Alexandria – A Real-Time Data Analyzer

Spring 2025, SDET101-102 final Project Report,   
Prof Craig Iaboni  
By Ricardo DaSilveira, rcd4@njit.edu

# Abstract

The Alexandria project is a real-time web application designed to analyze and visualize data collected from IoT-based test systems for automatic transfer switches (ATS). This browser-accessible dashboard allows users to view and filter data like voltage, coil current, and transfer time. Built using Python Plotly Dash and Flask, and an SQLite database, Alexandria helps engineers monitor test performance live and explore historical test records. The project emphasizes efficient local data storage, intuitive interface design, and responsive visualization of test metrics.

# Introduction

## Problem Statement

Research and Development (R&D) laboratories generate enormous amount of data daily from testing. This data is often complex, and produced at high speeds, making manual handling impractical and error prone. To harness the full value of this data, there is a critical need for automated systems that can collect, store, and classify it in real time. Such systems not only ensure data integrity and accessibility but also lay the foundation for advanced analytics and machine learning. Moreover, intuitive, and user-friendly interfaces are essential to allow researchers to visualize trends, explore datasets, and derive insights without needing deep technical expertise. By streamlining data management and analysis, R&D labs can accelerate discovery, improve reproducibility, and make more informed decisions. Since the application is intended to operate within a secured laboratory network where all users have unrestricted access to its data, user authentication and encryption implementation is not critical for its initial functionality. As such, these security features have been deferred to a later stage of development to prioritize core functionality and streamline deployment.

## Objectives

* Create a real-time dashboard for visualizing test data from IoT devices.
* Allow users to apply filters to analyze specific test cases.
* Enable live data updates using lightweight and easily deployable technologies.
* Ensure the application is responsive and easy to use on standard desktop browsers.
* Offer tools to allow data to be saved in for further use.

# Implementation Details

## Technologies Used

* **Plotly Dash**: An open-source Python framework designed for building interactive, web-based data visualization applications. It combines the power of Plotly for rich, high-quality visualizations with the flexibility of Flask for web development and React.js for dynamic user interfaces. Dash allows users to create complex dashboards and analytical tools using pure Python, without needing extensive knowledge of front-end technologies. It supports real-time updates, user inputs, and callbacks, making it ideal for data scientists and analysts who want to turn their data insights into shareable, interactive web apps quickly and efficiently.
* **Flask**: A lightweight Python web framework that makes it easy to build web applications by providing tools for routing, templates, and handling requests, all while remaining simple and flexible for developers.
* **React.js**: React.js is a JavaScript library for building fast, interactive user interfaces, by using reusable components and efficiently updating the UI when data changes.
* **MQTT**: The Message Queuing Telemetry Transport is a lightweight, publish/subscribe messaging protocol designed for low-bandwidth, high-latency, or unreliable networks. It’s ideal for IoT applications, allowing devices to efficiently send and receive data through structured topics, with support for different levels of message delivery reliability.
* **Pandas**: Widely used in data science and analytics, Pandas is a powerful Python library. It provides easy-to-use data structures that allow users to clean, filter, transform, and analyze data efficiently.

## Core Features

* **Real-Time Dashboard:** The dashboard graph and data table are automatically updated as new data arrives.
* **Dynamic Filters:** Data can be filtered by date and time ranges, and transfer direction.
* **Interactive Graphs**: Users can pause live updates, zoom in and out and change the x-axis data type to facilitate data analysis.
* **Data Table View**: Displays a detailed, structured table of the filtered data, allowing users to easily review, and interpret specific values.
* **Export tools**: Data Table values can be easily exported as a “csv” file type, and the line plot can download as a high-quality PNG image for reporting or further analysis.

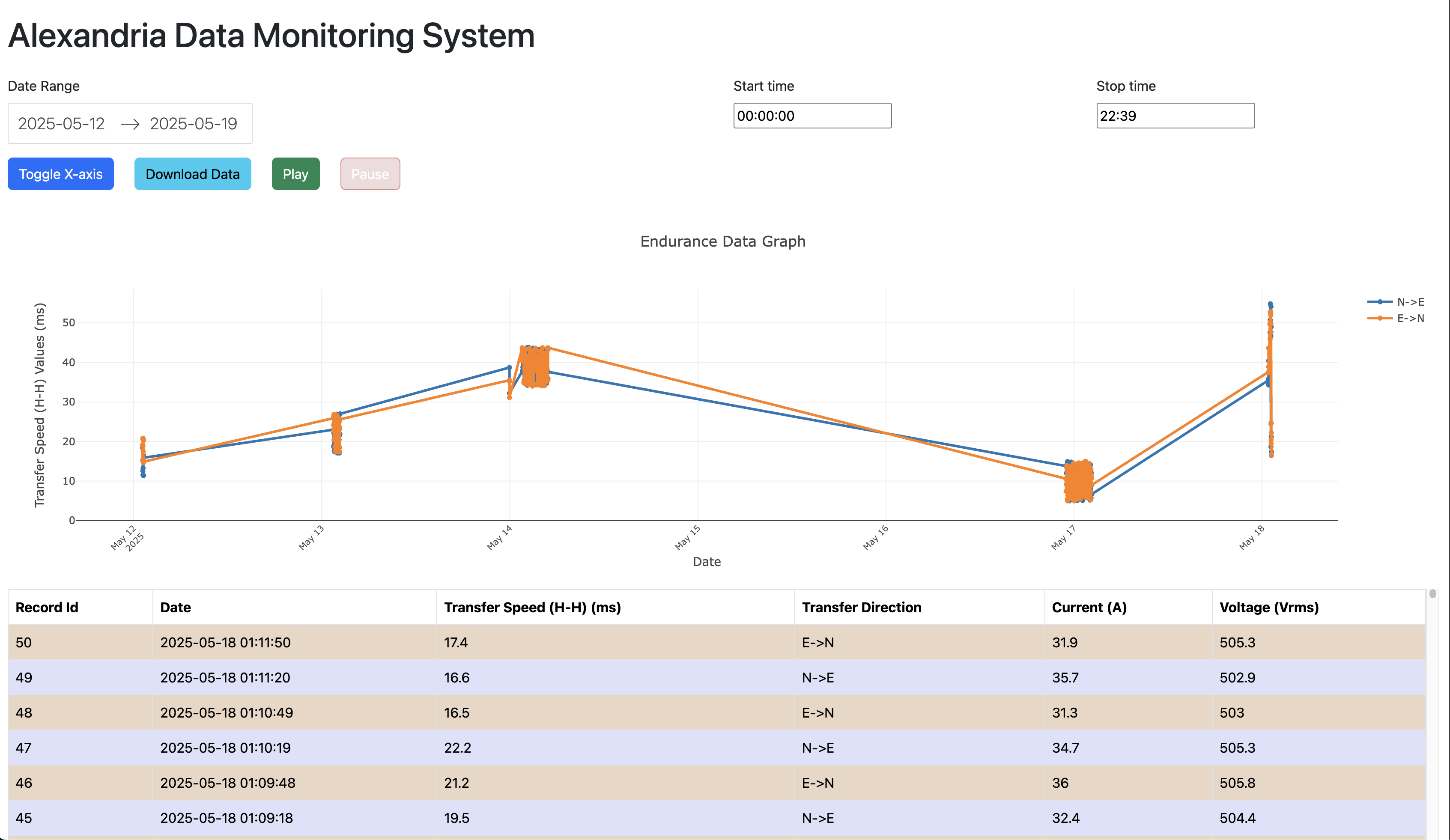


Figure - Dashboard Overview

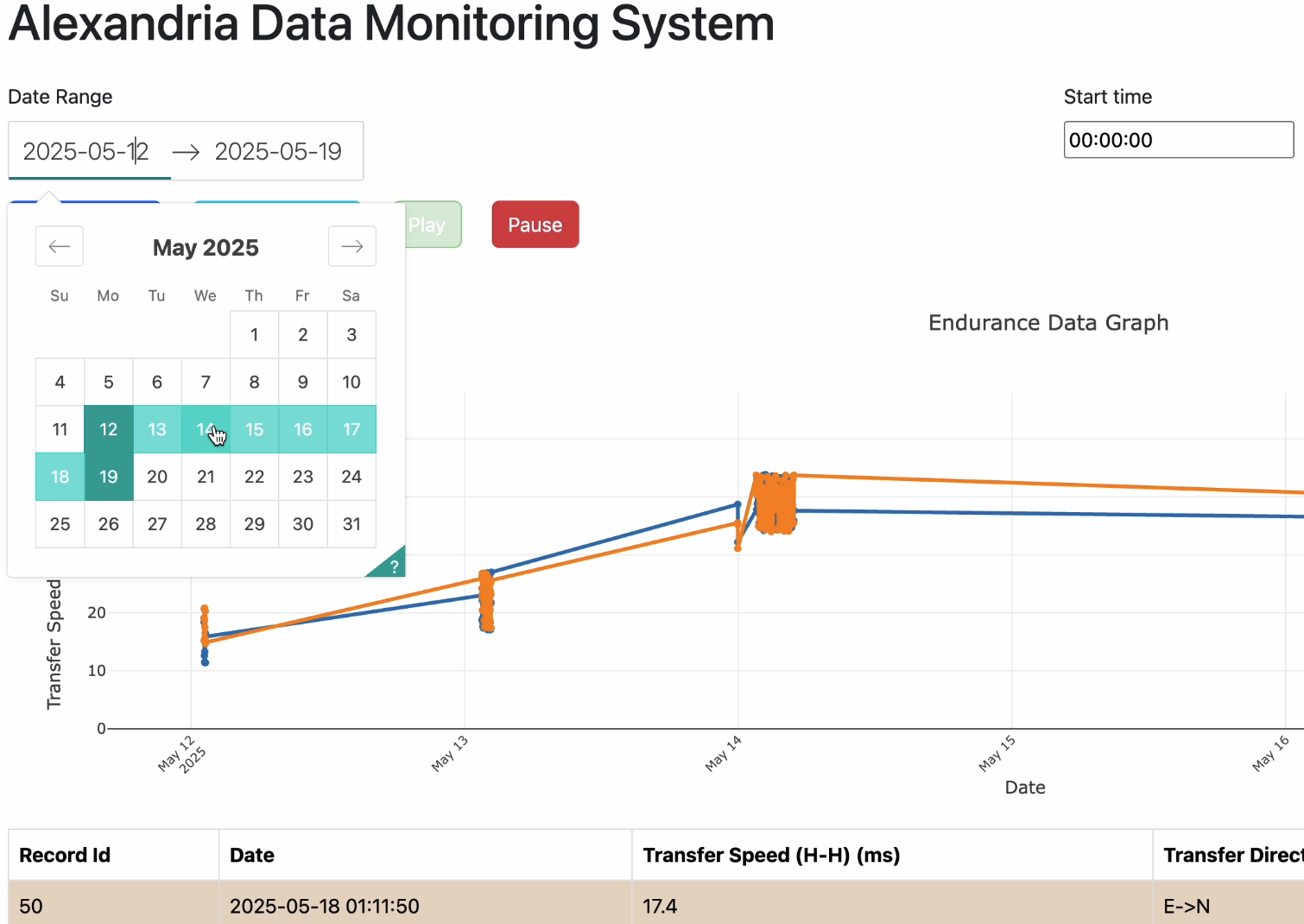


Figure – Filtering data using the “Date Picker” functionality.

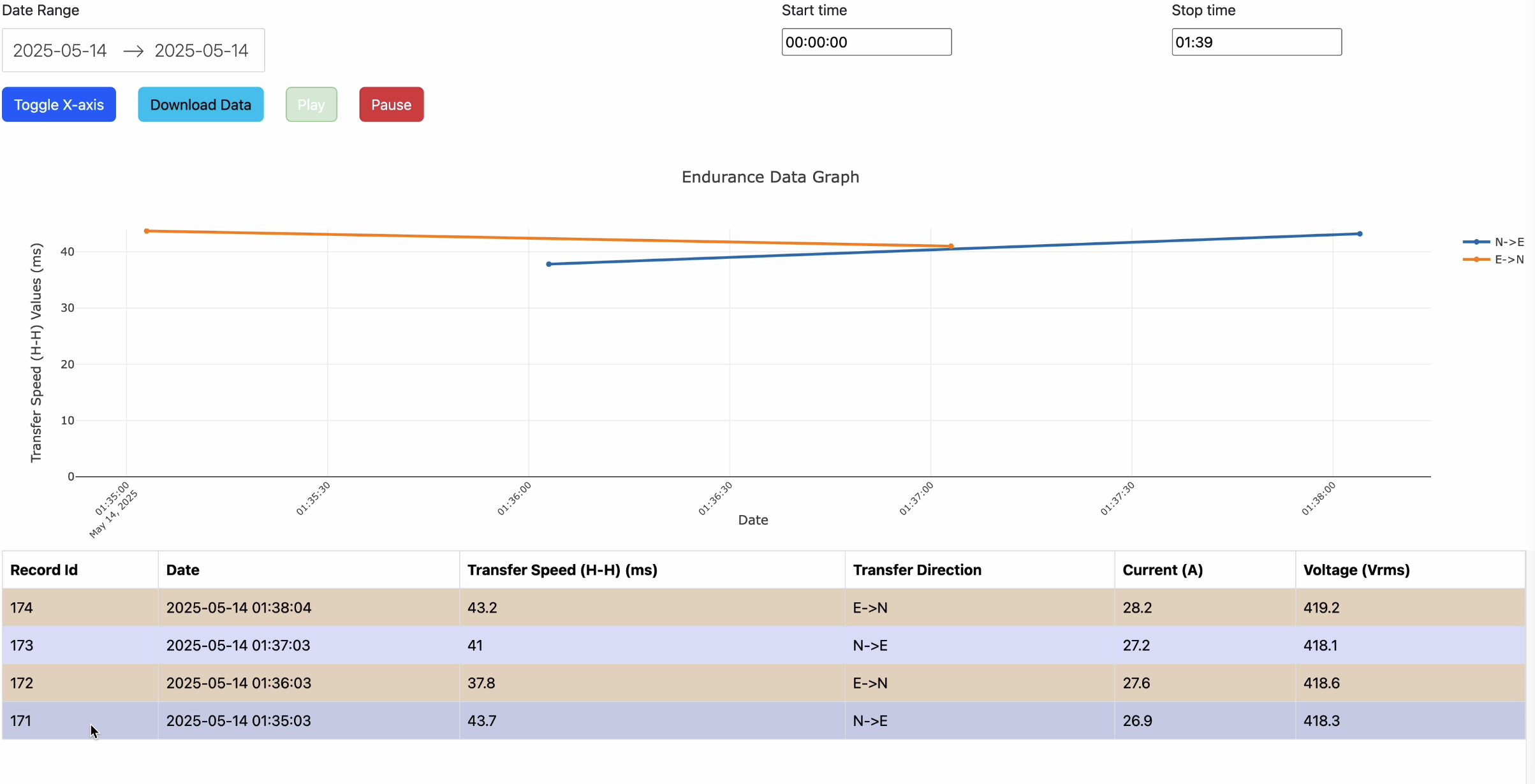


Figure - Updated Line Plot and Table with filtered data.

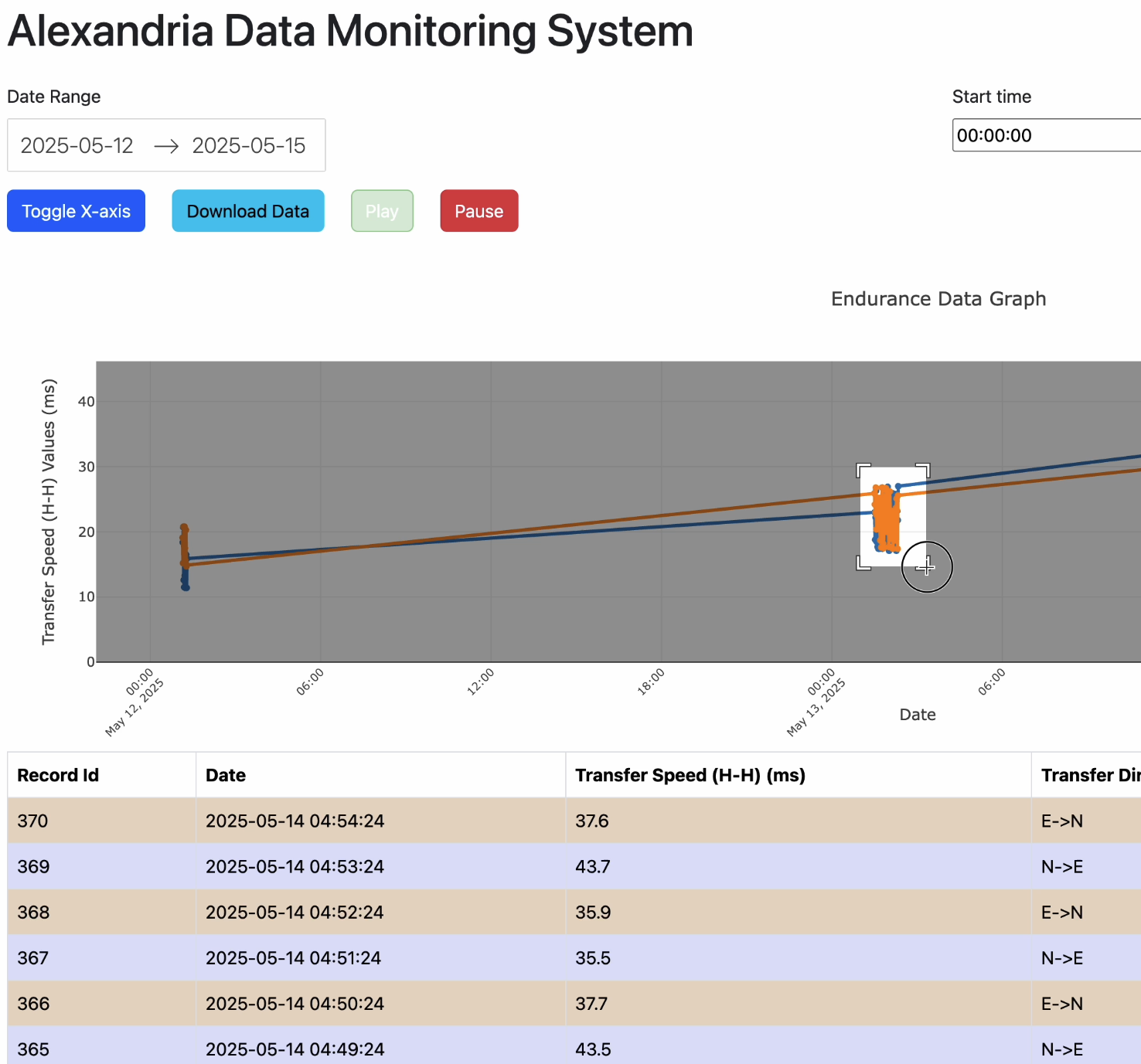


Figure – Graph zoom feature.



Figure – Plot update paused and zoomed in data section.

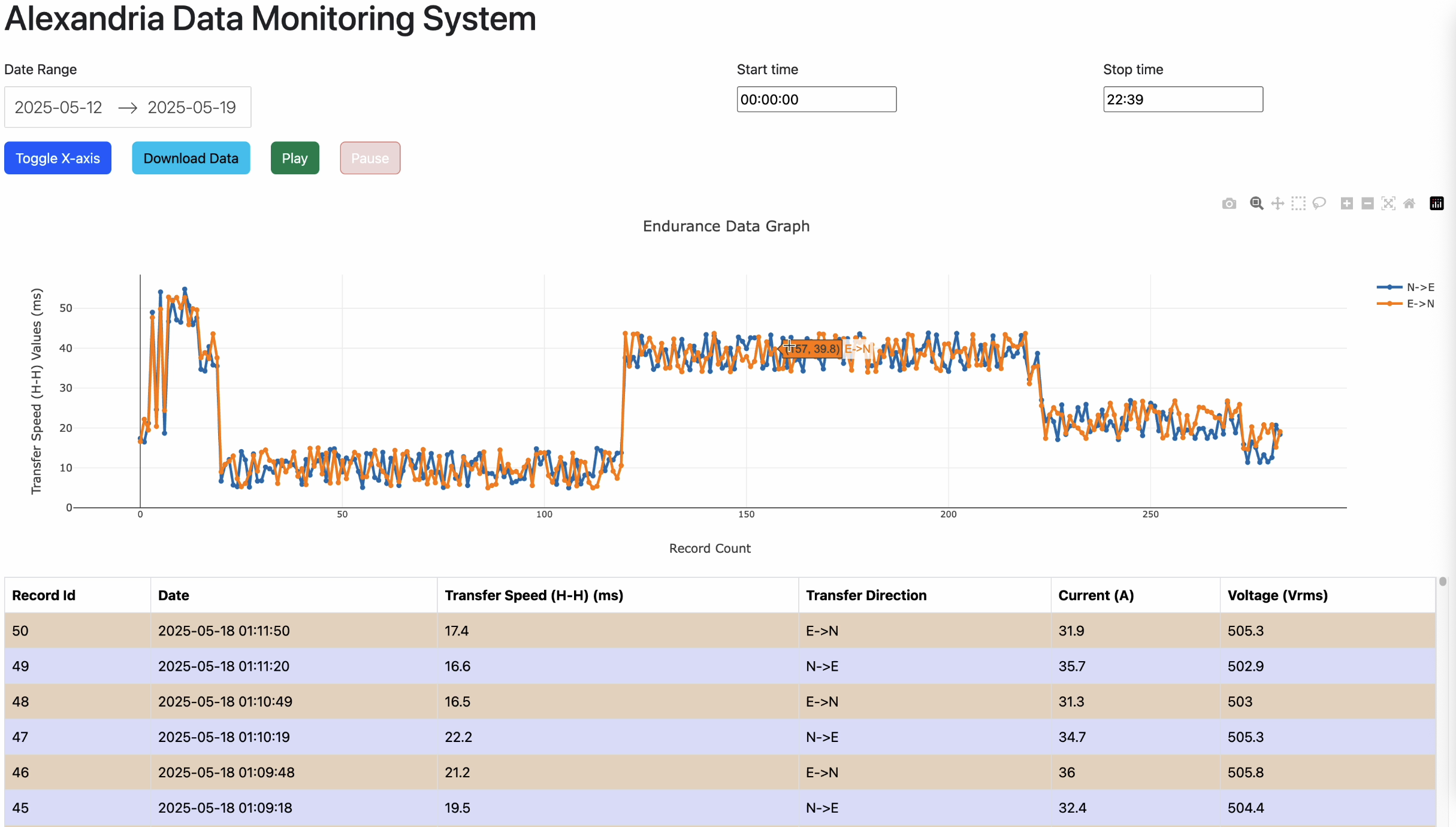


Figure – Axis section changed from “Date” type to “Numerical” type.

## Architecture Diagram

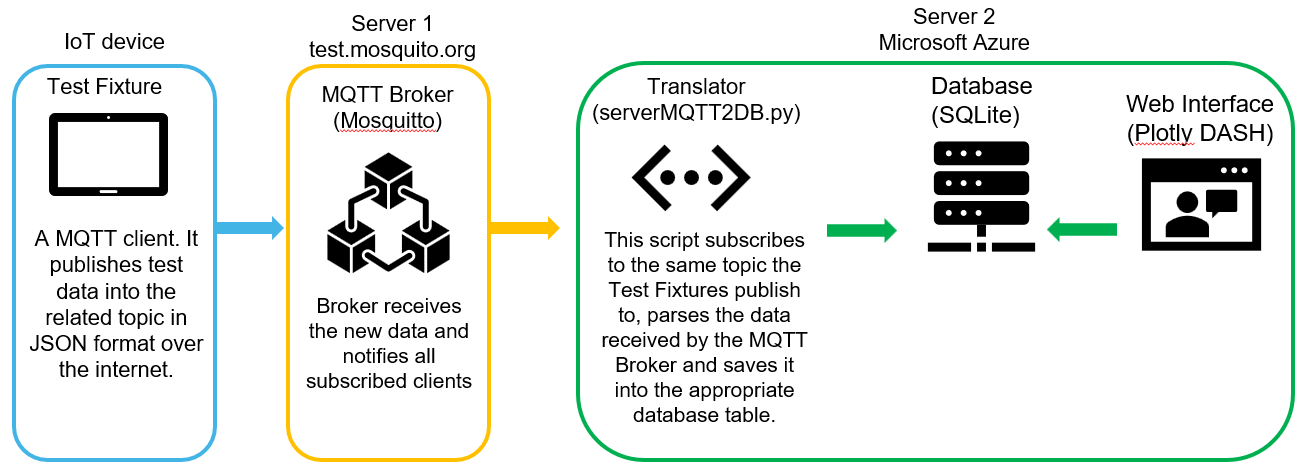


Figure 7 - System Architecture Overview

## MQTT Broker and topics structure and Database Schema:

The topic name will match the table in the database where the data will be stores, this will facilitate future expansion of the system where data for different types of tests will be separated by topic and table. The fields withing the topics will also match the table fields, further facilitating the development of new features.

## 

Figure 8 - Database Schema matching MQTT broker topic structure.

# Challenges Faced and Solutions Implemented

The development of the application came with its fair share of challenges, each adding a layer of complexity to the development process. Ensuring all components of the project communicated properly together in different hosts setup required significant troubleshooting and adaptation.

Below was a list of notable challenges, from easiest to most challenging:

* **Handling Dates and Time Types:** Managing different formats and ensuring consistency across time zones added complexity to data processing and display.
* **Filtering the Data:** Implementing efficient and responsive data filtering required careful logic to maintain performance, especially with larger datasets.
* **Aligning GUI Elements:** Designing a clean and user-friendly web interface using the dash bootstrap components was a steep initial learning curve and time-consuming but essential for usability.
* **Configuring the Database:** The original plan to use MariaDB was ultimately abandoned due to the complexities of configuring and managing a full database server on a cloud hosting platform. Instead, SQLite was chosen for its simplicity and ease of integration.
* **Hosting the Application:** Hosting was particularly complex due to the app’s multiple component architecture. Dash-specific hosting services lacked support for databases or MQTT brokers, while others that could host those services, had a very steep configuration learning curve. Ultimately, due to time constraints and a fast-approaching deadline, I decided to host the application on my home network using a Raspberry Pi device I had previously used for another class. This solution allowed me to run all required services in a single server, greatly speeding up deployment and allowing me to meet the project requirements within the available timeframe.

# Conclusion

## What I Learned

Throughout this project, I gained valuable experience in full-stack web development and deployment. I learned how to build interactive web dashboards using Plotly Dash, using the Dash Bootstrap Components to design clean and responsive layouts. I also got exposed to the challenges of server configuration for web hosting, discovering just how complex and time-consuming it can be to set up a reliable environment. It also led me to evaluate various cloud hosting platforms and their offerings, deepening my understanding of infrastructure-as-a-service options. On the backend, I learned how to install and configure databases, its schemas and use Python to store and retrieve data efficiently from it. Additionally, I became familiar with the MQTT protocol, understanding how it facilitates real-time communication between IoT devices. Altogether, this experience gave me a comprehensive view of developing and deploying modern web applications.

## Potential Future Enhancements

* Additional filtering options and plot types.
* User profiles to save dashboard preferences.
* System usage logging for system analysis
* Communication and storage encryption.
* Dashboards for different types of tests.
* Email alerts and alarms using user defined criteria.

# References

- Dash documentation: https://dash.plotly.com/  
- SQLite Python: https://docs.python.org/3/library/sqlite3.html  
- MQTT Protocol: https://mqtt.org/  
- ChatGPT: https://www.chatgpt.com/  
- Paho MQTT Client: https://pypi.org/project/paho-mqtt/  
- Dash Bootstrap Components: <https://dash-bootstrap-components.opensource.faculty.ai/>