**IoT Use Case: Smart Container**

# Overview

The smart container serves as a critical component within the cold chain logistics system, specifically tailored for temperature-sensitive transportation scenarios. Its primary goal is to deliver a steady stream of accurate, real-time data with minimal delays to external applications. This data encompasses live readings of temperature, humidity, and precise GPS coordinates.

For instance, consider a situation involving blood transportation: At Groote Schuur Hospital in Cape Town, there arises an urgent need for a patient requiring a blood type O transfusion. Unfortunately, the hospital's current supply falls short. In response, they promptly reach out to their closest blood bank for an emergency provision. The blood bank, recognizing the necessity for a controlled environment, entrusts the blood to our smart container, ensuring it is maintained at a precise condition.

Throughout the transportation process, both the hospital and the blood bank have the ability to closely monitor the container's location and temperature status in real-time, providing an extra layer of assurance.

Moreover, our system doesn't stop at ensuring the right conditions for blood transport. We go the extra mile by optimizing the route, selecting the most suitable container, and calculating the necessary amount of ice. This guarantees that the blood arrives in optimal condition, ready for immediate use.

But we don't stop there. We understand the importance of a safe handover. That's why we ensure a secure transfer of goods, utilizing a dedicated driver who confirms the delivery. Additionally, the client receives a confirmation upon receipt, adding an extra layer of reassurance for all parties involved.

# Technical Requirements and Boundary conditions

## Requirements / User Stories

* Route Optimization
  + Find the suitable Driver (algo)
  + Find the shortest way (TomTom)
* Visualization:
  + of real-time position track for client and admin
  + current temperature in the cooler box
* Notifications
  + e.g via sms or mobile app when temperature drastically changes
  + e.g when destination is reached
* Safe over-hand
  + pick up goods from driver = right driver and right good
  + deliver to customer = right good to right customer)

Security Aspects:

* Transportation layer: private cluster for access via Broker, no public access to subscribe for data
* DB access only from server (user, pwd on server side)
* Access to server from flutter app through JSON webToken
  + The JWT is a compact, URL-safe means of representing claims to be transferred between two parties. Basically, it used for authentication, and it is digitally signed to ensure authenticity.
  + **Creating the Jwt**: When the client logs in, the server generates a JWT and sends it to the client. The server signs the token with a secret that only it knows.
  + **Sending the Jwt**: The client stores the JWT, typically in a secure location like a browser’s `localStorage` or `HttpOnly` cookie. It then includes the JWT in the header of subsequent HTTP requests to the server.
  + **Verifying the JWT**: When the server receives a request with a JWT, it verifies the token’s authenticity by checking the signature using the same secret key. If the signature is valid, the server processes the request.
  + **Extracting Information:** The server can extract information from the JWT’s payload, such as user ID or roles, to determine access permissions or perform other operations.
* Use web Token, use key on server 🡪 For more information, read [Understanding JSON Web Tokens (JWTs) in Node.js](https://medium.com/@maselaandile/understanding-json-web-tokens-jwts-in-node-js-4b5e72fe7fee)

## Hardware

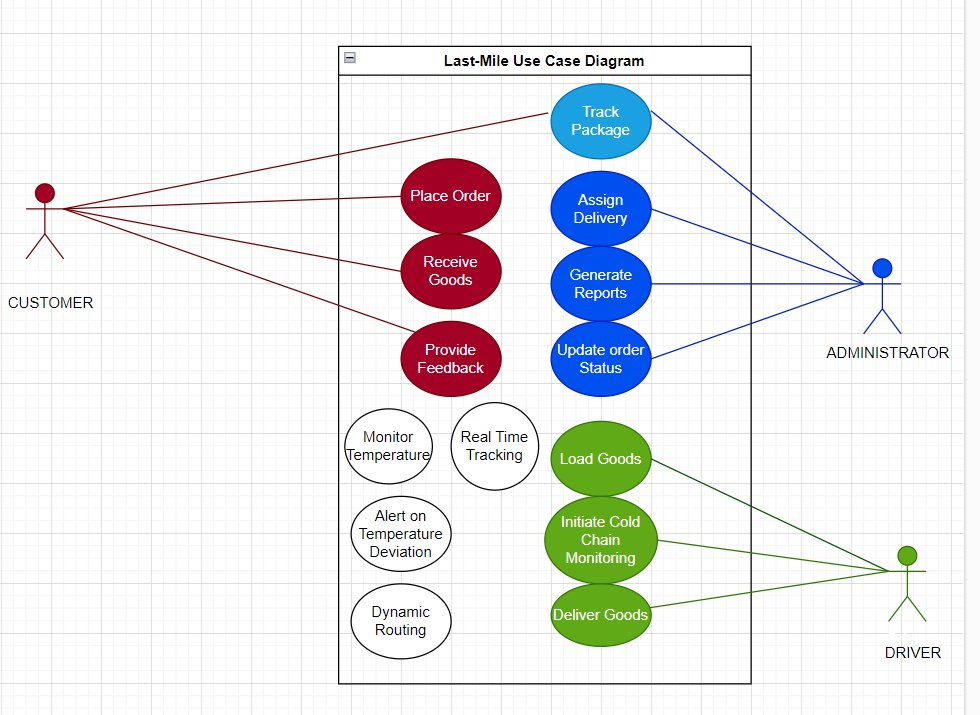
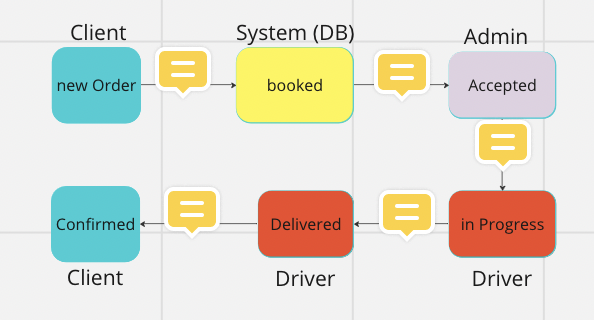
* Arduino ESP8266
* NEO-6M / NEO-7M GPS module
* Temperature and Humidity Sensor module
* Insulated container
* App: tested in android but possible in ios

## Recommended software development environment

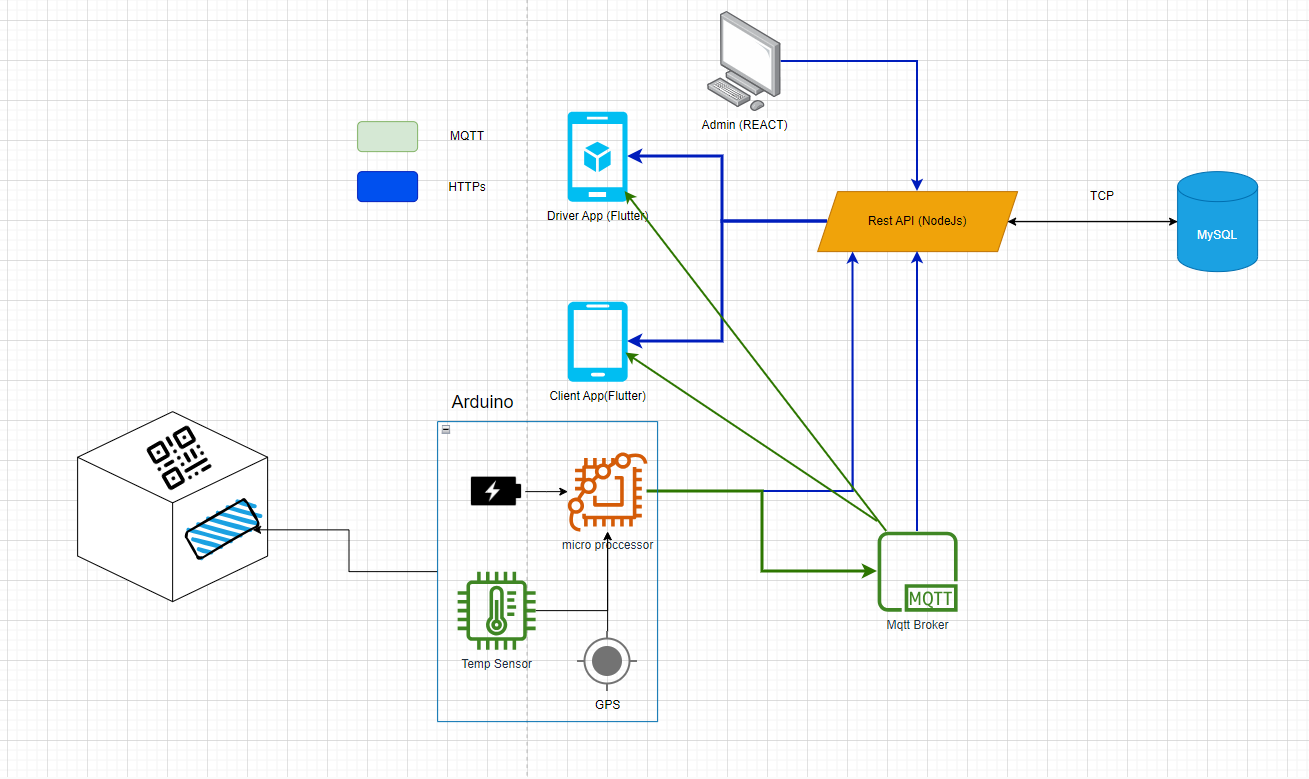
* Sense Layer: Arduino IDE
* Transportation Layer: HiveMQ for MQTT tests
* App Layer / Front End
  + Flutter (Mobile apps) 🡪 Visual Studio Code or Android Studio
    - Flutter Maps
  + React for admin (web-based application) 🡪 Visual Studio Code
* Backend
  + Node.Js 🡪 Visual Studio Code or Android Studio
    - Sequelize: Model definition, Validation, Association
    - Express: routing, middleware
    - Bcrypt: password hashing
    - JsonWebTokens: User Authentication
  + Database 🡪 mySql

# Design

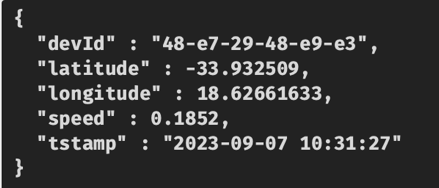
Use case Diagram & State Machine



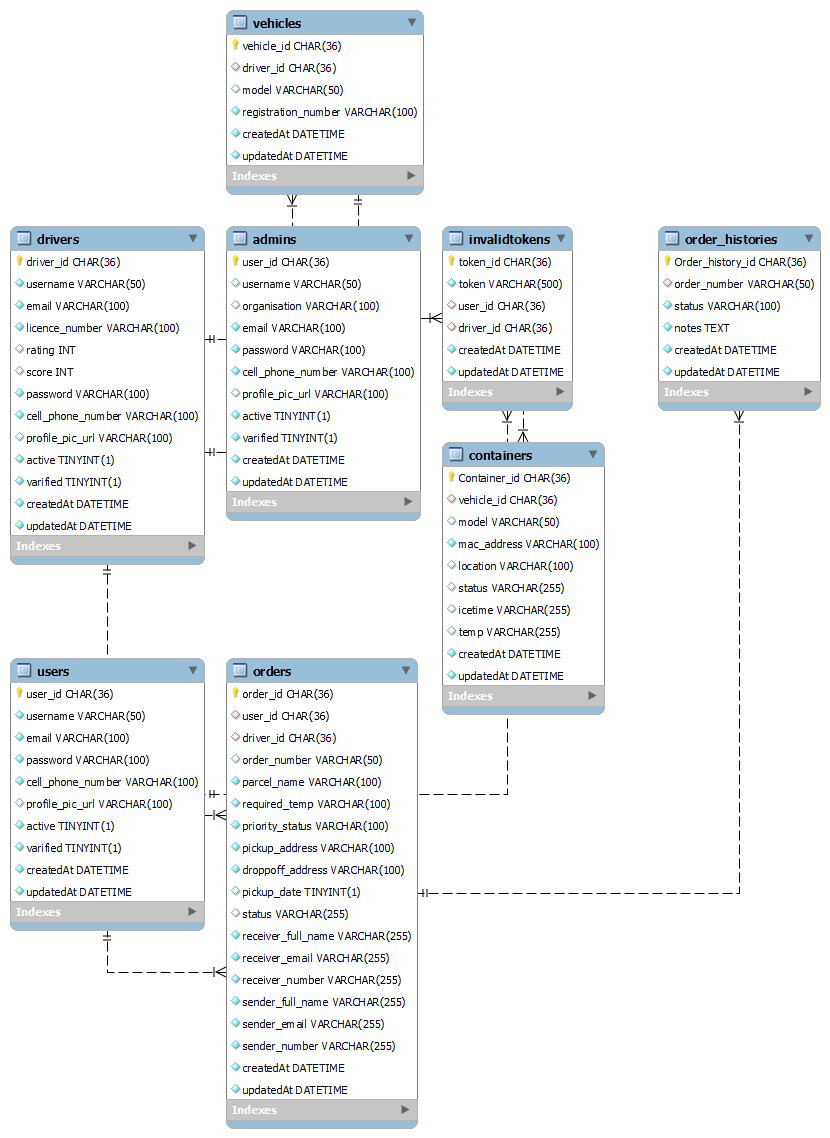
Proposed architecture – Component Overview



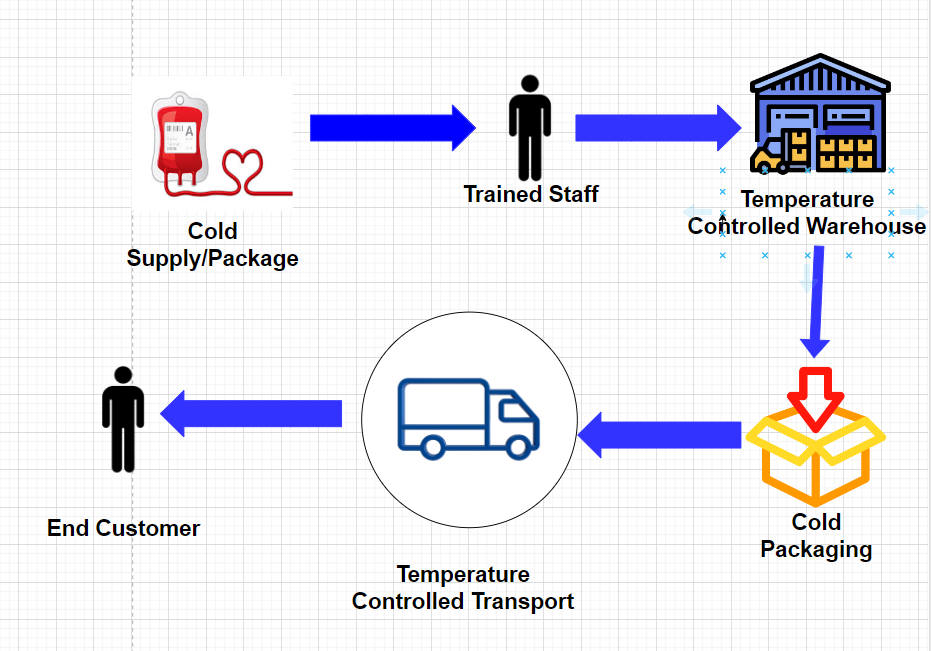
## Data Model for json-Object for IoTContainer



## Data Model for DB



## Process Model / Activity Diagrams



More in detail 🡪   
Activity Diagram on Miro 🡪 [Smart Container](https://miro.com/app/board/uXjVNRGzWBU=/)

# Implementation

**REST API Documentation for Node.js**

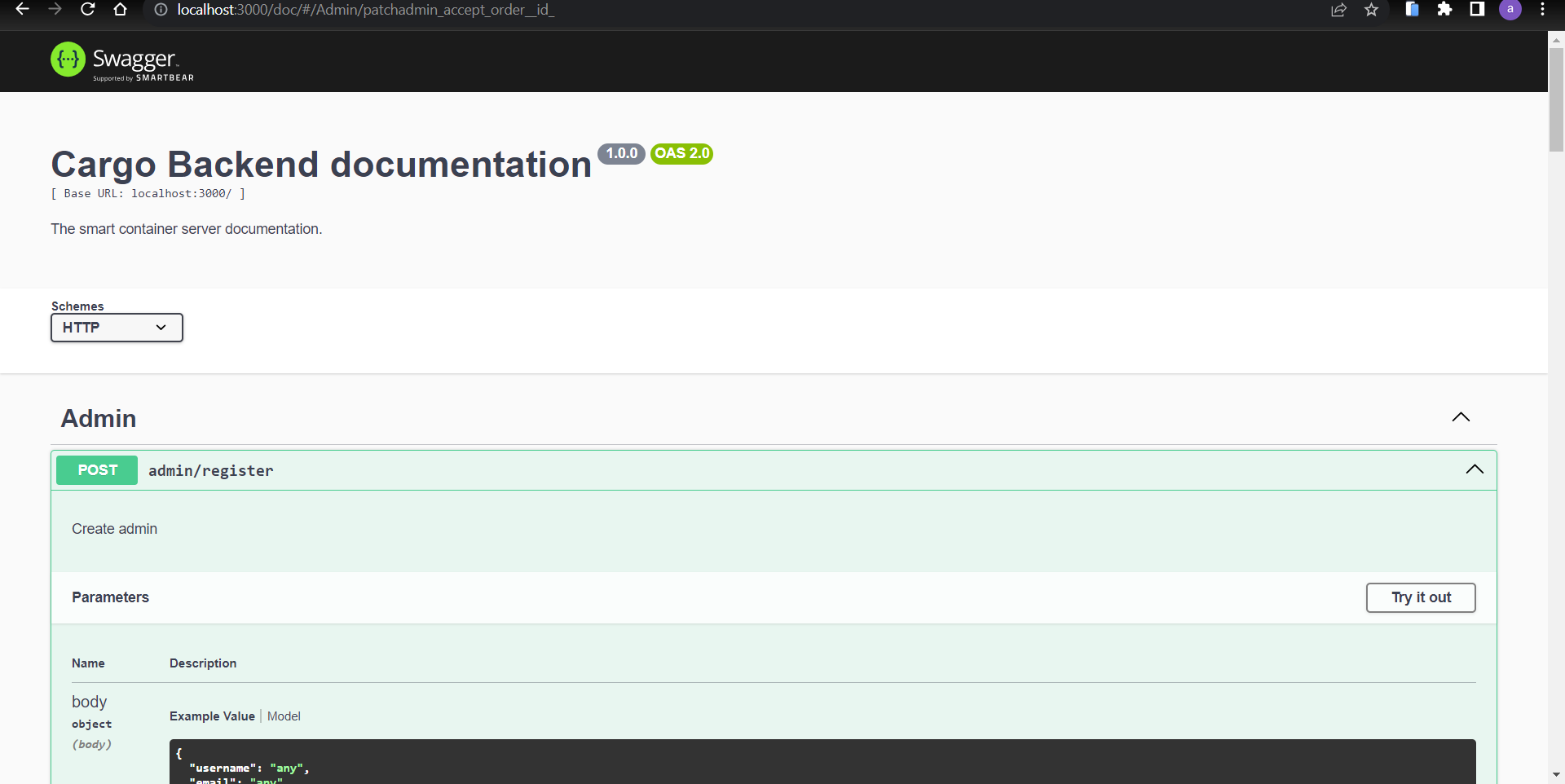
GitHub Repository:

https://github.com/CS-UWC/Last-Mile/tree/master/cargo\_backend

After running the server, visit the following endpoint to access the Node.js documentation in Swagger:

*server\_address>/doc*

This will display the Swagger documentation for the Node.js server.



How you use the TomTom-Api 🡪 https://developer.tomtom.com/documentation

## Smart Container

Github repository: https://github.com/dany-meyer/uwc\_tests/tree/main/Smart\_Container\_Arduino

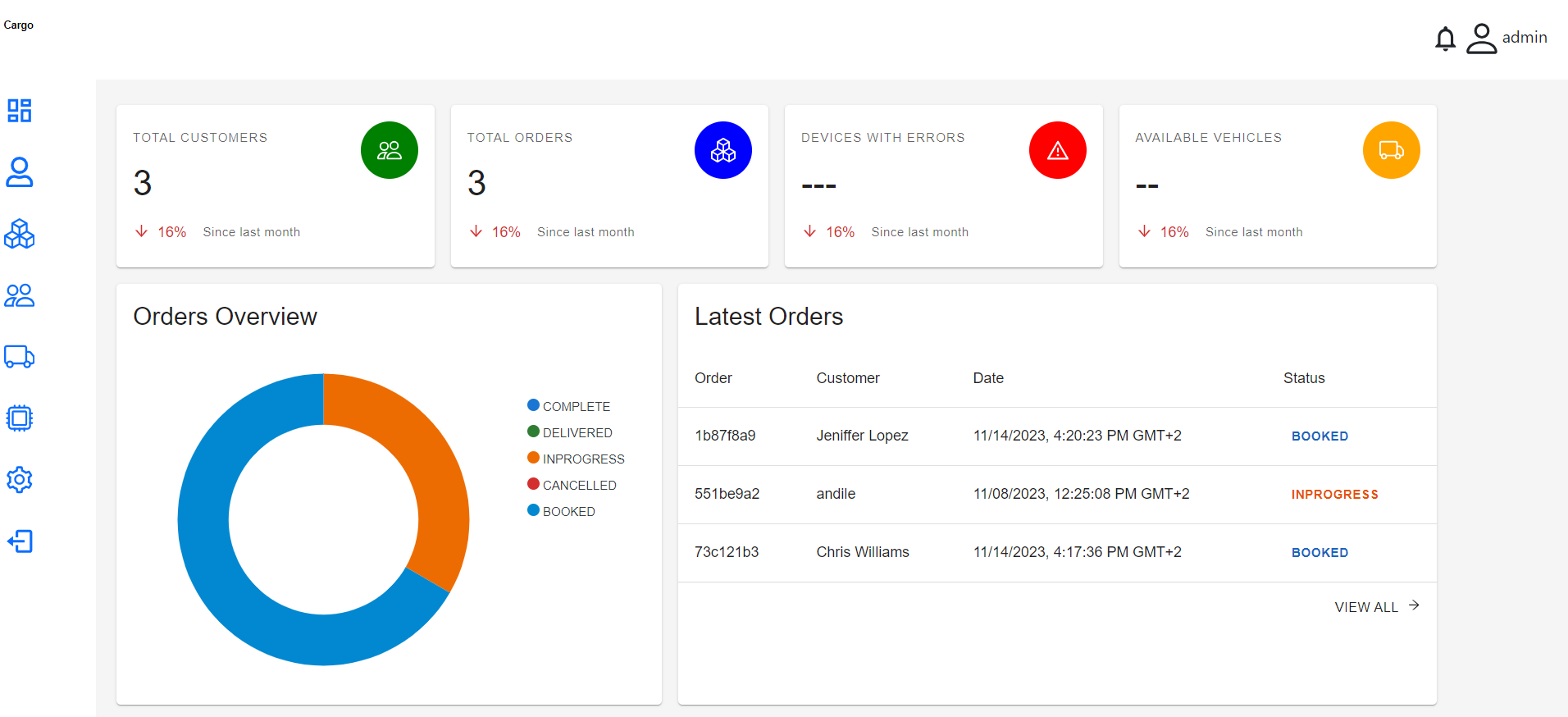
## Driver App

Github repository: https://github.com/CS-UWC/Last-Mile/tree/master/driver\_cargo

## Admin App

Github repository: https://github.com/CS-UWC/Last-Mile/tree/master/cargo\_admin

## Dashboard



# Test

## Modul Test

**NEO-6M/7M GPS Module**

The GPS module is rather inconsistent in picking the satellite signals to generate the coordinates. Often time the GPS module must be exposed to the sky to get satellite signals this may be a suitable solution in highly flexible testing environments but becomes an issue when the module is closed inside the container.

The power supply to the module does not have many inconsistencies but does require a stable power supply as it has its own battery.

**Temperature/Humidity**  
The above modules only require a steady connection to the correct pins do not have any other inconsistencies associated with them.

## Integration Test

Special Cases test

## Final Acceptance Test

Screen Shots

## Sources to learn from the Project

1. Already known
   1. Create Nodejs 🡪 [Tutorial](https://docs.google.com/document/d/1n7ki2LtqUOfzIHMMGSFyx_DYHL4wFl9GfklpUZcXEjU/edit)
   2. Programming the micro controller 🡪 [Smart Container](https://github.com/dany-meyer/uwc_tests/blob/main/Smart_Container_Arduino/)
   3. Create Flutter App 🡪 [Tutorial](https://docs.google.com/document/d/1uouZoHGFGccB8y8lW0MdrkQN0MzIstcZf64jOIO8a0w/edit)
   4. mySQL DB
2. Create Security with node.js using Jason Web Token 🡪 [Understanding JasonWebToken](https://medium.com/p/4b5e72fe7fee)
3. Program Database using js 🡪 [configuring sequelize for nodejs](https://medium.com/@maselaandile/configuring-sequelize-for-mysql-in-node-js-3a25962d41e0)
4. Flutter and web Token 🡪 web link
5. Private cluster for MQTT🡪
6. Learn the basics in react 🡪
   1. <https://www.javatpoint.com/reactjs-tutorial>
   2. https://www.freecodecamp.org/news/react-tutorial-build-a-project/
7. TomTom-API 🡪 https://developer.tomtom.com/documentation