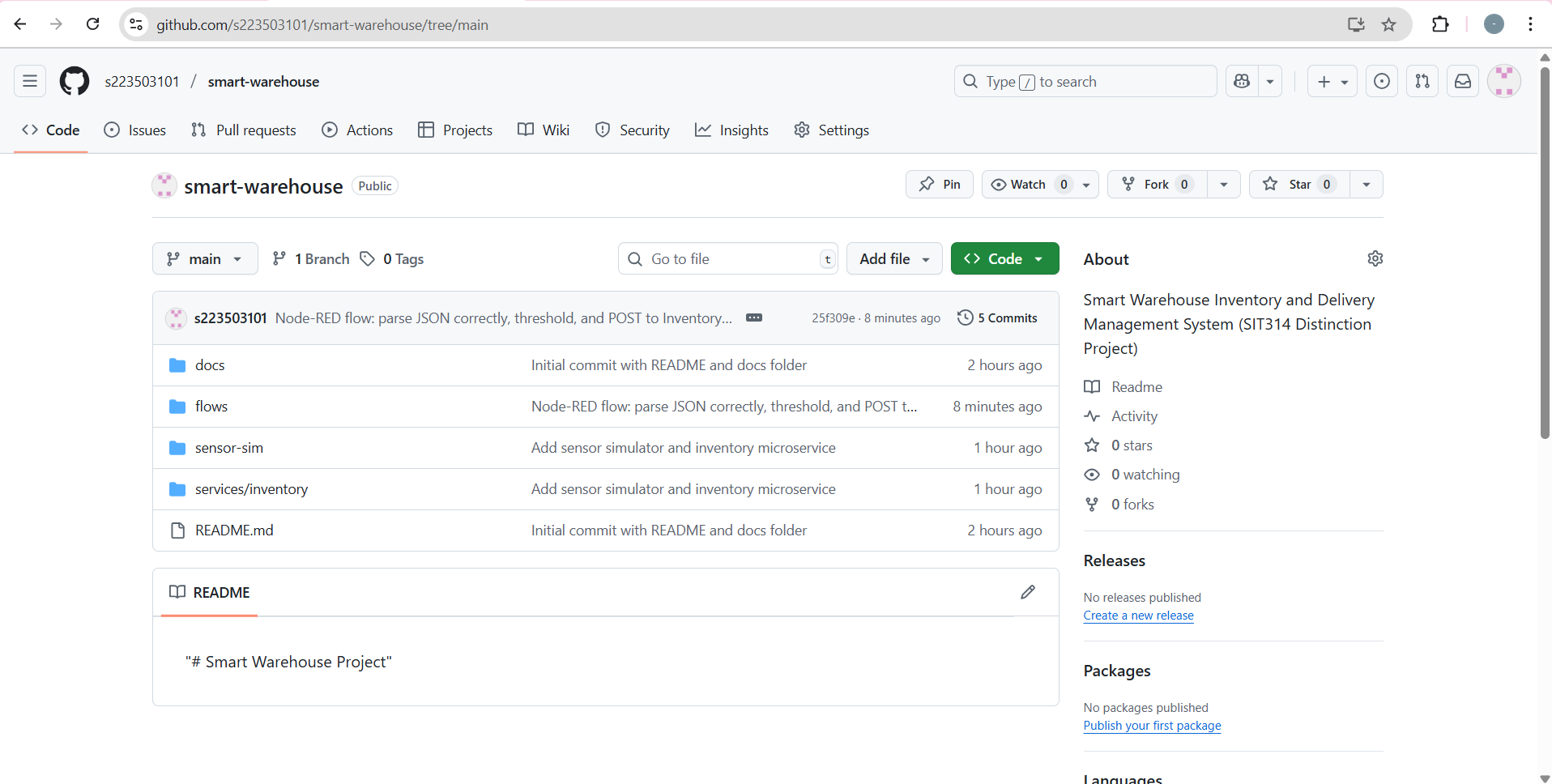
## 3. Progress Achieved So Far

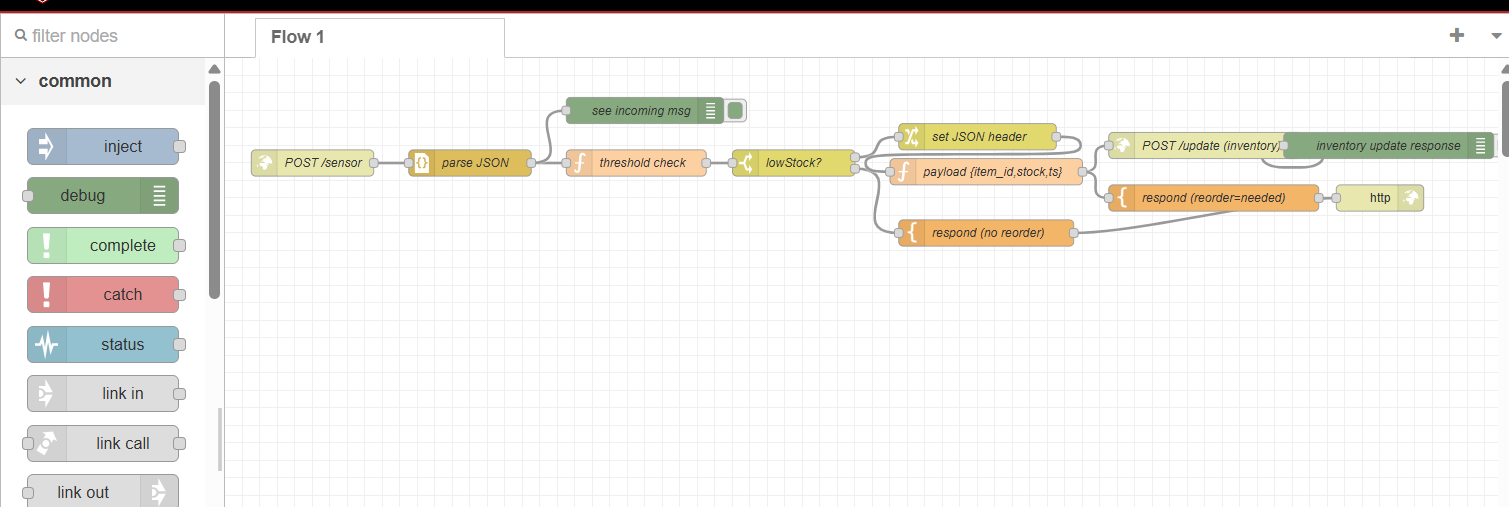
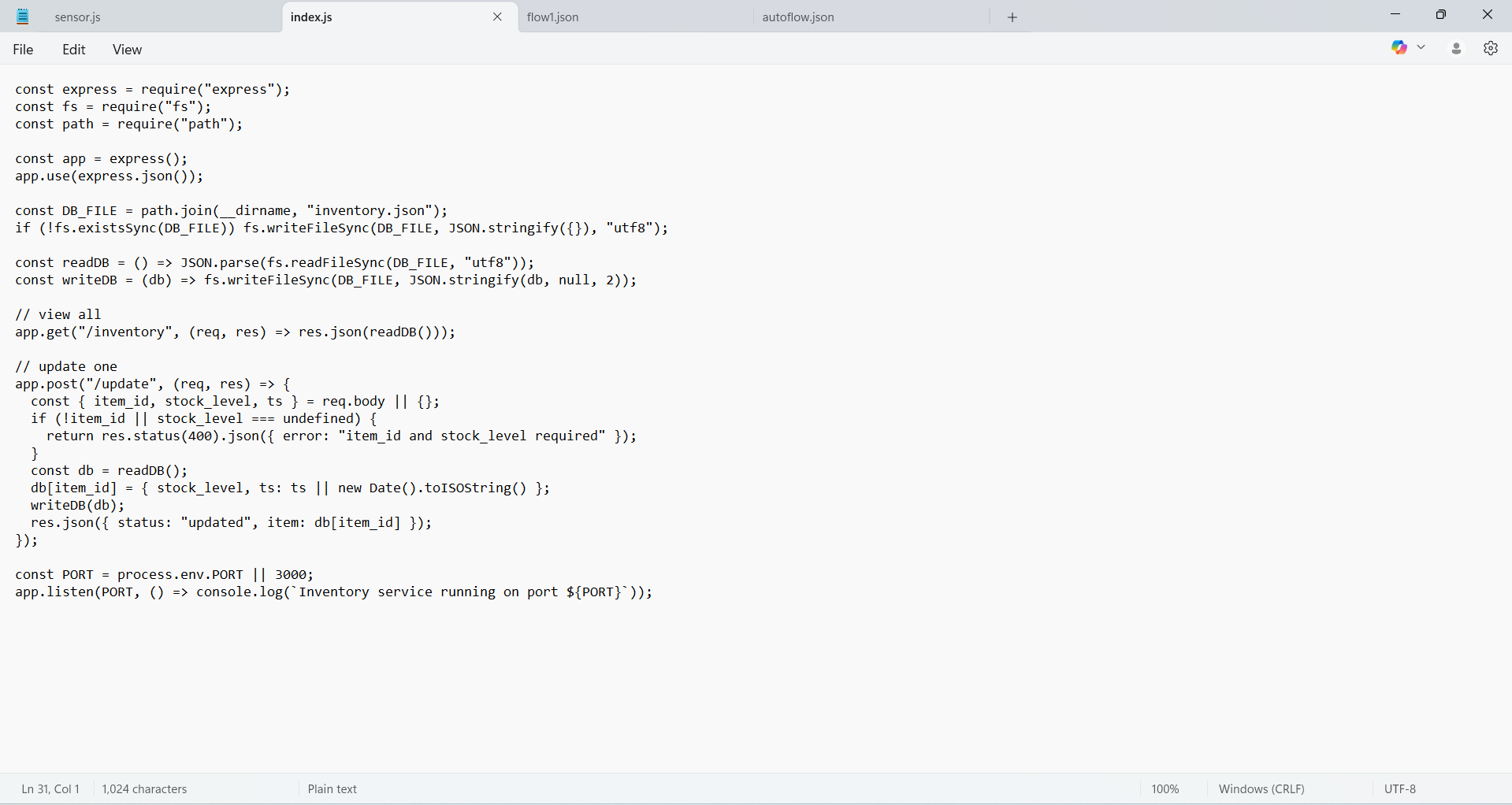
### 3.1 Repository Setup

A structured GitHub repository (\*smart-warehouse\*) has been created and organised into clear modules. The repository contains: sensor-sim/ (Node.js simulator), services/inventory/ (inventory microservice), flows/ (Node-RED flow export), docs/ (documentation files), and README.md (project overview). All updates have been committed and pushed to GitHub, ensuring version control.

A screenshot of a computer

AI-generated content may be incorrect.A black screen with white text

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.

### 3.2 Sensor Simulator

A Node.js script (sensor.js) was developed to simulate warehouse stock levels. It generates random stock\_level values every five seconds and POSTs them to Node-RED at /sensor. The simulator was tested successfully with console logs confirming transmission, e.g. [POST 200] {...}.

### 3.3 Node-RED Flow

A Node-RED flow was designed to process incoming sensor data. It parses JSON payloads, applies a threshold check for stock\_level < 10, and returns structured JSON responses. Debug nodes were added to visualise incoming messages and inventory responses. The flow was exported (flow1.json) and stored in the repository.

### 3.4 Inventory Microservice

An Express.js microservice was implemented with two endpoints: POST /update (updates inventory.json) and GET /inventory (retrieves inventory data). The service was validated using curl and browser testing, confirming correct updates and retrieval.

### 3.5 End-to-End Integration

Node-RED and the inventory service were successfully integrated. When low stock events are detected, Node-RED triggers a POST /update call. Testing confirmed that stock\_level=5 triggered a reorder and updated the DB, while stock\_level=15 did not update the DB. Verification through GET /inventory confirmed correct persistence of values.

## 4. Challenges and Solutions

|  |  |  |
| --- | --- | --- |
| Challenge | Impact | Solution Implemented |
| Payload parsing in Node-RED – sensor data arrived as strings | Threshold checks failed as stock\_level was not numeric | Added JSON node to convert to objects; updated function logic |
| Flow import/export issues – difficulties using Admin API | Slowed testing and configuration | Used Node-RED UI import/export; committed flow1.json to GitHub |
| Service integration – Inventory API triggered incorrectly | Database updated regardless of stock level | Added Switch node to ensure /update triggers only when lowStock==true |
| Debug visibility – unclear data flow | Delayed troubleshooting | Added Debug nodes to monitor incoming messages and inventory responses |

## 5. Next Steps

Although the local prototype is functional, the following tasks are pending:  
  
1. Cloud Deployment – Replace local JSON persistence with AWS DynamoDB and deploy services on AWS EC2.  
2. Additional Microservices – Develop Order and Delivery services, integrating them with inventory.  
3. Scalability and Testing – Simulate multiple sensors and loads; perform load and performance testing.  
4. Final Deliverables – Produce comprehensive final report and record a demonstration video.

## 6. Conclusion

The project has reached a key milestone with a fully working local prototype. The complete cycle of sensor simulation → Node-RED event processing → inventory microservice update has been demonstrated. The foundation is strong, and the next steps will focus on cloud deployment, additional microservices, scalability testing, and final deliverables. The project is on track to meet the SIT314 distinction requirements.