PL Grocery Management System

System Architecture Document – Daniel Allen – Folio Project

---

1. Introduction

The **PL Grocery Management System** automates grocery inventory management by tracking items using an Elegoo Arduino Uno with a numeric keypad to simulate a scanner. The system integrates local preprocessing using a fog node and further processing through an edge node, ultimately storing data in MongoDB Atlas. The front-end interface allows users to view, manage stock monitoring, and reorder items via cloud-hosted services.

---

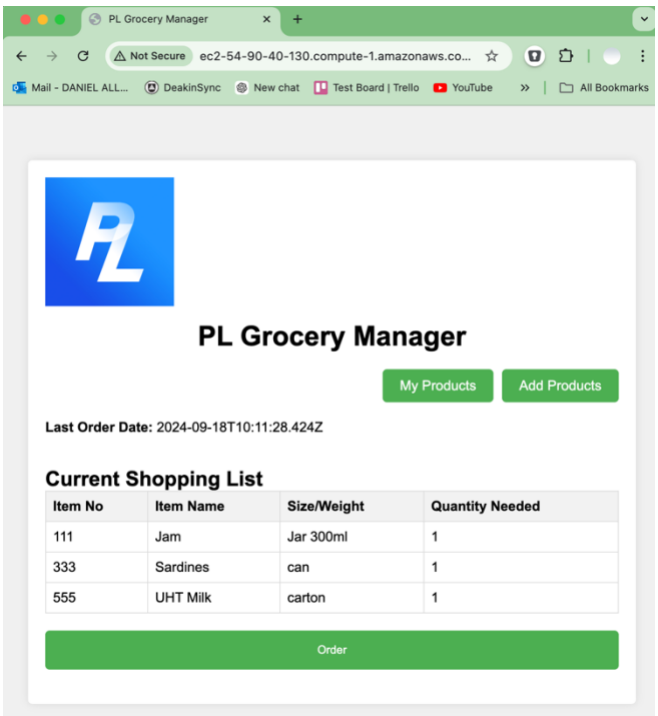
2. System Components

- **Hardware**: The system uses an Elegoo Arduino Uno to simulate a scanner using a numeric keypad. Users input item codes when groceries are consumed.
- **Fog Node**: A fog node preprocesses data locally, solving serial port conflicts and acting as a buffer by saving data to a JSON file.
- **Edge Node**: A Node.js server acting as the edge node processes data from the fog node, validates it, and sends it to MongoDB Atlas.
- **Database**: MongoDB Atlas, a cloud-based NoSQL database, stores grocery data, items, and system settings (e.g., lastOrderDate).
- **Frontend**: HTML, CSS, and JavaScript-based interface, allowing users to interact with their grocery inventory, add products, and place orders.
- **Cloud Deployment**: The system is containerized using Docker and deployed on AWS EC2 for scalability.

---

3. Data Flow

1. **Item Input**: The Arduino Uno sends data (via the keypad) to the fog node.
2. **Preprocessing**: The fog node saves the input to a local JSON file, ensuring system responsiveness and buffering data.
3. **Data Transmission**: The edge node processes the buffered data and sends MongoDB Atlas, which stores the information in relevant collections (groceryitems, items, settings).
4. **Frontend**: The front-end retrieves the stored data from MongoDB and displays it in the shopping list interface. From here you can add new items to monitor and you place an order for the items which updates the current stock levels.



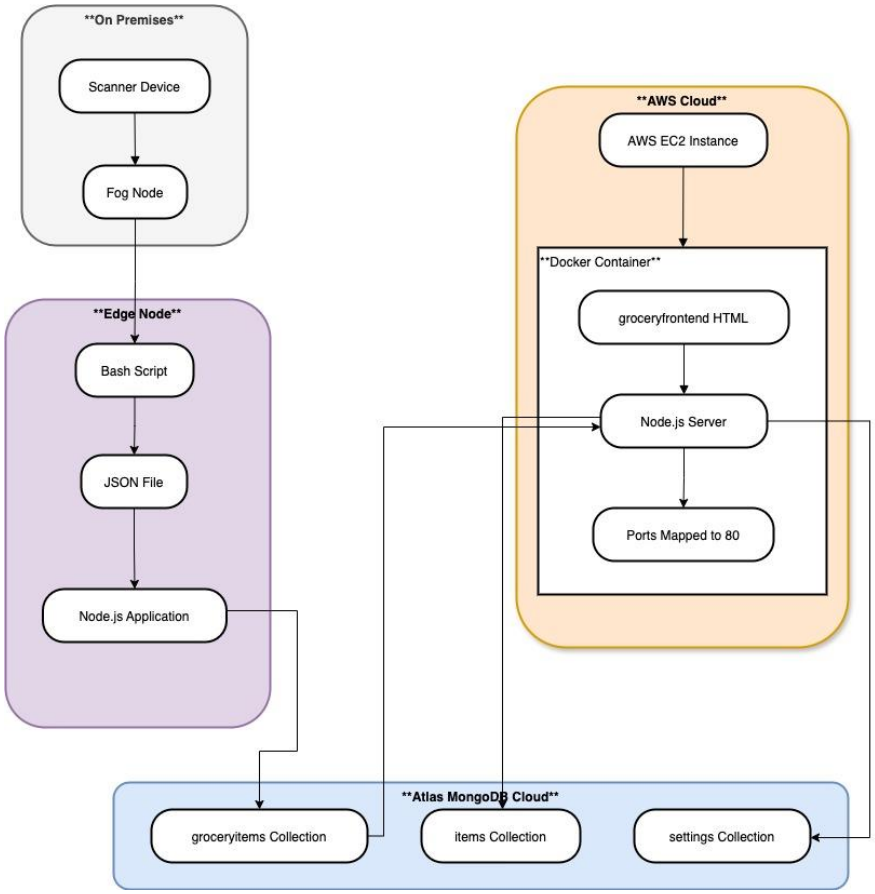
item

it to

4. System Architecture Diagram

This **UML diagram** illustrates the components (Arduino, fog node, edge node, MongoDB Atlas, front end, and AWS cloud infrastructure) and how they interact.

UML Diagram



---

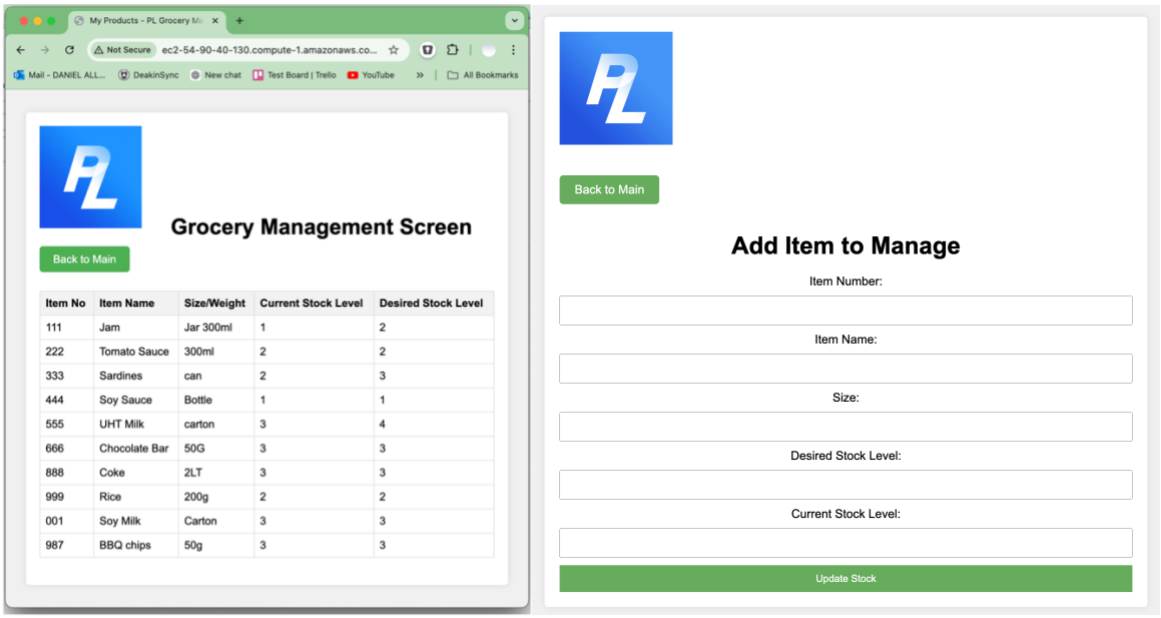
5. Deployment Architecture

- **Dockerized GUI**: The HTML based GUI along with the Node.js GUI-backend, was containerized using Docker to ensure consistent deployment across environments.
- **AWS EC2**: The cloud deployment involved setting up an AWS EC2 instance to host the Docker container. The system uses security groups for access control and public HTTP traffic routing.
- **MongoDB Atlas**: Instead of deploying MongoDB on EC2, or using the AWS version, MongoDB Atlas was used for scalability and alternate database management (AWS would have charged).

---

7. Scalability

- **Cloud Scalability**: Using AWS EC2 and MongoDB Atlas provides horizontal scalability. Docker allows for easy replication of services across multiple EC2 instances, if necessary.
- **Load Balancing**: The system is designed to be load-balanced in future iterations for distributing traffic across multiple servers if needed. Test scaling was achieved creating an image of the instance and creating another instance from that image. Load balancing was set up using the two instances. Atlas MongoDB also has scaling options.

- 

---

6. Security Considerations

- **Environment Variables**: The MongoDB URI and other sensitive data are managed through environment variables stored securely in .env files.
- **Port Security**: EC2 security groups were configured to restrict access, only allowing necessary HTTP and SSH traffic for administration.
- **Processed Data Flags**: To avoid redundant data processing, a processed flag is set on each grocery item stored in MongoDB, ensuring only unprocessed items are flagged for further actions. The database records are all retained for purposes of creating insights data functionality in the future.

---

7. Demo Video

https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=6ebda2a1-8227-4fca-a4cf-b1ef00b36d87