University of Nevada, Reno

Advanced Visualizations and User Support for the VISTA Web Portal

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in Computer Science and Engineering

by

Kaden Nesch

Dr. Sergiu M. Dascalu, Co-Advisor
Dr. Frederick C. Harris, Jr., Co-Advisor

August, 2025

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

Kaden Nesch

Entitled

Advanced Visualizations and User Support for the VISTA Web Portal

be accepted in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Dr. Sergiu M. Dascalu
Advisor

Dr. Frederick C. Harris Jr.,
Co-advisor

Dr. Yantao Shen,
Graduate School Representative

Markus Kemmelmeier, Ph.D., Dean,
Graduate School

August, 2025

Abstract

The increasing availability of sensor data presents new challenges in how information is visualized, interpreted, and acted upon. Effective data visualization is essential for ensuring that users can make sense of complex environmental trends and device outputs, especially within platforms that handle large volumes of time-sensitive data. This thesis focuses on enhancing the VISTA platform by developing advanced data visualization tools designed to improve the clarity and usefulness of sensor data for end users. Key features include interactive charts, enhanced selection tools, and a dynamic map view that displays sensor locations in real-time. The new visualization approach introduces new and improved ways to analyze data with expanded filtering capabilities. Alongside these improvements, user support has been expanded through clearer interface design and the integration of responsive features that allow users to interact with data more effectively. Together, these contributions aim to support users in understanding and analyzing sensor data with greater ease and accuracy. The improvements made to VISTA enhance its role as a valuable tool for monitoring and decision-making in sensor-driven environments, with the potential to support future developments in smart agriculture and other related fields.

Dedication

I dedicate this thesis to my friends and family, both in the United States and back home in Bulgaria, who have provided me with the resources and motivation to continue and finish my educational career. I thank my mother and father, who have worked selflessly to put me in a position to succeed, for which I am forever indebted to them. To my grandmother Zlatina, thank you for always letting me know there was a path for me to be successful, regardless of any self-doubt.

Acknowledgments

I extend my deepest gratitude to my co-advisors, Dr. Sergiu M. Dascalu and Dr. Frederick C. Harris Jr., for their unwavering guidance and mentorship throughout my undergraduate and graduate studies. Their wisdom and patience have been instrumental in shaping my academic journey. I would also like to thank my additional committee member, Dr. Yantao Shen, for his valuable participation and time.

To the entire NSF CSSI project group: Thank you all for your collaboration and shared dedication to this work. A special acknowledgment goes to my co-researchers, Nikhil Sharma and Chase Carthen, whose contributions have been instrumental to the foundations for the work presented in this thesis.

This material is based in part upon work supported by the National Science Foundation under grant number OAC-2209806. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, environmental concerns such as climate change and declining water and air quality have driven the need for more accessible and scalable environmental monitoring solutions. Environmental monitoring is defined as the observation and study of the environment, in which scientists collect data to gain insights into Earth's environment [1]. It plays a crucial role in understanding changes in ecosystems and the impacts of human activity on the planet. Traditionally performed by government agencies using specialized equipment, environmental monitoring has evolved significantly with the rise of digital technologies, allowing for new avenues of research, such as academic research projects. As the global threat of climate change and worsening air quality grows, the frequency of extreme weather and climate events has increased substantially. As shown in Figure 1.1, Parmesan et al. [15] present a visualization illustrating the general extinction risk for species worldwide due to the direct effects of climate change. Environmental monitoring enables us to better understand how changes in environmental variables, such as air quality and temperature, can lead to these extreme events and contribute to rising extinction rates. With issues like these posing threats to both current and future generations, data monitoring is becoming exponentially more valuable, as the volume of collected data continues to grow each year.

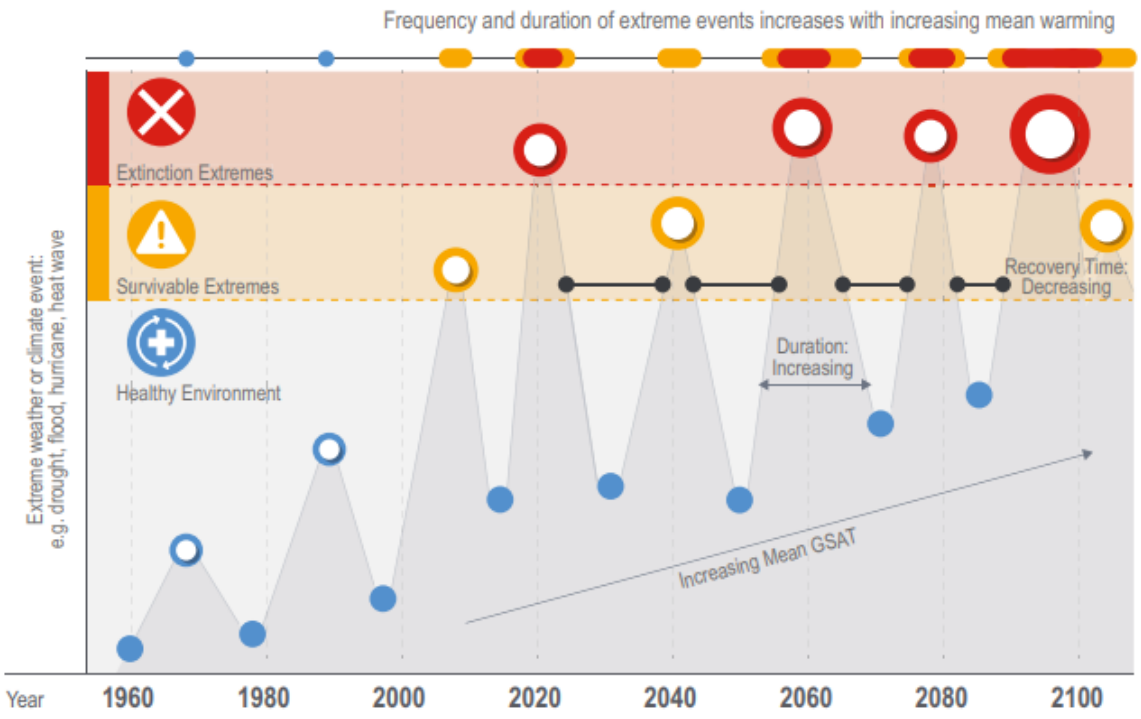

Figure 1.1: "How extinction risk is affected by changes in the frequency, duration, and magnitude of extreme weather or climate events" [15].

The Internet of Things (IoT), in particular, has enabled the deployment of low-cost, low-power sensors capable of collecting real-time data on variables such as temperature, humidity, and soil moisture. These sensors can be distributed across wide areas and configured to transmit data continuously, supporting applications such as early warning systems and academic research projects. These advancements have opened the door not only for scientific research but also for public participation in scientific research known as citizen science [2]. Citizen science initiatives have grown in popularity among individuals and communities, showing an increase in interest, such as documenting climate change impacts and local environmental issues. From tracking outdoor neighborhood temperatures to monitoring soil moisture in personal yards, IoT sensor data has empowered individuals and organizations to make informed decisions based on data. However, the effectiveness of these systems relies heavily on the ability to interpret the collected data through intuitive and meaningful visualiza-

tions. The power of IoT systems shines through the ability to access the data reliably on platforms that provide real-time updates and visualizations, which is the focal point of this thesis.

This thesis focuses on the VISTA (Visualizing IoT Sensor Tracking and Analytics) portal, a platform developed to support real-time monitoring and analysis of environmental sensor data. This platform is developed as part of the research project "CSSI: Elements: Innovating for Edge-to-Edge Climate Services", supported by the National Science Foundation's Cyberinfrastructure for Sustained Scientific Innovation (CSSI) program under grant number OAC-2209806. Built on top of The Things Stack LoRaWAN network server, VISTA enables users to register, manage, and visualize sensor data through a web-based interface. In doing so, it bridges the gap between complex backend infrastructure and the everyday user who wishes to explore environmental data. While the initial version of the portal established foundational features for sensor management, visualization, and data export, a recent user study revealed areas where the platform could be improved to better serve its target audience, which will be discussed in a later chapter. This thesis builds upon that foundation by integrating enhanced visualization features, UI improvements, and system refinements based on direct user feedback.

The remainder of this thesis is organized into chapters as follows: Chapter 2 reviews key topics including IoT technologies and visualizations within the platform, the technical architecture and functionalities of LoRaWAN and The Things Stack, an overview of the VISTA web portal before the thesis enhancements, and the software technologies used in the development of the thesis implementation. Chapter 3 explores similar platforms and tools developed for environmental monitoring and general data visualization. Chapter 4 presents the problem statement and proposed solution for the research. Chapter 5 details the software design and integration of the enhanced visualization features within the VISTA portal. Chapter 6 outlines inferred usage scenarios of the improved platform and how users might interact with it. Chapter 7 covers the specification, setup, procedure, and results of the usability study conducted

as part of this research. Finally, Chapter 8 summarizes the thesis findings and offers directions for future work and discussion of current limitations.

Chapter 2

Background

This chapter provides the technical foundation for the development of the VISTA portal and its underlying infrastructure. Section 2.1 begins with an overview of IoT sensor networks and the importance of data visualizations in effectively interpreting environmental sensor data. This includes a discussion of relevant chart types that are best suited for time-series and geospatial datasets commonly found in environmental monitoring. Following this, the chapter outlines the communication protocols and network server used by the portal, with Section 2.2 covering the LoRaWAN architecture and its implementation within The Things Stack platform. Section 2.3 introduces the VISTA portal, highlighting its core functionalities, user interface structure, and real-time monitoring capabilities. Lastly, Section 2.4 discusses the technologies and software stack used in the portal's architecture, which includes data ingestion, database storage, backend API services, and front-end frameworks that enable scalable, responsive, and user-friendly interactions with environmental sensor data.

2.1 IoT Data Visualizations

The Internet of Things (IoT) is a distributed network of interconnected devices (e.g., computers, sensors) that collect and transmit real-world data [11]. IoT systems can be divided into three architectural stages, each with its own responsibility in maintaining the system's functionality. In the first stage, sensors and actuators collect and transmit real-world data. The second stage is responsible for aggregating and

optimizing the collected data via a gateway-sensor link. Finally, Stage 3 leverages edge computing and cloud systems for advanced tasks, such as machine learning, visualization, and feedback loops, to refine the system [11]. In the scope of this thesis, the ladder stage's application of visualizations will be tackled.

The possible use cases of IoT technologies are rapidly progressing alongside an increased reliance on real-world data being collected. These systems allow for real-time monitoring and analysis in emerging domains such as healthcare, smart homes, and agriculture [6]. Based on the type of data that is collected, the inclusion of a data visualization is imperative for understanding the trends and relationships that exist within a dataset. Thus, selecting a chart type congruent with the collected data is paramount in effectively translating the information to the end-user. An example of this is represented in Figure 2.1, as a unique chart is used for differing fictitious IoT data sets. Some of the goals of charts to keep in mind when making a selection are composition, correlation, ranking, and distribution [3].

- **Composition**: The entirety of a visualization entity, such as a line graph.
- **Correlation**: The relationship between two variables (e.g., the relationship between the average soil moisture percentage and the average amount of rainfall).
- **Ranking**: The corresponding order of items or values. A direct example of this would be arranging clustered columns in a chart from least to greatest.
- **Distribution**: The dispersion of items or values (e.g., the number of clicks on a website for a week measured daily).

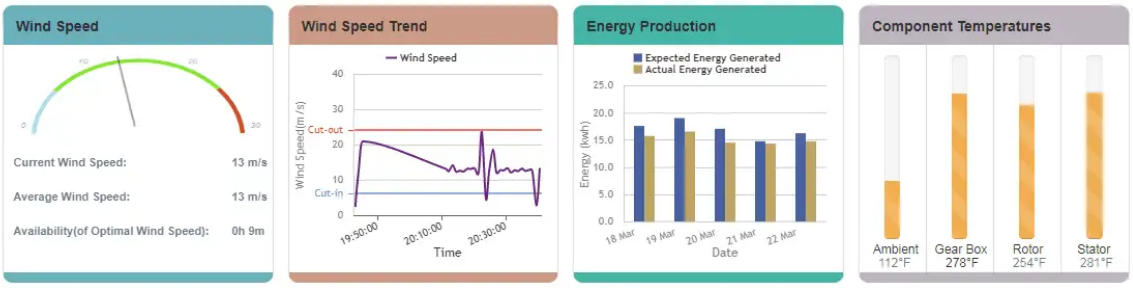

Figure 2.1: This is a set of four fictitious IoT data sets being represented by the most appropriate chart type [17].

With the scope of IoT sensor data in mind, this section also discusses a few select chart types that provide the most value for this type of data, some of which are directly integrated into the VISTA portal as the basis of the thesis. Here are a few examples of chart types, their general description/use case.

Table 2.1: Chart types used for IoT data visualization and their description [3]

Chart Type	**Description**
Line Chart	Displays trends over time by connecting continuous data points with straight lines. This chart type is particularly effective for visualizing time-series data such as temperature, humidity, or sensor voltage levels, enabling users to identify upward or downward trends and recurring patterns quickly.
Clustered Column	Represents data using vertical bars grouped by categories. Each group consists of multiple bars representing different variables, making it well-suited for side-by-side comparison of multiple sensor readings over shared time intervals or categories.
Geographic Map	Displays location-based data over a geographical area, often using pins, markers, or regions to represent physical devices such as sensors or gateways. Useful for visualizing deployment coverage within projects.
Heatmap	Uses color gradients to indicate intensity or concentration of values across space or time. Ideal for representing geospatial data density or frequency, such as signal strength or pollution levels across a region.
Scatter Plot	Shows individual data points on a two-dimensional plane, allowing users to examine relationships or correlations between two numeric variables. Effective for identifying trends, clusters, and outliers in datasets such as temperature vs. humidity or soil temperature vs. soil moisture.

2.2 LoRaWAN & The Things Stack

In order for IoT sensor data to be transmitted from physical environments to end users, a reliable communication protocol and a supporting network infrastructure must be in place. These systems ensure that information collected by distributed devices can be securely and efficiently routed through gateways and servers before reaching the application layer. Within this framework, LoRaWAN serves as a long-range, low-power wireless communication protocol designed specifically for IoT deployments, while The Things Stack functions as a network server that facilitates the management, routing, and processing of this sensor data. The following subsections

provide an overview of both components, highlighting their roles in enabling scalable and robust IoT systems like the VISTA portal.

2.2.1 LoRaWAN

LoRaWAN, a lower-power wide-area networking protocol designed for long-range communication between devices and the internet, operates on a wireless modulation technique stemming from the Chirp Spread Spectrum (CSS) technology named LoRa [10]. LoRa encodes information on radio waves utilizing chirp pulses, which provide a robust solution for long-distance contact with potential disturbances. To further elaborate, LoRaWAN is a Media Access Control (MAC) layer protocol that defines how devices transmit and format their messages. LoRaWAN networks contain a few basic elements that are essential to their overall architecture. Figure 2.2 visualizes this architecture from end-to-end (end devices to application server).

- **End Devices**: Sensors that send wireless messages to the gateways and also receive wireless information back from the gateways.
- **Gateways**: Devices that receive the wireless messages from the sensors and push them to the network server, and vice versa.
- **Network Server**: Software that manages the entire LoRaWAN network, and obtains the messages sent by the gateways.
- **Application Servers**: Software running on the network server that handles processing application data.

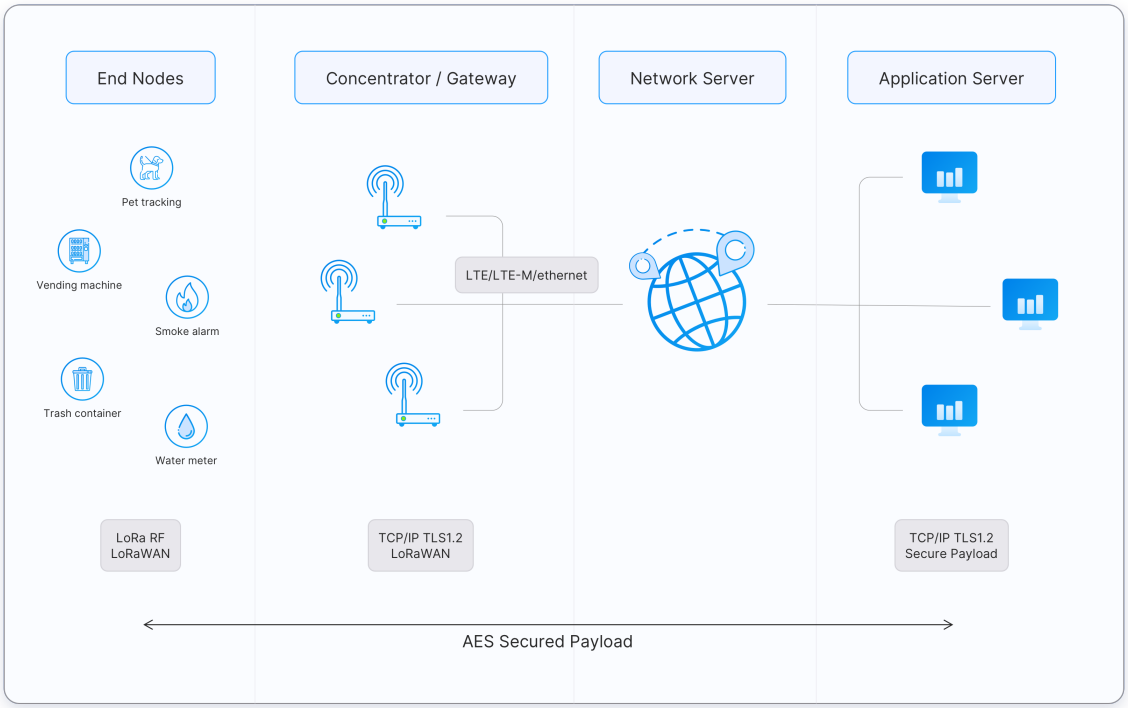

Figure 2.2: The LoRaWAN architecture [10].

2.2.2 The Things Stack

The Things Stack is an open-source LoRaWAN network server that links sensor deployments and records their data [10]. It is built to contain useful tools and services to effortlessly manage LoRaWAN devices such as sensors and gateways. The Things Stack supports multiple interfaces for data integration, including MQTT, Pub/Sub, and gRPC, allowing developers to efficiently route incoming sensor data to external databases, APIs, or client applications.

The platform is designed around modular components that handle different stages of the LoRaWAN lifecycle. These include device provisioning, join-request handling, MAC command scheduling, and downlink queue management. Devices are identified and managed using a combination of unique IDs and cryptographic keys to ensure secure and authenticated communication.

The Things Stack is also compliant with the LoRaWAN specification, supporting both Class A and Class C devices, and enabling advanced features like Adaptive Data

Rate (ADR). Its open-source architecture makes it particularly suitable for research applications and custom IoT deployments, allowing developers to host and extend their version of the network server.

In the context of the VISTA portal, The Things Stack functions as the back-end layer responsible for collecting and forwarding sensor data to the application's database. By using a local deployment of The Things Stack, VISTA developers gain full control over device management, data flow, and integration with external systems, making it an ideal choice for prototyping and experimentation in sensor-based research. In the following section, an in-depth overview of the VISTA portal, including its core functionalities and components, will be discussed.

2.3 VISTA Portal Overview

The Visualizing IoT Sensor Tracking and Analytics (VISTA) portal is a web-based platform designed to support environmental monitoring and citizen science through the deployment, tracking, and analysis of IoT sensor data. Built as a client-facing extension of The Things Stack, VISTA leverages LoRaWAN network infrastructure to provide real-time and historical insights into sensor projects. The portal allows researchers and citizen scientists to register sensors and gateways, associate them with specific projects and organizations, and explore their data through a suite of tools, including geographic maps, dynamic graphs, and exportable reports. Figure 2.3 shows the login page to the VISTA portal. It is designed to support both public outreach and academic research, encouraging transparency and participation in environmental-related data collection. The VISTA portal is divided into several functional areas accessed tabularly, each tailored to different aspects of device and data management. The following subsections provide an in-depth look at these components.

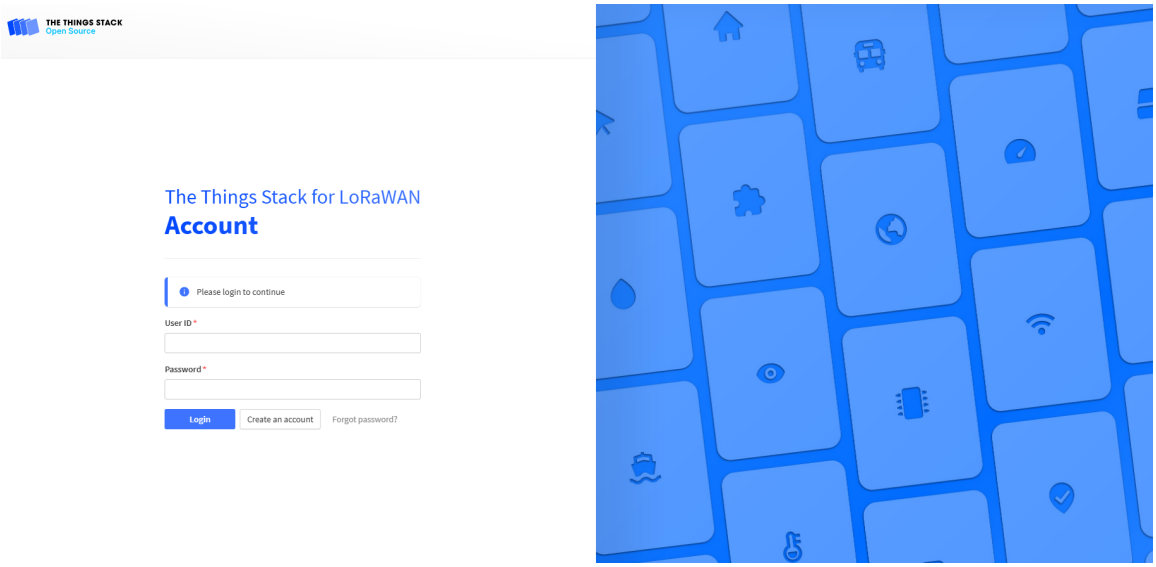

Figure 2.3: Login page. [10].

2.3.1 Projects

The projects tab allows users to classify and group together sensors as a collection called a "Project". Each project allows users to monitor, organize, and analyze data from multiple sensors that are deployed to observe data for environmental or citizen science purposes. Figure 2.4 portrays the create project page, where users are able to create their project by providing information such as the project name and description. Although there is no standardized naming convention within the VISTA portal, having a name that indicates the purpose or location of the project is to be expected. Within a Project, users can view detailed information about each sensor, including its metadata, latest data entries, and connection status. Figure 2.5 showcases the register sensor, which is located within the projects tab. Figure 2.6 identifies the end device overview page within the projects tab. This is where users are able to monitor their sensor in real-time, as The Things Stack natively provides live tracking of payload and status information. This tab also provides access to a suite of tools for exporting data, visualizing sensor trends, viewing sensor locations on an interactive map, and performing other project-level operations.

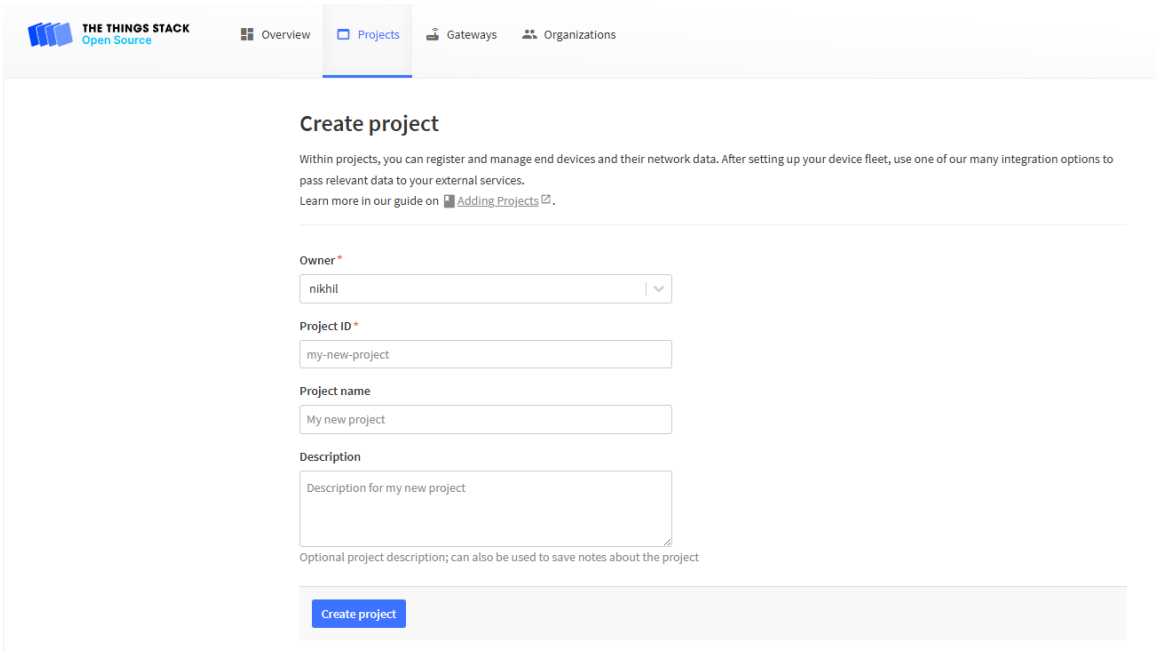

Figure 2.4: Create project page. [10].

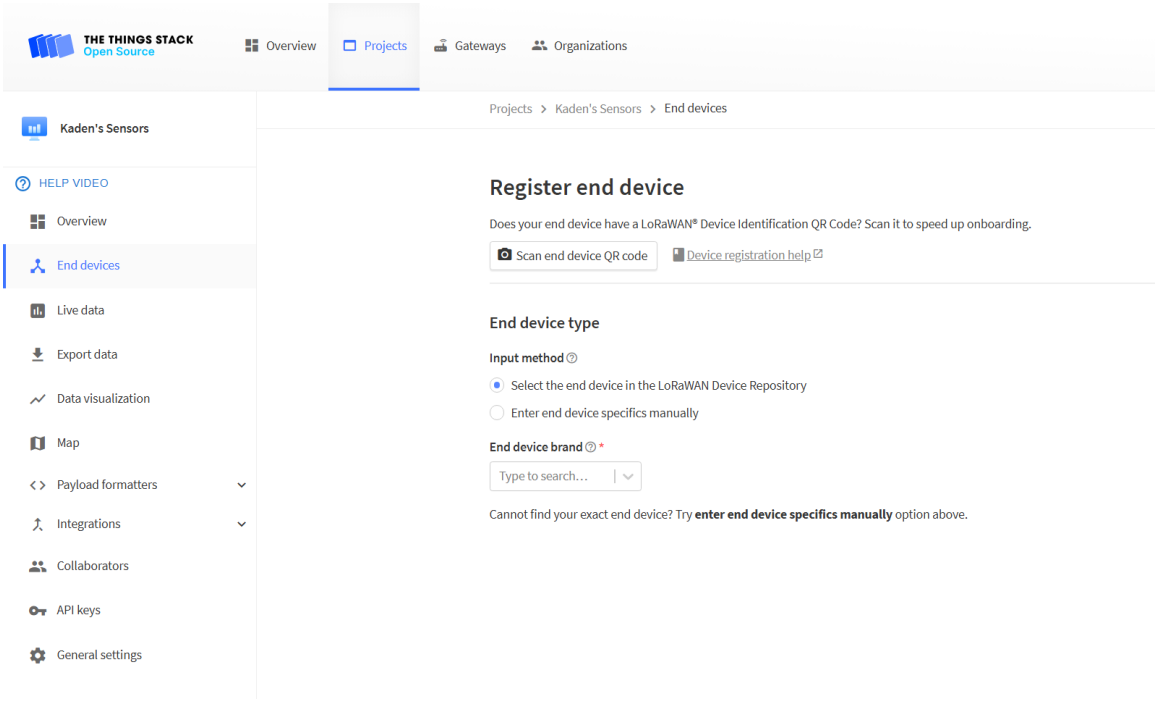

Figure 2.5: Register sensor page. [10].

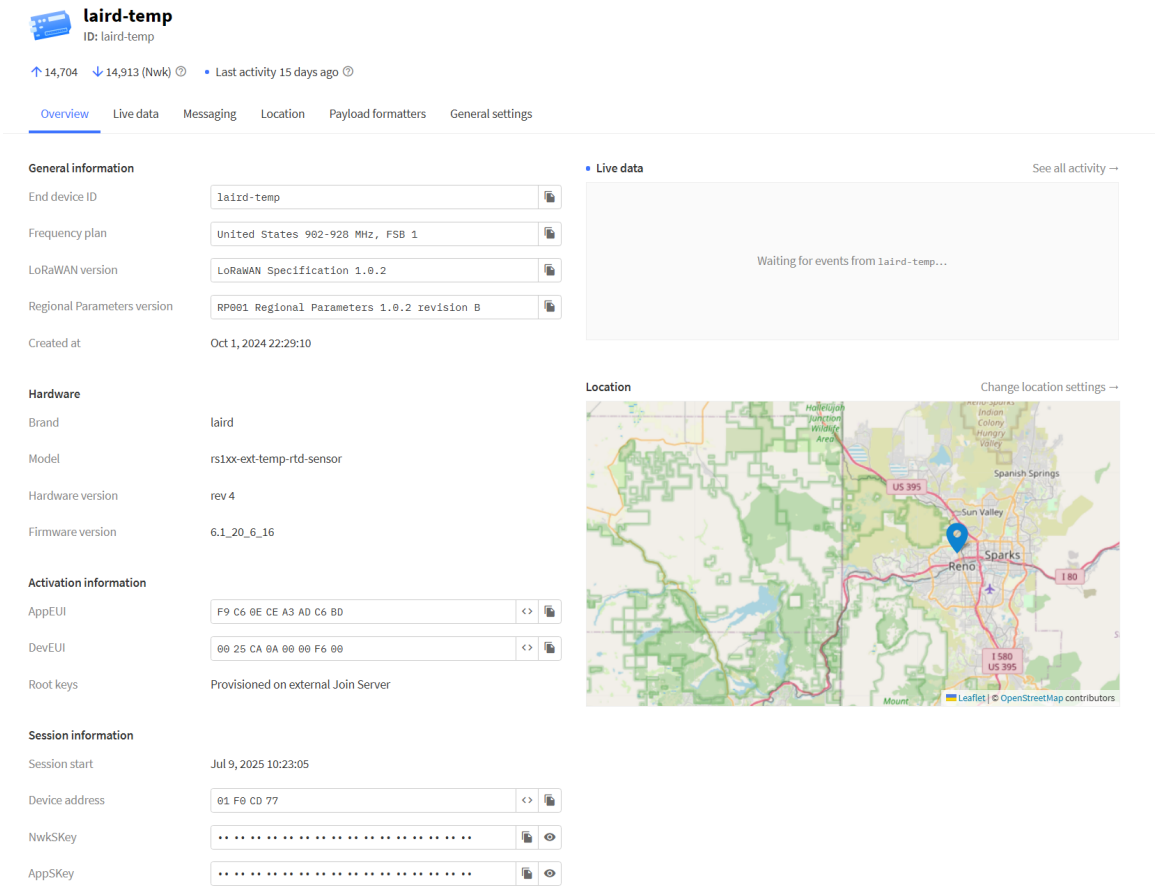

Figure 2.6: Sensor overview page. [10].

Export Data

The "Export Data" tab, as seen in Figure 2.7, allows users to select one or more devices, along with a custom time range, to query historical data within a project. Once a user requests to fetch the data, the resulting data is displayed in a tabular format, accompanied by several export options. Within the table, users can filter and organize the data according to their preferences. Individual columns can be sorted in ascending or descending order, filtered by specific keywords or phrases, and toggled to show or hide based on their current visibility. After customizing the table, users have the option to export the data as a CSV or JSON file, which can then be downloaded and viewed in external platforms such as Excel.

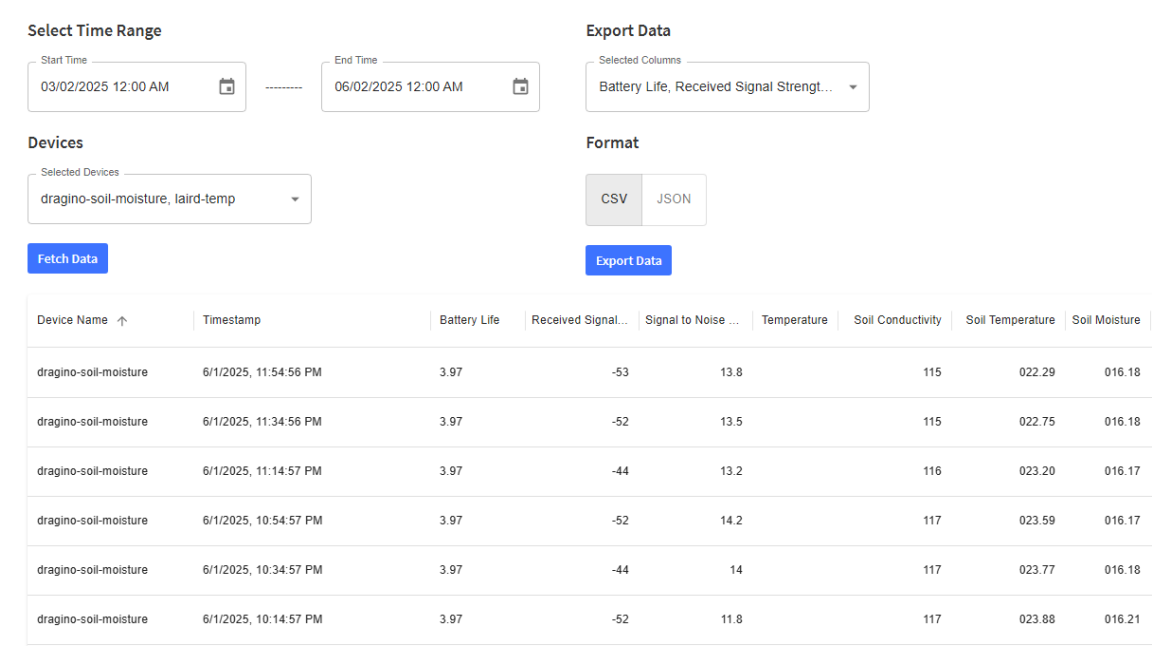

Figure 2.7: Data export page within the project tab on the VISTA portal.

Data Visualization

The "Data Visualization" tab, shown in Figure 2.8, is a prototype version of the data visualizations that have been developed during the research for the thesis, but shows a baseline functionality that has been improved. This page allows users to select device(s) and their reading(s) alongside either a custom time range equivalent to the one found on the export data page or a pre-defined time range presented via time buttons. This version of the visualization tab only offers one graph type: the line graph. Based on the length of the time range, an aggregate by option is available for the user to select, allowing them to choose whether the data should be aggregated by a specific time frame, such as per hour or day. The line graph facilitates the simultaneous visualization of multiple readings, independent of the device, with each reading represented by a unique color.

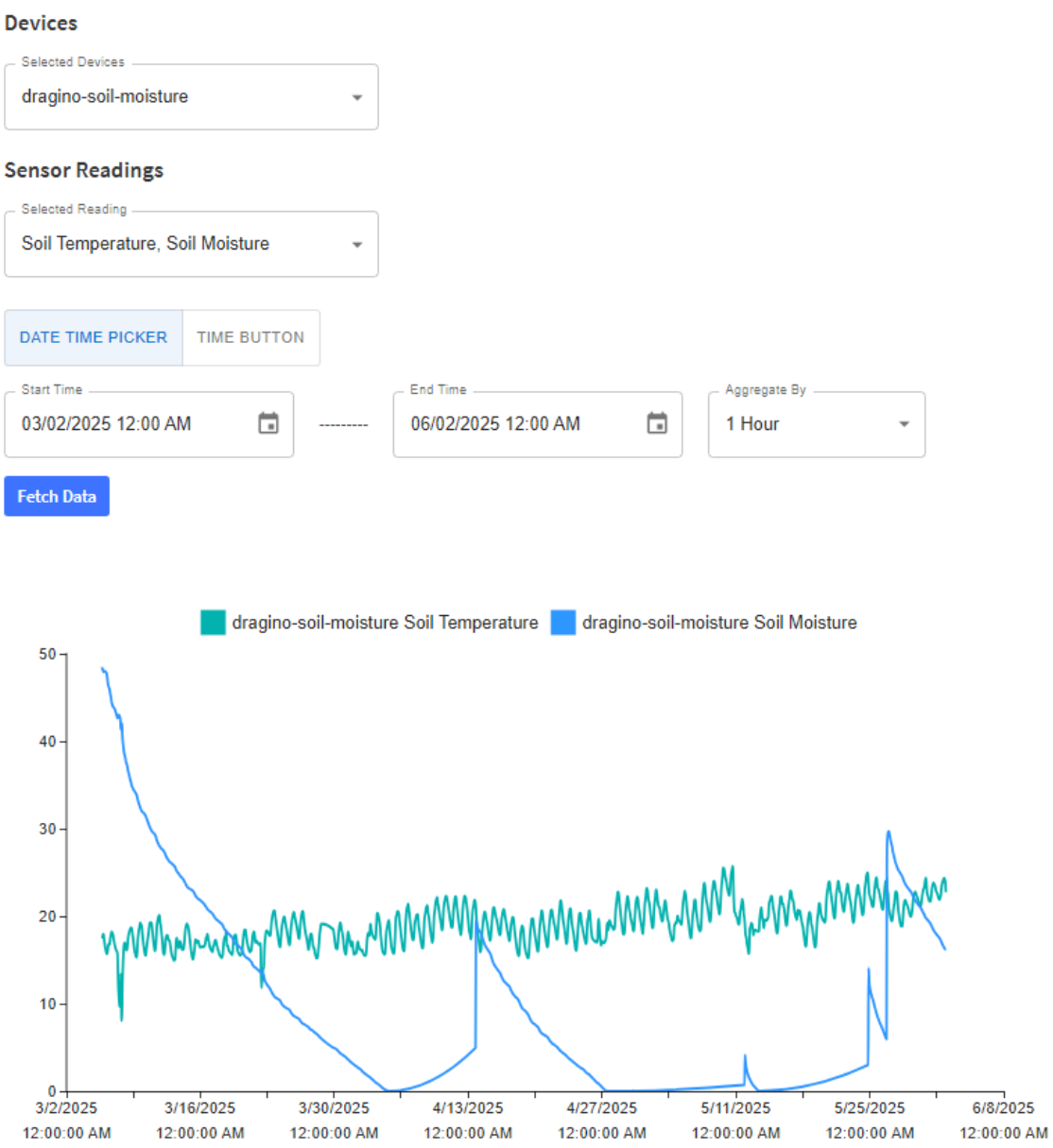

Figure 2.8: Data Visualization page within the project tab on the VISTA portal.

Location/Map

Within the "End Devices" tab, users are able to select each sensor that has been added to the project. Once selected, users can then set the end device location manually, as shown in Figure 2.9. This is a pre-existing functionality for The Things Stack and has not been added by the developers of the VISTA portal. This version

of the location functionality includes a map where users can click to set a location, or enter the latitude, longitude, and altitude manually of the sensor location. Once saved, a marker pinpointing the sensor location will remain on the map. However, an improved version of this functionality has been separated into a separate project tab that will be discussed in Chapter 5.

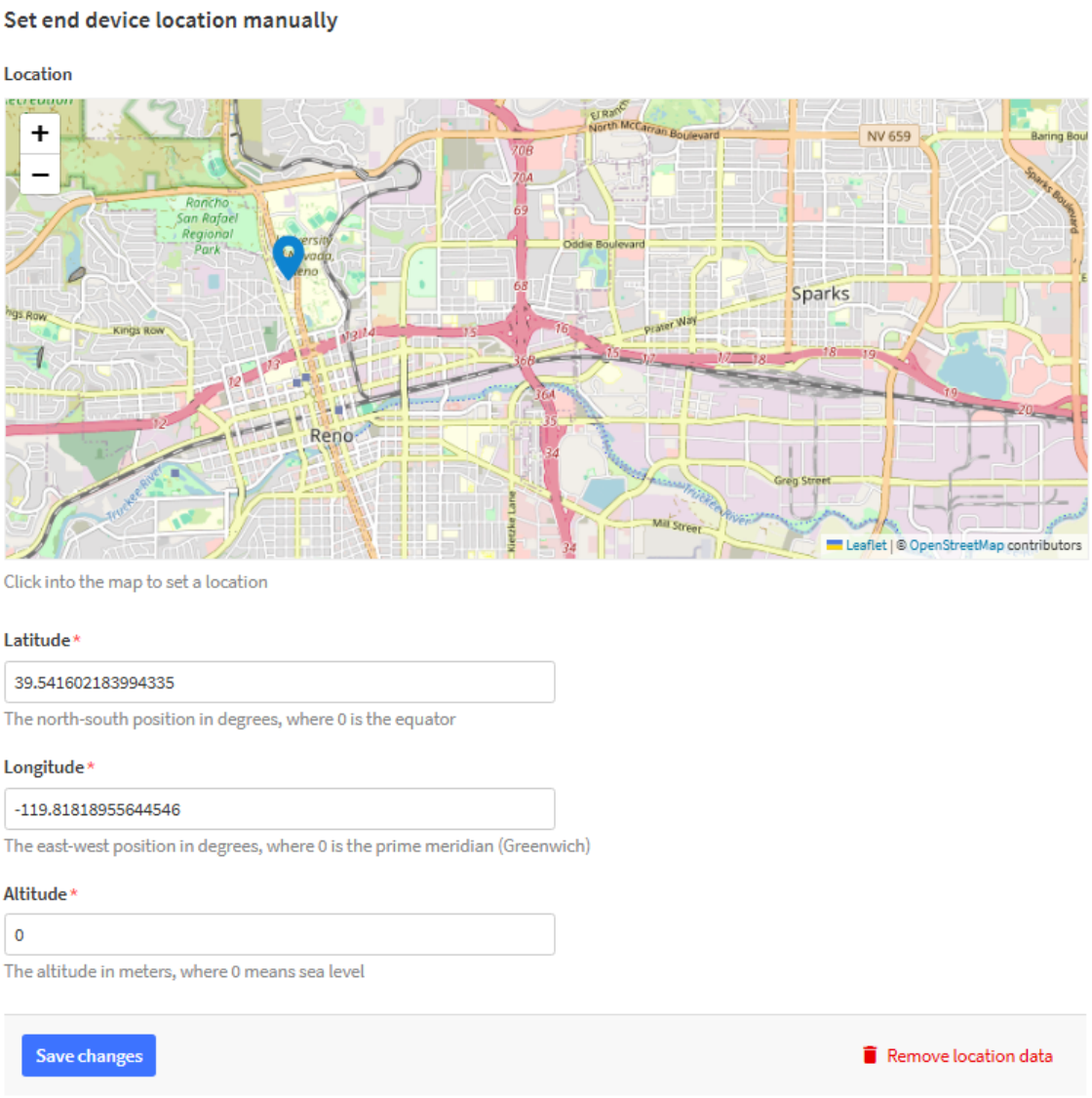

Figure 2.9: Map of sensor location in the project tab on the VISTA portal.

2.3.2 Gateways

The "Gateways" tab in the VISTA portal allows users to register and monitor LoRaWAN gateways that serve as communication middlemen between end devices and the network server. Gateways receive uplinks from sensors and forward them to The Things Stack, while also transmitting downlink messages back to the sensors when required. Each gateway is typically associated with a physical location and operates within a specific frequency plan. Since this functionality mirrors that of The Things Stack, the VISTA portal does not introduce any modifications to how gateways are handled. Users can view key metadata such as status, connection history, frequency settings, and location coordinates. Maintaining active and reliable gateways is essential for ensuring consistent sensor communication within the LoRaWAN network.

2.3.3 Organizations

The Organizations tab is used to group and manage user access and ownership across different projects and devices. This structure is inherited directly from The Things Stack, where organizations act as administrative entities that can contain users or projects as their members. Organizations help facilitate collaborative sensor deployments by allowing multiple users to manage and monitor shared resources. Roles and permissions can be configured per organization, ensuring that appropriate levels of access are maintained for each member, based on their role in the organization. Being a part of an organization is not a required step towards adding end devices to projects, but it is suggested to create/join one to maintain consistency within the platform.

2.4 VISTA Portal Technologies

The VISTA portal integrates several modern technologies to enable real-time data processing, visualization, and user interaction with IoT sensor data. These technologies span across the entire system stack, including messaging protocols, data

pipelines, databases, back-end services, and front-end frameworks. This section provides an overview of the major components and how they interact within the VISTA architecture. Figure 2.10 illustrates the complete full-stack architecture of the portal before any thesis development. Due to architectural changes, the architecture diagram of the VISTA portal with the included thesis developments will be different from the one displayed in this section.

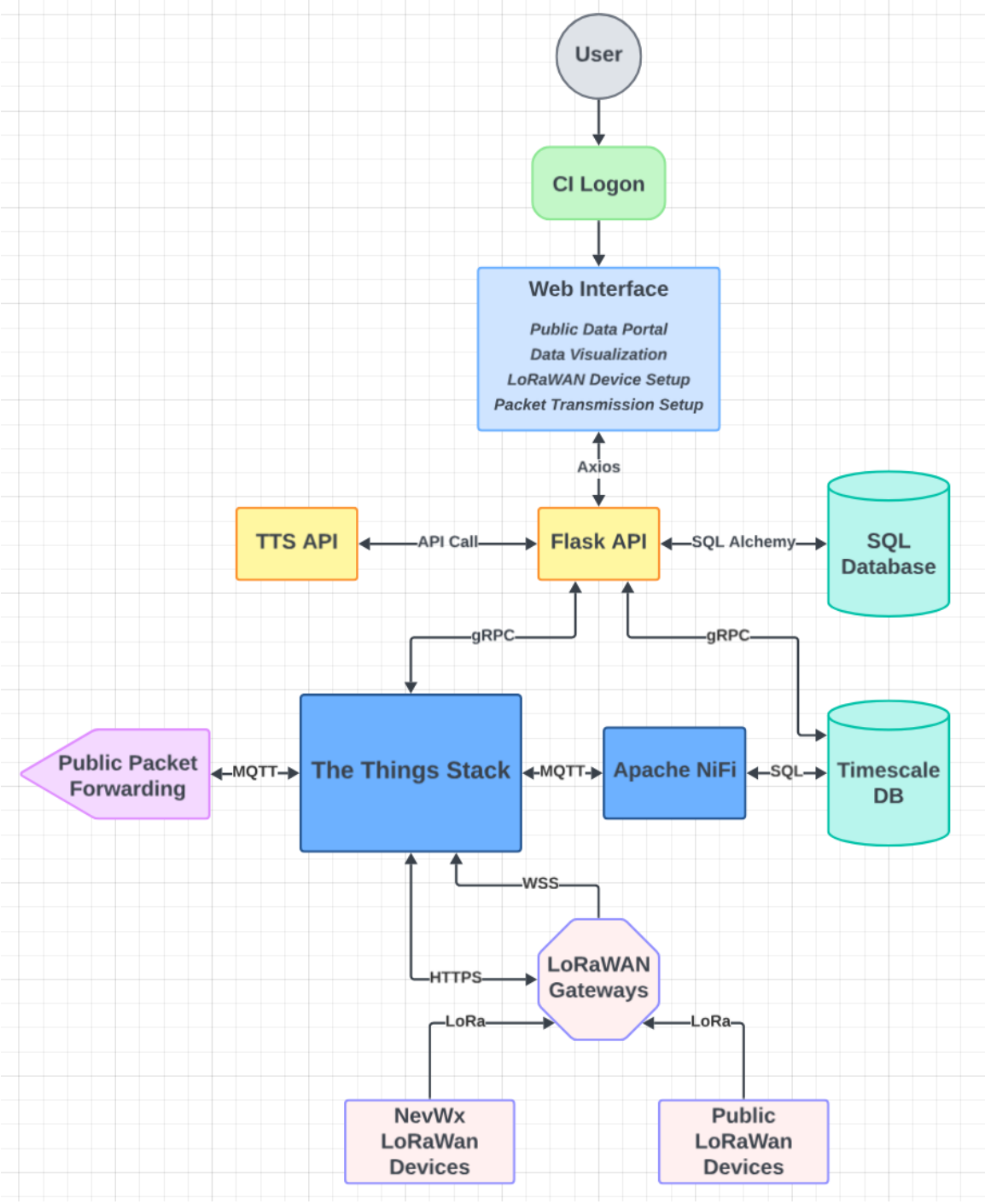

Figure 2.10: The full-stack architecture of the VISTA portal before thesis development.

2.4.1 Data Ingestion

The ingestion process begins when a gateway and sensor(s) are turned on and connected to the LoRaWAN network server. In this instance, the network server is the one operating on the VISTA portal. The data that is sent from the sensors varies depending on the sensor type, varying from fields such as soil temperature, air temperature, or humidity. Each sensor brand and type has its own prebuilt automatic transmission protocol, which transmits data on schedule to the nearest LoRaWAN gateway. Depending on the gateway brand and type, the communication range may vary, and the speed of the messages sent from the sensor could be affected. If the sensor is within range, the LoRaWAN network server receives the sensor message via the User Datagram Protocol (UDP), which is a connectionless protocol. This protocol ensures reliable speed and efficiency of data transmission. Once the data from the sensor has entered the LoRaWAN network server, it is automatically decoded by the built-in decoders provided by the sensor.

With the now decoded message, the message gets sent to an MQTT broker, which is a server that acts as an intermediary for sending the message to its next destination. The Things Stack has an already integrated MQTT broker that can send that message to the next layer of the pipeline, but a centralized version of the MQTT server has been developed for the VISTA portal for greater efficiency and control. The next layer of the ingestion pipeline handles data transformations and database routing. Apache NiFi [23] is used to create an ETL-esque (Extract, Transform, Load) pipeline. NiFi processes incoming payloads from MQTT, formats them, and then stores them into a time-series database, which is discussed in more detail below.

2.4.2 Database: TimescaleDB

All processed sensor data is stored in a TimescaleDB database. TimescaleDB is a time-series extension for PostgreSQL, designed for handling large volumes of timestamped data efficiently [25]. It supports automatic partitioning, real-time aggregations, and time-based queries, making it ideal for storing sensor readings that arrive

at regular intervals. Within VISTA, TimescaleDB is structured to store raw data, which is then queried by operations done in Flask, making it possible to aggregate the data for the export and visualization pages. Aggregation queries can be performed over hourly, daily, or weekly intervals, depending on the use case.

2.4.3 Backend Services: Flask API

A Flask-based backend serves as the middleware layer for the VISTA portal, which enables dynamic querying of the TimescaleDB database [24]. It exposes multiple endpoints that retrieve data from TimescaleDB and serve it to the front end in a structured format. These endpoints power various user-facing features, including the export of historical sensor readings, dynamic graph generation, and aggregation controls. The Flask API handles incoming queries from the user interface, performs validation, and executes optimized SQL commands against TimescaleDB. It also facilitates time-based filtering and ensures that users can request and receive only the data relevant to their project or sensor.

2.4.4 Frontend Framework: React and Material UI

The VISTA frontend is built using React, a JavaScript framework originally introduced in The Things Stack Console [18]. The portal extends the open-source frontend of The Things Stack to add new user-facing components for data visualization, mapping, and export tools. To maintain a modern and consistent user interface, Material UI is used as the component library. Material UI provides pre-built design elements that align with Google's Material Design principles, making it easier to create a responsive and visually clean dashboard experience. To further build on The Things Stack's main design, the components have been selected and altered to fit the theme of the overall portal.

An example of a large component provided by Material UI is the Data Grid that appears on the "Data Export" tab. As mentioned previously, this data table allows users to filter and sort their data individually by columns, which are prebuilt features

that Data Grid provides. This functionality allowed the developers to quickly include analysis options for the data table without the need to spend extra time developing them. Also mentioned previously, the prototype version of the data visualization in the projects tab was also constructed using a line graph component provided by Material UI. Due to the easy integration with other components, at the time, it was decided that this visualization would be adequate. However, it would be quickly discovered that this visualization component was ineffective due to its visually slow rendering performance. In scenarios where multiple sensor readings were selected, users experienced issues with hovering over the graph to view its values, which will later be analyzed in-depth in Chapter 4, as a result of a user study.

2.4.5 Graphing Library: Plotly.js

For the thesis development, the portal's sensor reading graphing library has been switched to Plotly.js, which is a powerful JavaScript graphing library capable of rendering interactive and high-performance charts. Plotly supports a wide range of chart types, including line charts, scatter plots, and bar graphs, with features such as zooming, tooltips, and custom styling [16]. Plotly.js is integrated directly into the React components used in the "Data Visualization" tab of the portal. It allows for real-time graph updates, dynamic axis scaling, and multi-series support—all essential features for interpreting environmental sensor trends over time. Unlike the chart implemented using Material UI, Plotly.js efficiently handles large data sets with multiple readings on the same graph. This graphing library not only suits the overall theme of the web portal, but also provides the tools to be able to analyze IoT sensor data without any issues. An example line graph representation provided by Plotly.js' documentation is shown below in Figure 2.11. This example highlights one of the major strengths of Plotly.js, which is the ability to include multiple lines on a line graph with visual clarity and effectiveness.

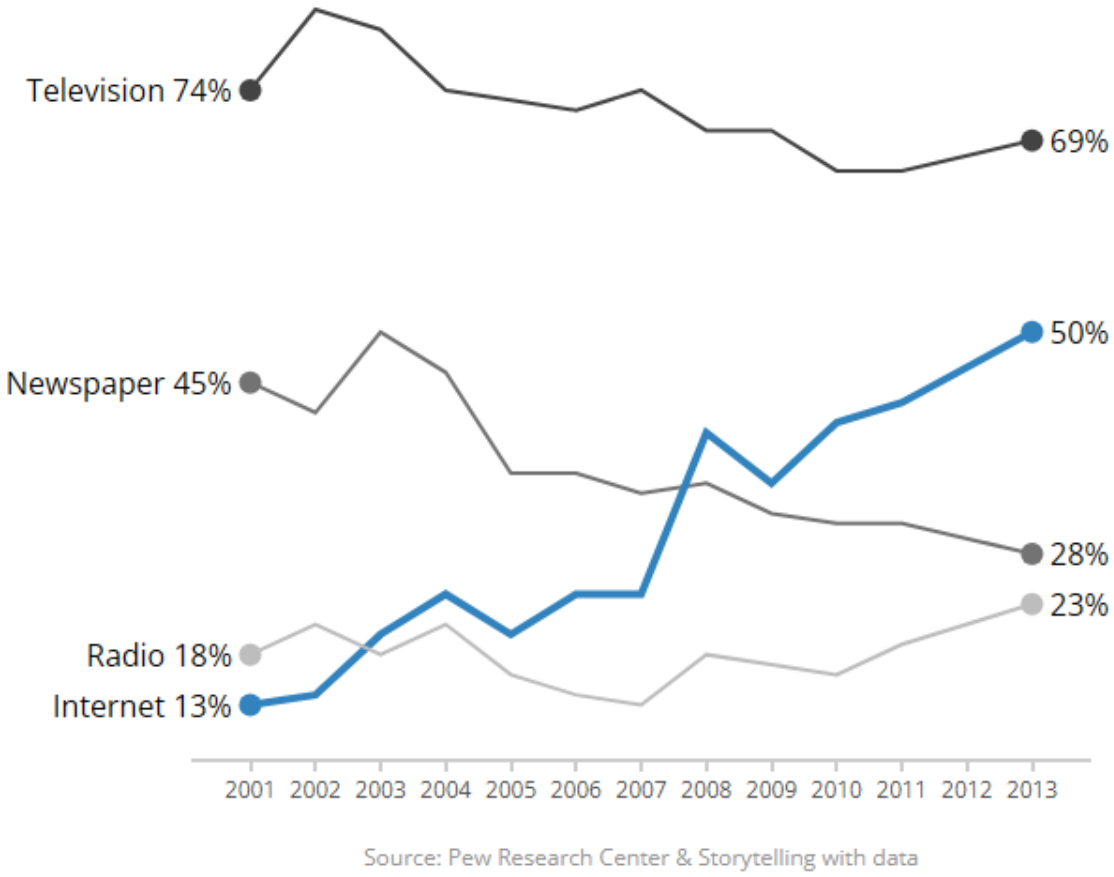

Figure 2.11: An example of a line graph created using Plotly.js [16].

Chapter 3

Related Work

This chapter provides an overview of similar data visualization platforms for IoT devices. More specifically, section 3.1 covers platforms that have been created to monitor and gather data related to environmental science, similar to that of the VISTA web portal. Section 3.2 explores the broader application of these visualization platforms, where their use can be attributed to personal or professional work unrelated to environmental science. Sections 3.3 and 3.4 provide a summary of tools and technologies that can be used to develop an effective visualization platform.

3.1 Environmental Visualization Platforms

The monitoring of environmental data using IoT devices is a common approach for gathering real-time streaming data to understand changes, impacts, and patterns in the environment. Developing a robust and scalable platform that can efficiently and accurately display data through a data visualization is one of the key components for achieving this. This section includes a few different modern platform implementations with a similar goal in mind: displaying environmental data in a data visualization platform accurately and efficiently.

3.1.1 Environmental Monitoring in Jitao

An example of this is Zhang et al. [27], who have developed a visualization platform for monitoring environmental factors, such as temperature, light stability, and humidity,

that affect crop growth. The authors mention that their primary goal in developing this network is to monitor the environment in which their crop grows and detect any issues that arise in real time. Their platform architecture differs from the VISTA web portal as they do not use LoRa devices in their protocol, but rather use a different set of IoT sensors and gateways to transmit the data to a cloud-based database and client. To achieve this functionality, the authors mention that a wireless data transmission module is used to send sensor data to a cloud database by using Low Power Wide Area Networks (LPWAN). This technology stack differs greatly from that of the VISTA web portal; however, a similar goal is achieved as streaming data is directly stored in a database via event-driven protocols.

The main objective of the authors' platform can be seen on the client-side software application, being the hub to display the dynamic display of charts in various different modules. In comparison with the data visualization page in the VISTA portal, their approach features a module that allows the user to visualize real-time and historical data. This differs from the VISTA approach, as only historical sensor data can be gathered from either the clustered column or line graphs. The authors claim that this approach allows users to develop a greater understanding of potential environmental patterns and have a scientific basis for environmental management. Although the VISTA portal only relies on visualization of historical data, being able to compare the most recent readings can nearly simulate the effectiveness of the visualizations created by the authors.

3.1.2 Environmental Sensing in Digital Twin Lakes

Chen et al. [7] also showcase an environmental monitoring visualization platform, but with an environmental sensing approach. Unlike the VISTA portal and the platform provided by Zhang et al., this visualization approach utilizes multi-dimensional and multi-view displays of environmental sensing data in a map-based visualization for lakes. For this platform, the authors specify that their main goal is to primarily utilize geospatial data for a deeper and more complex environmental analysis. This

difference in goals is the main distinction from the geospatial data visualization in the VISTA portal, which aims to assist the user in understanding their sensor/gateway deployment locations, rather than for analysis purposes. Although IoT sensors are a focal point of receiving data, the use of surveillance video data primarily aids in developing a proper video rendering of the geospatial visualization.

The authors' proposed client-side dashboard mainly consists of a map view of the location that is being monitored, in this case being China's Poyang Lake. The map view of the location contains data displayed in a heatmap manner, with colors indicating low or high values. Accompanying the map are supplemental visualizations such as line graphs, bar charts, and other similar visualizations that showcase values and trends for specific locations on the map. An advantage of having such a dynamic visualization platform is being able to visually pinpoint environmental data, as well as see it in real time. The scope of the authors' proposed project is much different from that of the VISTA portal, with the placed emphasis being on monitoring changes and patterns for a large body of water, compared to climate research and citizen science.

3.2 Miscellaneous Visualization Platforms

This section dives into data visualization platforms that have been designed to monitor IoT sensor data, but without a direct focus on environmental sciences. It is valuable to understand how other visualization platforms handle and display their data, especially when the platforms are created with the purpose of being used by others, and not only for research-intensive purposes. The platforms that have been selected for this section are also current, which provide relevant and contemporary insights into their design process and integration.

3.2.1 Building Management System Dashboard

According to a *Grand View Research* report, the market size of global smart buildings is projected to grow at a Compound Annual Growth Rate (CAGR) of 28.5% from 2024 to 2030 [20]. In light of this rapid expansion, researchers have focused on de-

veloping data visualization platforms for monitoring and optimizing smart buildings and homes. A notable example is the work by Svalina et al. [22], who developed a dashboard enabling users to interact with sensor-based visualizations to regulate and control energy consumption in buildings. Their study emphasizes the importance of applying preattentive attributes, such as color, shape, and size, and selecting appropriate chart types based on the nature of the data. By providing users with multiple types of visualizations for similar data, the researchers conducted a user study to evaluate dashboard effectiveness. The results indicated a strong user preference for having multiple visual options to interpret the same dataset. This trend mirrors findings from the VISTA portal's "Evaluating Data Presentation Tools for Effective Communication and Usability" user study, where participants similarly preferred flexible data representation. These insights collectively reinforce the importance of adaptive, user-centered design in IoT dashboards.

3.2.2 Cesena Smart Campus

Another example of smart monitoring is presented by Ceccarini et al. [4], who have developed a full-stack application to monitor and visualize smart data across the Cesena University campus. Similar to the Digital Twin Lakes project, their system incorporates camera feeds, though for a different purpose, being to count the number of people in specific areas, such as classrooms. In addition to video data, IoT sensors were deployed to track atmospheric conditions, including temperature, humidity, air pressure, and the presence of air contaminants such as formaldehyde. This approach mirrors the VISTA portal's focus on environmental monitoring, highlighting a shared use of atmospheric sensing technologies. The authors place strong emphasis on the development process of the client-side visualization platform, which uses dynamic charts built with Google Charts, Chart.js, and D3.js. As with other modern dashboards, a focus on chart diversity was central to their design. Offering multiple visualization types enabled users to explore and compare datasets more effectively, which was an approach also echoed in the VISTA portal's initial design philosophy.

3.3 Tools & Technologies for Visualizations

In addition to fully integrated dashboards, a range of standalone tools and platforms exist that support flexible and scalable data visualization. These technologies are often employed as third-party solutions, enabling users to quickly monitor, analyze, and visualize large datasets without the need for custom front-end development. Unlike client-side systems like the VISTA portal, the tools discussed in this section are not embedded within a single application but are instead designed for general-purpose use across a wide variety of domains. While they are not directly applicable to the architecture of the VISTA portal, they represent widely adopted approaches to data visualization that inform best practices and broader trends in the field.

3.3.1 Grafana

Grafana [9] is an open-source data visualization and monitoring platform that supports querying, analyzing, alerting on, and visualizing metrics from large datasets. One of Grafana's primary advantages is its flexibility, as it can connect to a wide range of data sources and supports real-time streaming, making it particularly well-suited for applications involving IoT sensors. Its open-source nature allows developers to customize and extend its capabilities freely. The following features highlight what makes Grafana one of the most widely used and powerful real-time visualization tools:

- **Customizable Dashboards**: Grafana supports flexible dashboard layouts with user-defined themes, branding, panel configurations, and interactive features such as navigation controls, permission settings, and alerts.
- **Advanced Metric Analysis**: Enables ad-hoc querying and comparison of time series data across multiple datasets, facilitating in-depth analysis in parallel.
- **Real-Time Alerts**: Automatically sends alert notifications to users via services like Slack, based on predefined metric thresholds or conditions.

- **Graph Annotations**: Users can tag charts with contextual notes to highlight important events, anomalies, or thresholds.

3.3.2 Chronograf

Chronograf [8] is another example of an open-source data visualization and monitoring platform, specifically designed to visualize real-time data using time-series charts. Unlike Grafana, Chronograf is built to work exclusively with data stored in InfluxDB, a time-series database that belongs to the same ecosystem.

One of the main advantages of Chronograf is its speed and efficiency when paired with InfluxDB, making it a strong choice for visualizing real-time IoT sensor data when InfluxDB is already the database of choice. Chronograf natively supports only time-series charts, which makes creating real-time visualizations simple and intuitive. However, if an ecosystem does not rely on InfluxDB as its data source, alternative tools like Grafana may be more appropriate.

3.3.3 Tableau

Tableau [13] is a widely used commercial data visualization and business intelligence (BI) platform designed to help users explore and analyze data through interactive dashboards and reports. One of Tableau's key strengths lies in its ability to connect to a broad range of data sources, including relational databases, spreadsheets, and cloud services. The platform offers a user-friendly, drag-and-drop interface, making it accessible to users with limited technical expertise. A wide variety of built-in chart types are supported, ranging from standard column and line charts to scatter plots, treemaps, and geographic maps. Tableau also provides abundant support for geospatial data, allowing users to create choropleth/heat maps based on latitude and longitude fields or shapefiles. Tableau dashboards are highly interactive, offering support for filters, tooltips, and parameter controls that enable users to explore the data dynamically. Completed dashboards can be published to Tableau Server or Tableau Public for web-based access and sharing. While Tableau is not open-source

and is less commonly used for real-time IoT monitoring, its extensive visualization capabilities and accessibility make it a popular choice for business and academic applications.

3.4 Graphing Libraries

Selecting the proper graphing library for a web portal's front-end is a vital step towards developing an effective platform that meets the needs of the users. While some visualizations benefit from being basic, others, such as the VISTA web portal visualization, benefit from having as many analysis tools embedded within the visualization as possible. Graphing libraries vary vastly in how they are structured to process and visually render data sets, with each method having its advantages and downsides.

3.4.1 Chart.js

Chart.js [5] is a popular open-source JavaScript library used for creating responsive and customizable data visualizations using the HTML5 canvas element. Since it is a general-purpose JavaScript library rather than one tied to a specific framework, it can be integrated into any JavaScript-based application, including those built with frameworks such as React. Chart.js provides support for a variety of chart types, including bar, line, radar, and more. It is widely adopted due to its simplicity and intuitive API, which allows developers to implement effective data visualizations with minimal configuration. One of the key features of Chart.js is its ability to automatically adapt charts to different screen sizes, eliminating the need for manual responsiveness handling. Built-in animations and transitions enhance the visual appeal and responsiveness, making it commercial-ready without the need for custom adjustments. Chart.js is particularly well-suited for applications requiring straightforward visualizations of moderately sized datasets. Its use of the canvas API makes it a viable selection for real-time IoT sensor data, although it generally offers fewer customization options compared to SVG-based libraries, such as the one that will be introduced next.

3.4.2 Recharts

Recharts [19] is a composable charting library built specifically for React applications. It utilizes SVG to render chart elements and is built on top of D3.js submodules, allowing it to inherit D3's powerful data-driven visual capabilities while presenting a simplified and declarative interface for React developers. The core philosophy of Recharts is to align with React's component-based architecture. Developers can build visualizations using JSX by combining pre-built components for different chart types. This modular approach allows for a high degree of customization while still maintaining code readability and structure in a React environment. Similar to the previously mentioned Chart.js, Recharts contains a wide variety of different chart types ready to implement directly via imports. While Recharts offers strong integration with React and is easy to use, it can face performance limitations when handling large datasets due to the overhead of SVG rendering. This rendering type makes this library inefficient to utilize for real-time IoT sensor visualizations.

Chapter 4

Motivation and Project Objectives

This chapter outlines the motivation behind the development work presented in this thesis by identifying the primary goals that emerged from a previous evaluation of the VISTA portal. The intended users of the portal are also discussed. Section 4.1 begins by discussing the original intent of the platform and the role of the prototype system that was evaluated in an early user study. The key findings and qualitative feedback from the study are summarized, revealing critical limitations in the platform's data visualization tool. Based on those insights, the three core objectives that the thesis addresses are presented. Section 4.2 introduced the intended users of the platform with example use case scenarios.

4.1 Motivation & Project Objectives

The VISTA portal was developed to serve as a platform for both scientists and citizens to access, monitor, and analyze their IoT sensor data. To achieve this goal, the portal must offer features that enable users to easily visualize and interpret the data being collected. This thesis specifically focuses on improving the data visualization capabilities of the platform by implementing more advanced and user-friendly tools. As mentioned in Chapter 2, a prototype version of the VISTA portal existed before this research. That version was originally designed to support monitoring functionality for scientists involved in ongoing research, as well as contributors to its citizen science components.

4.1.1 Insights from the Previous User Study

A user study was conducted to evaluate the effectiveness of the prototype's "Data Visualization" and "Data Export" modules. The study, titled "Evaluating Data Presentation Tools for Effective Communication, tasked participants with identifying specific data points, determining monthly averages, and locating peak values using both modules. Upon completion, participants shared qualitative feedback regarding their preferences, overall user experience, and suggestions for improvement. Although the quantitative findings were not directly tied to this thesis's objectives, the qualitative feedback revealed notable usability limitations within the visualization tools. These insights became the driving force behind the enhancements proposed in this thesis and helped shape its development goals.

4.1.2 Summary of User Study Feedback

In the post-study questionnaire, participants answered questions regarding their experience with the VISTA portal's visualization tools. They were asked to indicate which module they preferred, which was easier to use, what challenges they encountered, and how the tools could be improved. Before the study was conducted, the researchers hypothesized that the data table would most likely be preferred for our set of tasks, while the data visualization would be preferred more by the participants for real-world scenarios outside of this study. However, it was found that over 80% of the participants preferred the data table in both scenarios, marking a surprising finding that raised some flags about the effectiveness of the prototype visualization. Participants commonly reported difficulty locating specific data points due to noticeable visual lag when interacting with the line graph. Other feedback pointed to a lack of interactive features, such as zooming, scrolling, or panning, that are typically expected in modern visualization tools. In comparison, many users found the export table more helpful for actual data analysis, citing its sorting and filtering capabilities. A frequently suggested improvement was to introduce an additional chart type better suited for analyzing trends, while also enhancing the line graph to allow smoother,

more interactive navigation of time-based data. Although the prototype version of the line graph in the user study allowed all participants to accurately complete the majority of the tasks, it was evident that it wasn't the optimal choice for the participants.

4.1.3 Project Goals

The insights from the previous user study shaped the following primary goals of the thesis:

Improving Visualization Performance

One of the primary goals of this thesis is to improve the overall performance of the VISTA portal's data visualization tools by replacing the existing graphing library. The original implementation utilized an SVG-based rendering approach, which proved inefficient when displaying large amounts of time-series sensor data. The goal of the new graphing implementation is to select a library that uses a different rendering method, one that is designed to handle large data sets, even when multiple sensor readings are selected. Participants in the previous user study reported significant visual lag, especially when hovering over the line graph to inspect individual data points. This issue was worsened when multiple readings from various sensors were displayed simultaneously on a single graph, rendering the interaction nearly unusable. With glaring issues like this, the prototype was unable to handle large data sets that the researchers queried, marking a much-needed change, regardless of the user study results.

Introducing Analytical Visualization Tools

While line graphs are effective at showing trends and patterns in time-series data, they are not always well-suited for in-depth analytical tasks. The second goal of this thesis is to enhance the analytical capabilities of the VISTA portal by integrating a new form of visualization specifically designed for more detailed data analysis. This new visualization would allow users to visually analyze specific subsets of their sensor data,

helping them answer questions such as: What are the highest or lowest readings for a specific sensor over a week? How do two variables correlate over time? Additionally, the goal is to give users the ability to more precisely select which data readings they want to display and analyze, rather than being limited to full-sensor visualizations. This allows for a more targeted and efficient experience when exploring environmental datasets.

Enhancing User Experience and UI Design

The final goal of this thesis is to improve the overall user experience of the VISTA portal, particularly in terms of interface usability and accessibility of visualization tools. Users in the previous study indicated the need for additional functionality and improved layout clarity when interacting with visualizations. This goal includes adding more visualization forms to the platform and embedding interactive tools, such as zooming, panning, or switching time ranges, directly within the charts. Furthermore, the UI layout of the visualization page will be refined to reduce visual clutter, increase readability, and help users more easily locate key controls and options. Collectively, these improvements aim to create a more intuitive and supportive environment for both scientists and citizen science users monitoring environmental data.

4.2 Intended Users

The VISTA web portal's intended users are citizen scientists and researchers who plan on using IoT sensors for collecting data, such as climate researchers. At this stage, researchers from the University of Nevada, Reno are the primary users, but in the future, the intended users are researchers from other institutions as well. To be able to register and log in to the portal, users must have access to either their institution login or through a third-party platform such as GitHub or Google. Although this functionality allows anyone with an account through the third-party platforms to register and utilize the portal, at this stage, they must either be a part of the research development or a known citizen scientists of the project. To be able to utilize the

portal actively, the user must be in contact with the administrators to be able to properly install gateways and IoT sensors, as well as have an understanding of the LoRaWAN protocol.

4.2.1 Citizen Scientists

Citizen scientists are individuals or community members who contribute to scientific research outside of traditional academic or professional institutions [26]. They often participate in environmental monitoring projects to collect, share, and analyze data relevant to their local surroundings. The VISTA portal supports these users by offering an accessible platform where they can register sensors, monitor real-time data, and export readings for personal or community-based analysis. This group is an intended user base due to their growing role in climate-related initiatives and community-based research efforts. By lowering the barrier to participation, VISTA encourages grassroots involvement in environmental science. This results in citizens who have no technical or scientific background being able to collect and monitor environmental data effectively.

Example use case: A community gardening group manages a public garden space that showcases a wide variety of native and seasonal plants. The garden allows visitors to use this as an educational and personal experience to learn about these plants and admire them simultaneously. However, maintaining optimal growing conditions throughout the year has posed recurring challenges, especially as climate patterns become less predictable. Due to these conditions potentially affecting the plant's health and visual appeal, IoT sensors connected through the VISTA portal send important garden soil and air data. Using the portal's data visualization tools, the garden group can monitor and track the environmental data of their garden. For example, they can observe declining soil moisture trends during dry periods, prompting more targeted irrigation schedules that conserve water while sustaining plant health. Temperature data helps inform decisions about when to cover certain plants during colder periods to prevent freeze damage. With this data, the community gardening group can

also use the visualization tool for analysis, allowing them to make predictions by preparing from historical data. An example of this is understanding the usual temperature variations during the winter months from previous years and preparing for those conditions early in the next year.

4.2.2 Institutional Researchers

Institutional researchers include faculty members, graduate students, and lab groups affiliated with academic or government institutions engaged in environmental research. These users typically conduct long-term data collection campaigns to study climate patterns and ecological systems. The VISTA portal provides an interface for these users to manage sensor networks, monitor data in real time, visualize patterns, and export large datasets for scientific analysis. They are a primary target user group due to their technical expertise, existing infrastructure for research, and demand for platforms that support high-volume, reliable data handling. Their feedback is especially valuable for shaping the development of visualization tools and system improvements.

A research group at a university focuses on natural hazard warning systems, particularly in mountainous regions where an unpredictable climate poses increased risks of hazards such as avalanches or floods. Their project aims to develop an early warning system based on environmental indicators from various regions. The researchers deploy a distributed network of IoT sensors capable of tracking variables such as soil moisture, precipitation rate, and snow levels. Each sensor group is integrated with the VISTA portal and separated into projects based on where they are located. Through the portal, researchers can visualize trends in precipitation and temperature fluctuations. If a trend is detected that can indicate a potential avalanche or flood in the region through the visualizations, the researchers can immediately begin to take action. The researchers utilize the analysis visualization to document previous events and log important data that can help in future cases.

Chapter 5

Software Architecture & Implementation

This chapter outlines the full-stack design and implementation efforts introduced in this thesis, covering both the front-end and back-end systems that enable sensor data interaction within the VISTA portal. It begins with Section 5.1, which presents the functional and non-functional requirements defined for the system, detailing the essential features and performance benchmarks that guided development. Section 5.2 then introduces the use case modeling, capturing representative user stories and interactions for both citizen scientists and researchers to ensure the portal meets practical usage needs. Section 5.3 provides an overview of the updated three-tier system architecture, describing the technologies and communication protocols that support the enhanced visualization features. Section 5.5 follows with an overview of the front-end architecture, highlighting the technologies and component structures used to deliver a responsive and modular interface. This includes the Visualization Module and Map Module, which allow users to explore sensor data temporally and spatially through intuitive tools and visual feedback. Section 5.4 then details the design of the Flask-based back-end and explains the endpoints, data querying logic, and data transformation processes that support both visualization and user interaction. Subsections 5.5.1 and 5.5.2 go deeper into the functional implementation of the Visualization and Map Modules, respectively. Subsection 5.5.3 discusses user-centric features such as validation, visual toggling, and interface feedback that were built to enhance usability.

Finally, Section 5.6 describes the most significant software development obstacles encountered during this work, including database migration issues and the complexities of implementing advanced filtering logic.

5.1 Requirements Specification

5.1.1 Functional Requirements

In software systems, functional requirements define the technical capabilities that must be implemented for the system to be considered successful [14]. For the additions proposed in this thesis to the VISTA portal, the functional requirements are outlined in Table 5.1. These requirements are organized into three levels, reflecting both their importance to the thesis goals and the relative complexity of their implementation. Level 1 includes the core functionalities that were essential to achieving the baseline objectives of the research and were relatively straightforward to implement. Level 2 consists of more advanced features that require greater development effort and contribute to the overall enhancement and robustness of the platform. Level 3 outlines desirable features that were not implemented in the current scope of the thesis but represent valuable opportunities for future work, which will be discussed further in Chapter 8.

The Level 1 requirements represent the core functionality necessary to deliver the foundational features of the improved VISTA portal. These features ensure that users can interact with and interpret sensor data in a meaningful way. Requirements such as FR1 and FR2 introduce the ability for users to switch between two distinct visualization types, time series and clustered column charts while ensuring that these charts render appropriately upon selection. FR3 adds temporal control through time selection features, allowing users to either utilize the pre-defined periods or select their customized period, which is essential in providing users with the needed support for reaching their needs in visualization customization. Ensuring that the system displays accurate data upon request (FR4) is a critical baseline for any data-driven

Name	Level	Description
FR1	1	The system will show the buttons to choose between time series and clustered column charts.
FR2	1	The system will include the implementation of each chart type when a user has clicked on the corresponding button.
FR3	1	The system will include a time button and a date-time chart option for selecting periods on each chart type.
FR4	1	The system will display accurate recordings on each data visualization chart type when the user requests to fetch data.
FR5	1	The system will allow the user to hover over their data to more closely examine it in the visualizations.
FR6	1	The system will include an option for the user to select data aggregation for visualizations.
FR7	1	The system will allow users to mark sensor locations within the map module.
FR8	1	The system will allow users to view the most recent reading for marked/unmarked sensors on the map module.
FR9	1	The system will show the marked sensor locations on the map.
FR10	2	The system will include embedded tools within the visualization, such as the option to drag.
FR11	2	The system will include advanced filtering options for the clustered column chart.
FR12	2	The system will allow the user to view more than 10,000 data points on the visualization page.
FR13	3	The system will utilize Cloud Technologies for visualization storage.
FR14	3	The system will allow for full user customization on the data visualization page.
FR15	3	The system will allow for heatmap/choropleth options on the Map visualization.

Table 5.1: Functional Requirements

application. FR5 enhances usability by supporting hover functionality for inspecting specific data points, a key feature identified during the previous user study. FR6 introduces basic data aggregation capabilities, which help summarize data for users without overwhelming them with extra unnecessary information. In the map visualization component, FR7, FR8, and FR9 deliver the minimum viable experience for using the map module, which includes: supporting sensor location marking, showing the most recent readings, and rendering sensor icons. Altogether, these Level 1 requirements form the essential building blocks needed to meet the project's most immediate and impactful goals.

The Level 2 requirements provide enhancements that increase the sophistication, performance, and analytical capabilities of the portal. FR10 introduces embedded chart tools such as dragging, which elevate the interactivity of the visualizations and allow for deeper data exploration. FR11 supports advanced filtering within the clustered column chart, enabling users to customize the data that they are viewing for analysis. One of the most critical performance upgrades is addressed in FR12, which focuses on ensuring that the system can handle over 10,000 data points without introducing noticeable lag. This requirement directly responds to the performance limitations identified during the user study, ensuring that the platform remains usable at scale, which was a major issue with the system before the thesis development.

Finally, Level 3 requirements capture optional or "nice-to-have" features that, while not essential for the immediate scope of this thesis, represent possible directions for future development. These requirements were not included in the thesis research due to things such as time constraints, the need for unnecessary funding, and the low level of importance. FR13 proposes the integration of cloud technologies for storing and retrieving visualization data, improving scalability and collaboration across distributed users. FR14 envisions a more customizable user experience by allowing individuals to personalize aspects of the visualization page, such as layout, chart defaults, or color schemes. Lastly, FR15 aims to enhance the geospatial visualization tool set by adding support for heatmaps and choropleths, which would allow

users to observe trends or concentrations of sensor readings across geographic areas. These features, while not implemented during this research phase, represent points of interest that could be developed further down the line.

5.1.2 Non-Functional Requirements

Non-functional requirements (NFRs) define the quality attributes and operational constraints of a system rather than specific behaviors or features [14]. Unlike functional requirements, which describe what the system should do (such as rendering specific chart types or handling user interactions), non-functional requirements describe how the system should perform under certain conditions. As shown in Table 5.2, these requirements focus on performance, scalability, responsiveness, and system architecture. For example, requirements such as low-latency rendering (NFR1) and stable component rendering (NFR4) ensure that users have a smooth and efficient experience when working with large datasets. Others, such as modularity (NFR5) and technology stack consistency (NFR3), help ensure the system remains maintainable and extensible. Collectively, these non-functional requirements are critical for ensuring that the VISTA portal is not only feature-complete but also robust, responsive, and scalable for long-term use.

5.2 Use Case Modeling

Use case modeling plays a critical role in identifying how different types of users will interact with the VISTA portal. Because the platform is designed to serve both institutional researchers and citizen scientists as intended users, it is essential to clearly define their key goals, the features they rely on, and how they interact with the visualization and monitoring tools. These use cases help ensure the platform supports real-world needs and can guide both the system's functional development and user experience design.

Name	Description
NFR1	The system should render data visualizations with less than 100ms delay when hovering over 10,000+ data points.
NFR2	The system should support responsive design for seamless usability across desktops, tablets, and mobile browsers.
NFR3	The system must utilize Flask as its backend framework and React as its front-end framework, ensuring consistency with the chosen development stack and maximizing compatibility with modular components.
NFR4	The visualization components should load and become interactive within 2 seconds of page access under normal network conditions.
NFR5	The system should be modular, allowing new chart types or tools to be integrated with minimal refactoring.

Table 5.2: Non-Functional Requirements

5.2.1 User Stories

To better understand user needs and expectations, this section outlines example user stories for the two primary user groups: citizen scientists and institutional researchers. These stories represent the high-level goals and tasks that users hope to accomplish through the portal. First, the user stories for citizen scientists using the VISTA portal will be listed below. The user stories for citizen scientists are catered more toward the user experience of using the visualization pages.

1. As a citizen scientist, I want to be able to easily select the date and time that I want to visualize my data in. I want to be able to select a custom time frame.

2. As a citizen scientist, I want to be able to intuitively interpret the data visualizations without the need for external explanations.

3. As a citizen scientist, I want the ability to see where I placed my sensor on a map so that I can keep track of it without forgetting.

4. As a citizen scientist, I want to have a visualization page with limited complexity so I can quickly monitor my sensors.

Although the user stories between the researchers and citizen scientists are very similar, the ones for the researchers are more focused on functionality. Unlike citizen scientists whose user stories revolve around user experience, the researchers' user stories are primarily designed for best performance and analysis assistance. The user stories for institutional researchers using the VISTA portal will be listed below.

1. As a researcher, I want to be able to have an advanced form of filtering and aggregating data for each data visualization.

2. As a researcher, I want to be able to analyze large time-series datasets from multiple sensors, so that I can identify trends for scientific study.

3. As a researcher, I want quick access to the visualizations after I request to fetch them, regardless of the size of the dataset.

4. As a researcher, I want to map sensor locations geographically and visualize real-time readings so that I can make an analysis based on location.

5. As a researcher, I want to have the option to analyze data with multiple chart types.

6. As a researcher, I want tools within the visualization to help me view and interpret it with increased clarity.

Following the user stories outlined in the previous section, Table 5.3 captures a selection of the primary use cases derived from the functional needs of the VISTA portal's intended users. These use cases help formalize how both citizen scientists and institutional researchers are expected to interact with the platform. Each use case reflects real-world scenarios users may encounter, including visualization access, sensor management, data retrieval, and custom filtering. By mapping these interactions in a structured way, we clarify how the platform supports its users' goals and validate that the functional requirements have been implemented to serve these workflows.

5.3 Software Architecture & Implementation

The VISTA portal, enhanced with new visualization features, operates on a three-tiered architecture that enables seamless interaction between its core components. At the front end, the interface is built using React, based on the open-source code

Name	**Title**	**Description**
UC01	AccessVisualization	The user logs in and navigates to the visualization module, where they are presented with options for selecting between a time series or clustered column chart, initiating their data exploration.
UC02	ViewSensorLocations	The user navigates to the map module and selects the option to mark or unmark sensor locations, enabling them to visualize active devices in their environment.
UC03	HoverDataInspection	The user hovers over points on the chart to view individual data values and observe specific sensor readings at exact timestamps.
UC04	ChartVariation	The user navigates to the data visualization module and has the option to select either the clustered column chart or the time-series line graph as their chart type.
UC05	TimeSelection	The user accesses the date/time tool and adjusts the visible window of the chart to display readings within a custom time frame for detailed temporal analysis.
UC06	ApplyAggregation	The user selects from various aggregation options (e.g., hour, day, week) to condense large datasets into meaningful trend lines in the time series chart.
UC07	ApplyAdvancedFilters	The user interacts with the filter panel on the clustered column chart to limit the dataset based on the time of day and day of the week.
UC08	ResponsiveExperience	The user accesses the VISTA portal on multiple device types and experiences consistent usability and layout responsiveness across screen sizes.

Table 5.3: Primary Use Cases Derived from User Stories

base provided by The Things Stack, as discussed in Chapter 2. This user-facing layer communicates with the backend through RESTful HTTP requests handled by the Flask framework. Depending on the nature of the request, Flask processes the parameters and retrieves the appropriate sensor data from a TimescaleDB database, which stores data streamed from field-deployed sensors via their respective gateways. This section explores the architecture and design of each layer of the system and explains how their integration enables the VISTA portal to receive user input and return an effective visualization.

5.3.1 System Overview

To help conceptualize the architectural structure of the VISTA portal, an updated system architecture diagram is provided in Figure 5.1, building upon the version initially presented in Chapter 2. This revised diagram reflects the current state of the system, incorporating new functionalities introduced through this thesis. While the overall architecture remains largely consistent, several key components have been updated. Notably, user authentication is now handled by Globus Authorization, replacing the previously used CI Logon. Additionally, gRPC has been fully deprecated from the system in favor of RESTful HTTP requests, which now serve as the sole method of communication between the front-end and back-end. Although some new features, such as the integration of help videos and an enhanced data export page, have been developed by others within the VISTA research group, they fall outside the scope of this thesis. However, they are included in the diagram to represent the system's complete architecture accurately.

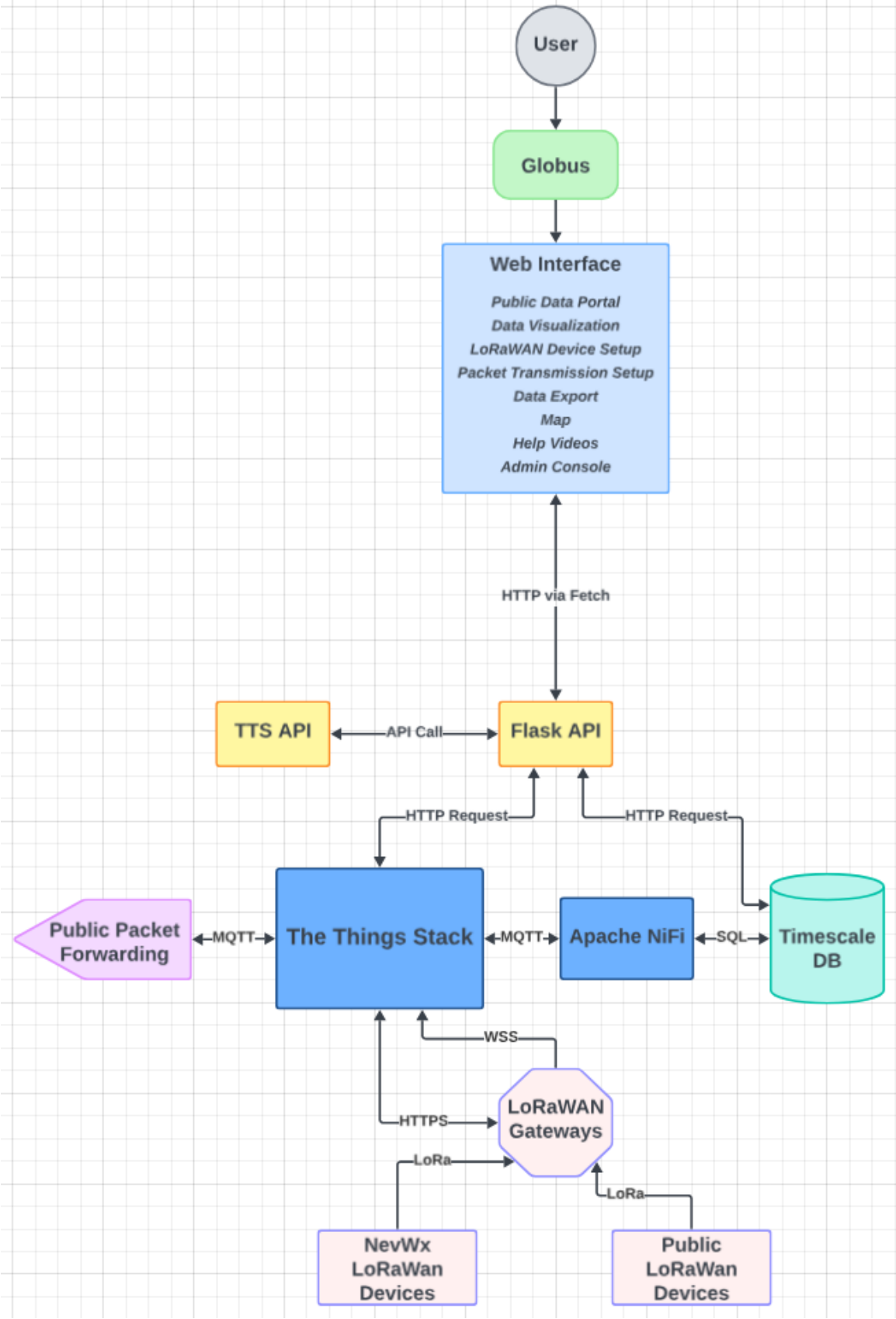

Figure 5.1: The updated system architecture of the VISTA portal, reflecting new visualization components and changes to authentication and communication protocols.

5.4 Backend Architecture & Design

The backend of the VISTA portal is built using the Flask framework, which handles all interactions between the front-end interface and the TimescaleDB database where the sensor data is stored. This backend layer is responsible for receiving user requests from the front-end, including selected periods, sensor IDs, and payload types, and querying the appropriate data accordingly. SQLAlchemy is used for database interactions, allowing the system to dynamically construct queries and fetch relevant results based on user-defined parameters. The backend also provides endpoints to support core features like standard time-range visualizations, aggregation options, clustered data visualization, and retrieval of device types and metadata. Additionally, helper logic has been implemented to standardize unit formats and apply derivation expressions to certain sensor readings, ensuring consistency across payload values shown in the front-end. These components work together to ensure that users can interactively fetch, filter, and visualize their sensor data in near real-time, while also supporting scalable and modular development for future enhancements.

5.4.1 Core Flask Endpoints and Responsibilities

The core Flask endpoints are defined in the `data.py` file. These endpoints are responsible for processing incoming requests, extracting parameters, querying TimescaleDB using SQLAlchemy, and returning results in structured JSON format. These are the endpoints that enable the visualizations in the front-end to display the correct data.

`/get_device_type`

This endpoint is the first interaction point triggered by the front-end when users select a project and begin configuring their data request. It returns the available device types and their associated payload types. These values are extracted by querying the database for the distinct device IDs associated with the user's project and joining them with their recorded readings. Internally, SQLAlchemy constructs the queries using filter clauses based on project ID and device type, and groups the data to return

only the unique identifiers. The returned response is then parsed by the front-end and used to populate drop-downs that allow the user to select which readings they would like to visualize. The endpoint is crucial for ensuring that users only see valid sensor options. Without this, incorrect reading types might be requested, leading to failed visualizations or empty responses. This endpoint is also essential for populating the distinct device ID and device type on the "Map" module of the portal. It allows for the correct devices to be loaded into the staging area of the map, where the user can then select and add them to the map.

`/data`

This endpoint powers the custom date-time range picker on the front-end. If a user is not satisfied with the exactness of time measurements through the time buttons, they will use the date-time picker, which will hit this endpoint as a result. This endpoint will be reached by the front-end regardless of what visualization type the user selects. It accepts parameters such as start time, end time, reading type, and selected device ID(s). SQLAlchemy is then used to create dynamic queries that select rows from the TimescaleDB hypertable within the defined time interval. Optionally, if the user selected an aggregation level (e.g., hourly, daily), it modifies the SQL command to group and aggregate values accordingly. To ensure a valid response, the endpoint includes validation for start and end dates and raises exceptions when the parameters are incomplete. It also calls helper functions to reformat payloads and standardize units (as described later). This endpoint is essential for supporting flexible, user-defined data exploration over large datasets for customized periods.

`/dataButton`

The `dataButton` endpoint supports one-click time window selection for users who prefer to use the predefined buttons on the interface (such as “Last 24 Hours”, “Last 7 Days”, etc.). The front-end passes a button label (e.g., "24h"), which the backend converts into a valid start and end timestamp relative to the current time. The

endpoint uses conditional logic to generate the proper time range and then delegates the data retrieval to the same SQLAlchemy logic used in the standard `/data` endpoint. Due to this, this endpoint will also be reached by the front-end regardless of what visualization type the user selects. This reuse of logic improves maintainability, while the endpoint itself increases accessibility for less technically inclined users who might prefer quick selections over custom inputs.

`get_clustered_data`

This function supports the clustered column chart on the front-end. It is not directly exposed as an endpoint but is called internally when a clustered chart is requested. The function retrieves data within a given time range and then groups it by categories such as "hour of the day" or "day of the week." Using SQLAlchemy, it executes SQL group-by queries and applies averaging or counting functions to summarize trends. The results are structured so that the query will return multiple JSON objects within one large JSON if more than one reading or value is part of the result. For example, users can compare how soil temperature varies throughout the day across multiple sensors or dates. This function introduces complexity due to the need for rearranging and restructuring data, which makes it a distinct pipeline from the basic line graph. It adds analytical depth and helps researchers explore patterns that are otherwise not easily visible in time-series views. The main purpose of this function is to further allow researchers and scientists to divide and customize the data frame that they have initially selected.

Example Interaction

To demonstrate how the backend responds to user interactions, consider Listing 5.2. In this example, a fictitious user is sending a fetch request to render a clustered column chart using the predefined "24H" time button for the 'Soil Temperature' reading from the 'dragino-soil-moisture' sensor. Additionally, the user has applied a filter to include only sensor data collected between 5:00 PM and 10:00 PM, with an

aggregate of every 2 hours. The summary selected is "Average", which will show the average value of based on the mentioned filters. These inputs are passed as query parameters in the request body.

This request is routed through the front-end via the '/dataButton' endpoint, where the payload includes information such as the selected reading, aggregation type, device ID, project ID, and time filter constraints. Once received, the back-end prepares the appropriate SQLAlchemy query and calls the `get_clustered_data` helper function. That function then processes and groups the data by hour of the day or day of the week (based on the user's preference), and returns the formatted result to the front-end as a JSON object. The response, shown in Listing 5.3, is used to render the clustered column chart with the filtered data points displayed accordingly. The response that is provided is not formatted to be directly used in the front-end, meaning that there are validations made there to accommodate and provide renaming functions.

```
POST /dataButton
Content-Type: application/json

{
  "data": {
    "Example_Dev_EUI": ["temp_SOIL"]
  },
  "visualization_type": "clustered",
  "summary": "average",
  "period": "24H",
  "aggregation": "2 Hours",
  "sensor_ids": ["Example_Dev_EUI"],
  "payload_types": ["temp_SOIL"],
  "time_filter": {
    "start": "05:00",
    "end": "7:00"
  }
}
```

Listing 5.1: Example POST request for clustered column chart with hour filter.

```
POST /dataButton
Content-Type: application/json

{
```

```
  "data": {
    "Example_Dev_EUI": ["temp_SOIL"]
  },
  "visualization_type": "clustered",
  "summary": "average",
  "period": "24H",
  "aggregation": "2 Hours",
  "sensor_ids": ["Example_Dev_EUI"],
  "payload_types": ["temp_SOIL"],
  "time_filter": {
    "start": "05:00",
    "end": "7:00"
  }
}
```

Listing 5.2: Example POST request for clustered column chart with hour filter.

```
{
    "data": [
    {
        "dev_eui": "Example_Dev_EUI",
        "payload_type": "temp_SOIL",
        "timestamp": "2024-07-11T17:00:00",
        "unit": "\u00b0C",
        "value": 21.4,
    }
```

Listing 5.3: Sample response returned by the backend after clustered data processing.

This workflow showcases an example of a full front-end to back-end process on the portal, in which the user requests data and receives the correct data based on what is requested. The ability to apply custom filters while selecting predefined time windows greatly enhances the usability of the VISTA portal, enabling users to perform fast and targeted data exploration for real-time monitoring and decision-making. The full visibility of the request details from the front-end side of things will be more closely analyzed and discussed in Section 5.5.

5.4.2 SQLAlchemy and Database Communication

The backend communicates with TimescaleDB using SQLAlchemy [21], a Python-based object-relational mapping (ORM) toolkit that abstracts SQL queries into Pythonic constructs. SQLAlchemy is chosen in this architecture due to its flexibility, support for

advanced query building, and seamless integration with Flask. It allows the backend to generate dynamic queries based on incoming request parameters such as selected devices, time ranges, payload types, and aggregation settings. This is essential for supporting interactive, user-driven visualizations that are tailored to specific queries.

In each endpoint, SQLAlchemy is used to perform key operations that support the data flow between user input and visualization output:

- Filter sensor readings by project ID, device ID, and reading type
- Apply aggregation (e.g., hourly, daily, weekly) for line chart visualizations
- Group and average data across categorical dimensions (e.g., hour of day, day of week) for clustered column charts
- Join sensor metadata with associated payloads for front-end rendering and dropdown population

This query logic is typically defined inside Python functions or inline within the Flask route handlers. SQLAlchemy compiles the query into raw SQL before submitting it to TimescaleDB, which is an extension of PostgreSQL optimized for time-series workloads. By leveraging TimescaleDB's performance-enhancing features—such as automatic time-based partitioning and hypertables—the system is capable of querying hundreds of thousands of records quickly, which is essential for maintaining frontend responsiveness. Another benefit of SQLAlchemy in this architecture is its protection against SQL injection and invalid queries. This is particularly important in the VISTA portal, where many query parameters originate from user input on the frontend. Query composition is handled with type-checked filters, ensuring that malformed or unauthorized SQL does not reach the database.

The integration between SQLAlchemy and helper utilities, such as those in `helper.py`, enhances query results by appending derived unit types, applying normalization rules, and formatting response payloads. These enriched responses are then converted into JSON and returned to the front-end for visualization. The back-

end's modular structure, powered by SQLAlchemy, also allows the portal to scale. For example, the same querying framework supports both `/data` and `/dataButton`, reducing code duplication while preserving flexibility. This is especially useful in functions like `get_clustered_data`, which build more complex group-by queries for categorical analysis.

Overall, SQLAlchemy not only facilitates the communication between Flask and TimescaleDB but also serves as the backbone of backend logic that enables the customizable and responsive data exploration workflows presented in the frontend visualization modules.

5.4.3 Unit Standardization and Reading Classification

The `helper.py` file plays a supporting role by offering utility functions used throughout the Flask backend. One of the key functionalities developed during this thesis was the standardization of units for sensor readings. Before the thesis implementation, the data visualization provided values for each of the readings, but did not include any units alongside the values. This caused a major issue when researchers and citizen scientists using the platform fetched multiple readings for one graph, as the absence of units prompted occasional confusion. Sensor payloads vary between device manufacturers, so having a consistent unit mapping is essential for presenting data in a clear and comparable format.

This is accomplished by using a unit translation dictionary within `helper.py`, which maps various raw unit strings into a standardized form (e.g., mapping "C", "celsius", and "degC" all to "°C"). The backend also supports transformations using lambda functions for derived metrics, allowing raw readings to be converted into more interpretable values. Another utility is the classification of reading types. This determines whether a sensor's payload type is numeric, categorical, or derived, and enables the front-end to populate dropdowns with appropriate options. Without this classification logic, the front-end would not be able to interpret the available readings or display them with contextual accuracy.

5.4.4 Integration with the Front-end and Visualization Triggers

Every backend endpoint has a corresponding trigger in the front-end that determines when it should be called. For example:

- When a user selects a device/reading type, a call is made to `/get_device_type`
- When the user clicks a time button and requests to fetch the visualization, the front-end calls `/dataButton`
- When the user sets a date range in the custom picker and requests to fetch the visualization, the front-end calls `/data`
- When clustered column visualization is selected, the same workflow is applied in the previously mentioned endpoints, but the backend also calls the logic defined in `get_clustered_data`

This tight coupling ensures that all visual elements are backed by accurate, dynamically retrieved data. The structured flow between front-end triggers, backend endpoints, and database results guarantees that the visualization interface reflects the user's interest accordingly. Another benefit of the way that the integration of the front-end and visualization triggers is structured is that they are scalable. If another visualization type is added to the portal, an addition of another function similar to the `get_clustered_data` would be added, without changing the routine workflow that is followed by the current visualization types.

5.5 Front-end Architecture & Implementation

The front-end of the VISTA portal is developed using the React framework and The Things Stack source code and is responsible for managing all user-facing interactions, interface logic, and the dynamic rendering of visual content. It serves as the bridge between users and the underlying data infrastructure, enabling intuitive exploration

of sensor datasets through visual and spatial tools. Built on top of Material UI (MUI) and integrated with Plotly for charting, the front-end allows users to configure custom visualizations, apply filters, and select various chart types in a responsive and modular environment. It communicates directly with the Flask backend through RESTful HTTP requests, sending user-defined query parameters and rendering the returned data in real time. Two major components introduced in this thesis, the Visualization Module and the Map Module, enable temporal analysis and geographic representation of the sensor locations. These components are designed to immediately reflect changes based on user input, offering a seamless and interactive experience for both casual users and institutional researchers. The architecture of the front-end is designed to support a modular component design, making it possible to easily add new visualization types and advanced filtering options when necessary.

5.5.1 Visualization Module

The Visualization Module is the most significant feature introduced during this thesis and serves as the primary interface for users to explore and analyze sensor data visually. Built entirely within the front-end of the VISTA portal, this module supports two distinct visualization types: time-series line charts and clustered column charts. Each chart is designed to accommodate different analytical goals and is rendered using Plotly.js, which offers a highly interactive, browser-native charting experience with minimal performance overhead. This section outlines how each chart type functions, the filtering and aggregation options available, and the architectural decisions that shaped their implementation.

To maintain modularity and efficiency, the user interface—developed using Material UI (MUI)—is consistent across both visualization types. As described in Section 5.4, the `/get_device_type` endpoint provides the necessary metadata to populate dropdown selections for devices and their available readings. The same interface design is applied to temporal selection tools, where users can either choose a predefined period (such as “24H” or “7D”) using time buttons or define a custom date-time

range via a picker. These options ensure a uniform experience, regardless of the chart type.

Each time selection type also incorporates a corresponding aggregation method. For instance, time button presets come with default aggregation settings (such as hourly or daily intervals) to prevent performance issues and reduce visual clutter caused by too many data points. These default settings are defined individually for both the line chart and clustered column chart and are discussed further within their respective subsections.

Time-Series Line Chart

The time-series line chart enables users to explore continuous sensor readings over a defined period. Once users select one or more devices and associated payload types, they can choose to retrieve data using either a predefined time button or the custom date-time picker. Based on this selection, the front-end sends a query to the appropriate backend endpoint, `/dataButton` or `/data`, along with parameters for device ID(s), reading type(s), project ID, and aggregation level. To ensure that all required values are entered, a validation utility prevents submissions with missing or conflicting selections, which is discussed further in Section **??**.

After the backend returns the requested dataset, the visualization is generated using Plotly's 'scattergl' mode, which creates a line chart by plotting the data as a scatter graph with line interpolation. While Plotly does not have a dedicated "line graph" type, setting the mode to "lines" within 'scattergl' achieves this effect. This configuration also leverages WebGL rendering, which offloads the graphical processing to the user's GPU, making it highly effective for datasets containing tens of thousands of points. This improvement addresses previous system limitations that caused substantial visual lag in SVG-based rendering for large datasets. Although WebGL is preferred for heavy data visualizations, Plotly retains SVG rendering as a fallback option for smaller datasets, offering additional flexibility. The Plotly chart includes the following built-in features:

- Zooming and panning for navigating dense data segments
- Hover tooltips that show timestamps, values, units, and device metadata
- A range slider to define the visible window over large datasets
- A lasso selector for highlighting and isolating data points
- A download tool for exporting the chart as a PNG image
- A reset-axis button to return the graph to its original view

These features are configured directly within the Plotly layout and configuration objects and can be selectively toggled for each visualization type depending on the use case. For time-series visualizations, the ability to zoom, hover, and export snapshots allows researchers to efficiently analyze trends while maintaining usability.

Once the graph is rendered, Plotly dynamically labels the axes using the unit information returned from the backend. This ensures that users always see measurements with the correct context, especially when multiple reading types are plotted together. Each reading is represented with a distinct color and is listed in a native Plotly legend beneath the graph. Users can toggle the visibility of individual readings by clicking on their respective names, allowing them to isolate specific trends in crowded plots. Additionally, as users hover over a data point on the line, a tooltip will appear showing the precise timestamp and value, complete with unit and device name. This interactivity makes it easy to analyze specific events or fluctuations within the selected range.

Clustered Column Chart

The clustered column chart is a more advanced visualization type that enables users to compare sensor readings across categorical time dimensions such as hour of day or day of week. Unlike the time-series chart, which presents a continuous flow of data points, the clustered chart groups values into discrete categories, offering a high-level summary that is useful for identifying trends or comparisons between two or more

readings. Once the user selects their device(s), reading(s), and preferred period, via either the predefined time buttons or custom date-time picker, an additional set of options becomes available. These include the selection of an aggregation type (either hour of day or day of week) and a summary type. These controls are critical in transforming the selected time-series data into a structured format suitable for categorical comparison. Since the clustered column chart presents and compares aggregated values as individual columns, the summarize options compute a single representative value based on the user's selected summary method. The summary types that are available for users are the following:

1. **Average:** Find the average value of the requested dataset.
2. **Minimum:** Find the minimum value of the requested dataset.
3. **Maximum:** Find the maximum value of the requested dataset.

In addition to these summary dimensions, the clustered column chart includes a feature for advanced filtering. This allows users to narrow the dataset by specifying a time-of-day window (e.g., only show values between 5:00 PM and 10:00 PM) or limiting the visualization to certain days of the week (e.g., only showing values for Saturday and Sunday). These added advanced filters allow users to have more variety in their analysis options using the clustered column chart. This is in contrast with the line graph, which does not offer any advanced filtering options.

Once the grouped data is received, the frontend renders the visualization using Plotly's 'bar' chart type, grouped by the selected summary category. If multiple devices or payload types are selected, Plotly automatically clusters the bars side by side within each category, making it easy to compare values across sensors or locations. Each of the bars will also be color-coordinated in a similar way to the line graphs, ensuring that the user can differentiate each reading when comparing them. Although the clustered chart reuses many of Plotly's built-in interactivity features, such as hover tooltips, zoom, export to PNG, and axis resetting, it distinguishes itself by the

comparative layout and filter-driven customization. Each bar in the chart displays the average (or other selected aggregation type) of all data points falling within that time slot, rather than showing raw values. This helps users understand trends such as peak soil temperature by hour, or how humidity fluctuates across different weekdays. As with the time-series chart, readings are labeled with units, and users can toggle visibility for each reading directly from the chart's legend. The hover interaction reveals not only the value and unit but also the grouped dimension of date and time, providing full context for interpreting the visualized data.

5.5.2 Map Module

The Map Module provides a geospatial overview of all sensor locations available to the user, offering a complementary perspective to the temporal analyses provided by the Visualization Module. Rather than focusing on time-based readings, this module centers on geographic context by rendering a dynamic map that displays markers corresponding to each sensor device registered in the user's account. This functionality was developed using React Leaflet, a widely adopted library for rendering Leaflet maps in React applications, and connects directly to backend APIs for real-time device metadata retrieval.

Upon mounting the map visualization component, the front-end initiates a call to the backend `/get_device_type` endpoint. This endpoint returns an object that includes the `dev_eui` of each sensor along with associated metadata, which populates the individual sensor pop-up card with the most recent reading. Once this response is received, the module iterates over the data and populates the map with individual markers at the corresponding coordinates. Each marker is interactive: clicking on a marker opens a pop-up that displays the sensor's device ID and type, as well as the most recent reading. Also embedded within the pop-up is the ability to set the new location of the sensor or to view the "End Devices" module within the projects tab.

The underlying logic of the Map Module is designed to handle a wide variety of sensors in a scalable and efficient manner. Device markers are rendered condition-

ally—only after successful API resolution—to avoid rendering incomplete or undefined data. Additionally, a fallback behavior is implemented to filter out devices that may lack valid GPS coordinates, ensuring that the map only displays accurate and meaningful information. This is critical in maintaining the map's integrity, particularly in research scenarios where geographic precision is essential.

Functionally, the Map Module offers several enhancements to improve the user experience:

- **Zoom and pan controls** allow users to explore the geographic spread of sensors in granular detail.
- **Cluster control logic** ensures that maps with densely packed sensor deployments remain readable and visually uncluttered.
- **Automatic map centering and bounding box fit** are used after all device markers are loaded, ensuring an optimal initial view.
- **Unmarked & Marked Sensors** provide two different state type for sensors. All sensors that have not been assigned a location on the map are listed in card format underneath the map, with the ability to set the location. If a sensor location is set, the card is removed from under the map and replaced by a point on the map resembling the new set location.

From an architectural perspective, this module enhances the spatial understanding of sensor placement and assists in linking sensor behavior to environmental conditions or location-based patterns. It also serves as an entry point for future enhancements, such as geofencing logic, sensor grouping by environmental zone, or the ability to click a sensor and directly launch a filtered time-series chart.

While the Map Module is separate from the time-based Visualization Module, it shares the same backend data source and endpoint logic, reinforcing the integrated nature of the VISTA portal's architecture. In particular, both modules rely on the

backend's consistent metadata formatting, making them tightly coupled despite serving distinct visualization purposes.

5.5.3 User Support

A core focus of this thesis was to enhance the usability and reliability of the VISTA portal's front-end. In particular, the user support features implemented throughout the Visualization and Map Modules were designed to minimize unnecessary complexity, reduce error, and promote an intuitive experience for both technical and non-technical users. This section outlines key design decisions and implementation details that contribute to overall usability, focusing on form validation, interface responsiveness, and guided interaction design.

One of the most critical user support mechanisms introduced in the Visualization Module is the implementation of form validation logic. Given the large number of parameters users can configure, such as device ID, reading type, time interval, and aggregation method, validating user input before submission was necessary to prevent incomplete or conflicting queries. A lightweight yet effective validation utility checks that the required combinations of device(s), payload type(s), and time selections are properly entered. If any essential input is missing or invalid (for example, if a user tries to submit a query without selecting a reading type or time range), the interface will immediately present a contextual alert to inform the user of the issue. This prevents backend calls from being made with malformed or incomplete requests, saving both computation time and user frustration.

Another usability enhancement is the inclusion of a chart toggle button component, which allows users to switch between the time-series line chart and the clustered column chart with a single interaction. Rather than requiring users to navigate to separate pages or reconfigure their parameters, the toggle preserves existing selections such as device and reading types, and applies them to the newly selected chart type. This seamless switching supports exploratory data analysis by making it easier to evaluate the same dataset from multiple perspectives, without repetitive configura-

tion or loss of context.

The user interface design also prioritizes clarity and guidance through consistent layout structure and accessible component grouping. All input selections, such as devices, readings, time range, and aggregation method, are grouped logically and placed directly above the visualization window. Each section is clearly labeled, and default values are pre-populated wherever applicable to assist users in getting started without needing to understand every detail of the system. The time button presets (such as "1H", "7D", or "1M") are configured with default aggregation values specifically chosen to prevent visual overload in the chart. These defaults not only improve performance but also simplify the decision-making process for users unfamiliar with data aggregating concepts.

Across both visualization types, the inclusion of built-in Plotly features like zooming, hovering, range sliders, and export tools further enhances user empowerment. These interactive tools are natively integrated but selectively configured depending on the chart type, ensuring that only contextually relevant options are presented to the user. For instance, lasso selection is included in time-series visualizations but is omitted from the clustered chart where it does not apply. This curated experience avoids visual clutter while providing meaningful control to the user.

The Map Module also incorporates important user support features, particularly for managing sensor locations. When a device has not yet been assigned a geographic coordinate, it is listed as a card beneath the map with a call-to-action to set its location. This visual distinction between unmarked and marked sensors improves clarity and provides a structured workflow for managing devices. Once a location is set, the sensor is removed from the unmarked list and immediately placed on the map, reflecting the change in real time. The same goes for the reverse workflow, which can also remove the marked location and place it back underneath the map. The interactive marker popups not only show the device name and type but also include a direct link to the associated sensor's details within the project's "End Devices" tab, offering an efficient navigation path between different parts of the portal.

5.6 Software Challenges

Although the development of the VISTA portal's front-end proceeded steadily, a number of software-related challenges emerged, particularly during the later stages of building and integrating the data visualization features. Two key obstacles—real-time data access and the implementation of advanced filtering for categorical visualizations—proved especially difficult due to the nature of the system's architecture and its reliance on a live sensor network. This section explores these two primary challenges in depth.

5.6.1 Restricted Access to Post-Migration Data Streams

One of the most significant limitations encountered during the development of the Visualization Module was the inability to access real-time sensor data during the final months of implementation. This issue arose due to a structural change in the system's backend infrastructure: a migration of the TimescaleDB cluster and associated Pub/Sub pipeline that separated the production and development environments. Before November 2024, the development and production codebases shared the same database instance, which allowed full access to live data for frontend testing and chart validation. However, following the migration, all newly streamed data from deployed sensors and gateways was routed exclusively to the production environment.

As a result, the development environment was no longer able to access incoming sensor readings after the migration date. This presented a major hurdle when building and testing new features, particularly the time button shortcuts and the custom date-time picker used throughout the Visualization Module. While the interface was functional, testing queries for recent data (i.e., the past 24 hours or current week) yielded no results during local development. To work around this constraint, all frontend data queries had to be carefully crafted to fall within historical time ranges before November 2024, when data was still being shared across environments. For example, testing a "24H" time button required manually adjusting the default logic

to request data from October 2024 or earlier. Similarly, the custom date-time picker had to be used in a way that intentionally avoided current or future timestamps, which would return an empty dataset.

Although the interface technically allowed any time range to be selected, user-level validation and visual feedback had to be adjusted to gracefully handle empty responses. This added extra complexity to the development and quality assurance process. Because of this limitation, features dependent on real-time behavior, such as continuous sensor monitoring or confirmation of recently received readings, could not be evaluated until the application was deployed into the production environment.

5.6.2 Complexities in Implementing Advanced Filtering

Another major development challenge emerged during the implementation of the advanced filtering logic for the clustered column chart. Unlike the time-series line chart, which only required continuous data over a selected time range, the clustered column chart had to group and aggregate values by categorical dimensions (e.g., hour of day or day of week) while also applying optional constraints such as time-of-day windows and day-of-week inclusion filters. This resulted in a highly conditional backend workflow that was prone to query errors if parameters were missing, improperly formatted, or logically incompatible.

One of the more persistent problems during development stemmed from the order in which filters were applied and how they interacted with each other. For instance, when both a custom date range and a time-of-day filter were selected, the backend initially failed to correctly apply both layers of filtering. In early implementations, the time-of-day filter would override the base date range, causing the grouped aggregation to return inconsistent or incomplete results. Additionally, when the day-of-week filter was introduced, some combinations of days and time windows resulted in empty query results, even when valid data existed in the database, because of subtle Python logic errors in how "strftime" and time comparisons were combined in the query pipeline. "Strftime", which filters based on the value, by including "%a", which provides a

value back to the front-end indicating the day of the week in text form.

Another challenge involved ensuring consistency between the frontend and backend interpretations of filter values. For example, time-of-day windows selected in the user interface had to be converted from local browser time into UTC before being passed to the backend. Failing to do so resulted in time offsets that excluded relevant data points from being returned. Similarly, ensuring that the day-of-week indexing aligned between JavaScript (where Sunday is "0") and PostgreSQL (where Sunday is also "0") required explicit conversions and normalization logic to avoid mismatches.

The development process was iterative, with many rounds of testing and logging needed to identify and resolve these logic bugs. Once the individual filters were functioning correctly, an additional layer of complexity was introduced by the need to format the output data so that it could be correctly clustered by Plotly on the frontend. This required the backend to restructure the response format, organizing grouped values by category (e.g., "Monday", "Tuesday", "Wednesday") and aligning them with the appropriate summary statistics (average, min, or max). Ensuring the structure matched what Plotly expected for grouped bar charts was nontrivial, especially when certain devices or payloads had no data for one or more categories, which caused rendering artifacts in early testing.

Despite these obstacles, the filtering workflow was eventually stabilized through a combination of backend refactoring, frontend validation improvements, and careful synchronization of default values. These changes made it possible to offer an advanced filtering interface that is both powerful and intuitive for users, but the process of reaching that point involved considerable trial and error across both the frontend and backend codebases.

Chapter 6

Prototype in Action

This chapter presents a visual walkthrough of the prototype developed as part of this thesis, highlighting key pages and functionalities of the VISTA portal that were either introduced or significantly enhanced. The selected interface components focus primarily on the data visualization and map modules, demonstrating how sensor data can be explored through interactive graphs and spatial representations. Each figure provides a snapshot of the portal in action, showcasing how users can interact with time-based filters, rendering mechanisms, advanced options, and geospatial tools.

6.1 Interface Showcase

Figures 6.1 and 6.2 introduce the primary ways in which users define the time range of their queries. In Figure 6.1, the interface displays the "Time Button" options, allowing users to quickly select from predefined time intervals such as "1H", "24H", or "7D." Each button also applies a default aggregation value to strike a balance between performance and data clarity. In this example, the user has selected the device `"dragino-soul-moisture"`, the reading `"Soil Temperature"`, and a 7-day time interval with an aggregation of every hour.

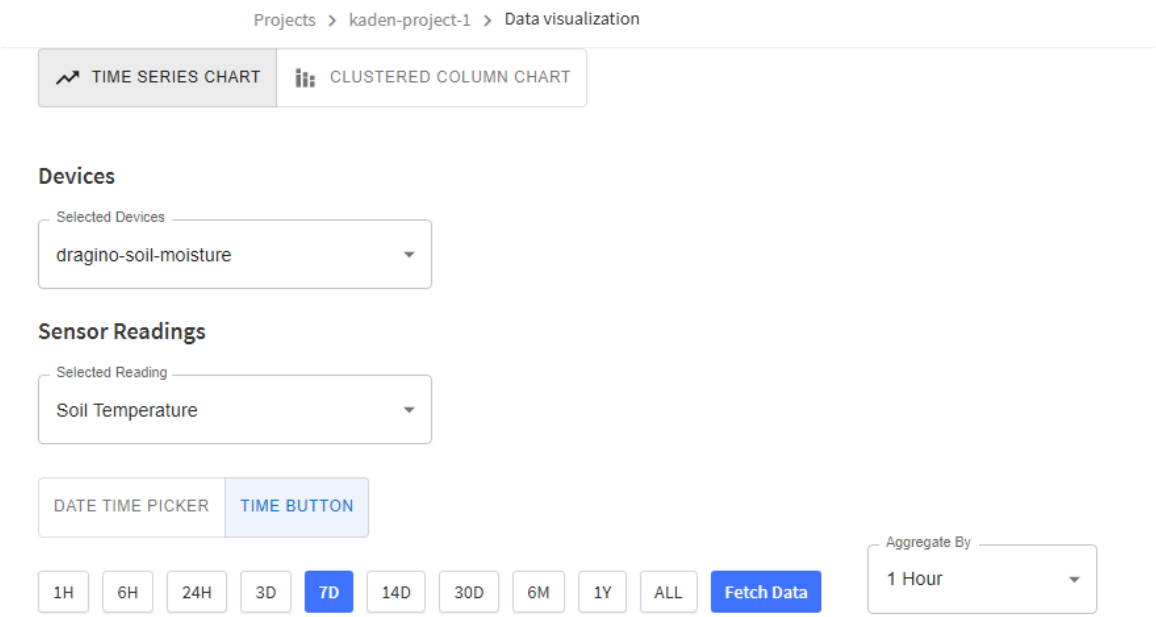

Figure 6.1: Data visualization page showing the "Time Button" options for the time-series line graph.

In contrast, Figure 6.2 highlights the custom date-time picker, which gives users more granular control over the start and end of the desired time range. Although most of the user-selected options are the same as in the previous figure, the user has defined that the 7-day time interval falls between November 1 and November 7. This method is advantageous when investigating specific time windows, ensuring flexibility in exploratory analysis.

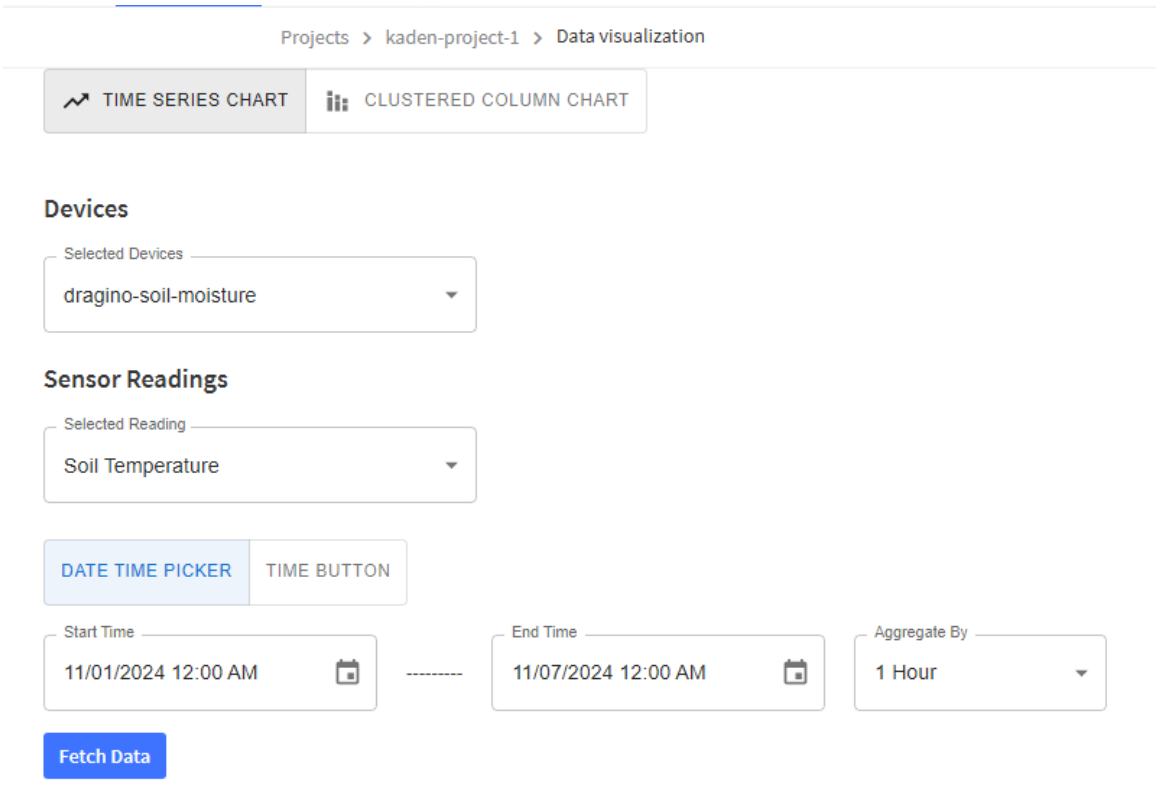

Figure 6.2: Data visualization page showing the "Date-Time Picker" options for the time-series line graph.

Following the selection of input parameters, Figure 6.3 displays the rendered result of a time-series line graph. This figure demonstrates the output of fetching the data from Figure 6.2. The chart is produced using Plotly's `scattergl` rendering mode, which utilizes WebGL for optimized performance on large datasets. Interactive features such as zooming, hovering tooltips, axis resetting, and export tools are all visible. These enable users to investigate the dataset visually with precision and responsiveness. The axes are dynamically labeled using metadata returned from the backend, and each reading is represented by a unique color with legend toggles to isolate individual series. This figure demonstrates the strengths of utilizing the line graph to visualize trends and patterns, as it paints a picture of the fluctuations of soil temperatures throughout different times of the day.

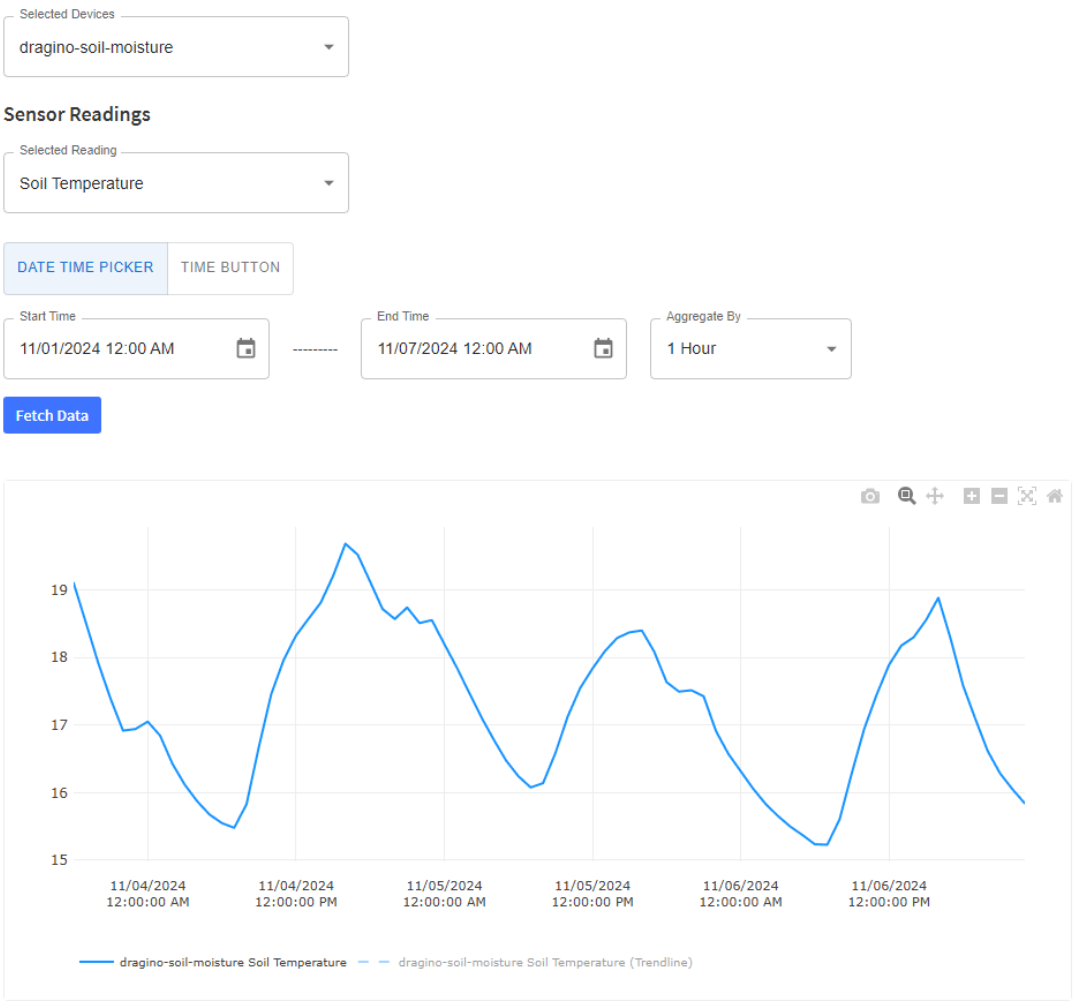

Figure 6.3: Data visualization page showing a rendered time-series line graph based on the user-selected inputs.

In addition to time-series plots, the visualization module supports categorical data analysis via clustered column charts. The user can select what statistical measure they want to filter the data by, which in this case is set to average. Figure 6.4 highlights the "Advanced Filter" options available for this chart type. Users may filter their dataset by specifying custom time-of-day ranges and selecting specific days of the week, enabling a more refined view of the data. These filters were added to accommodate use cases where trends are only relevant during certain temporal windows, such as work hours or weekends. The clustered column chart also includes new default aggregation values for the specified period, reducing the possibility of

visual crowding, which becomes a real issue with large datasets within clustered column charts.

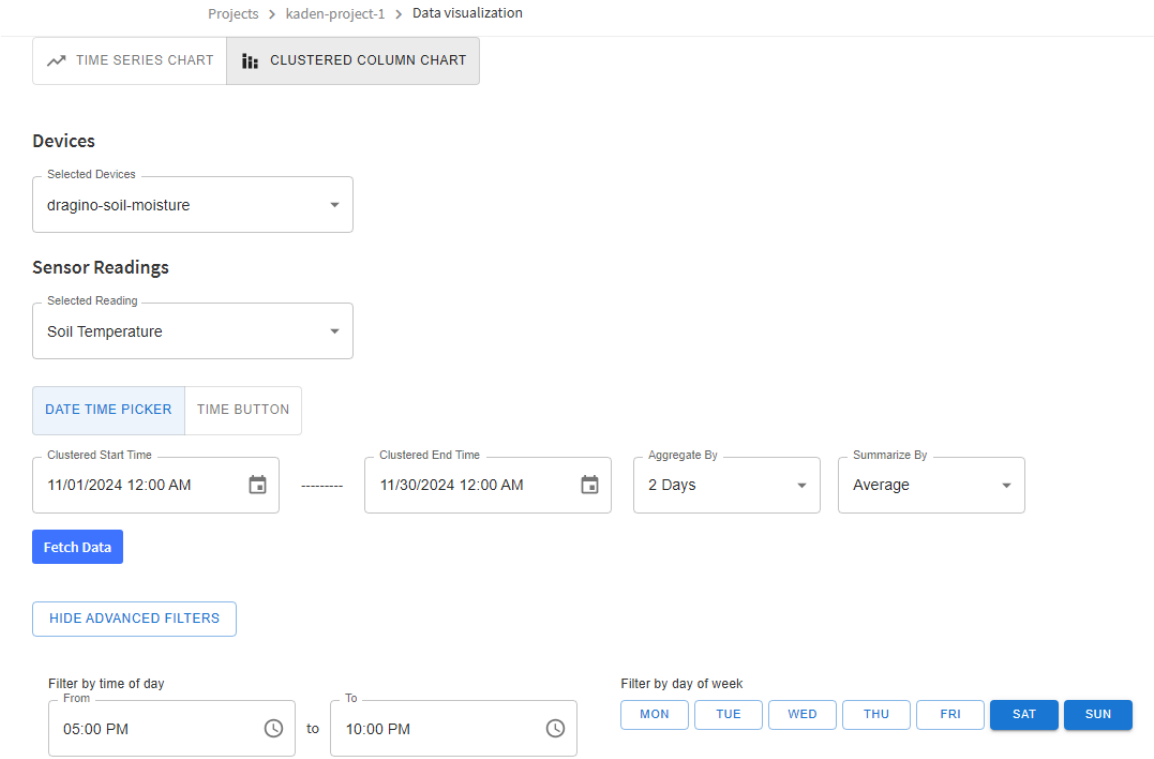

Figure 6.4: Data visualization page showing the "Advanced Filter" options for the clustered column chart.

Building on this, Figure 6.5 shows a fully rendered clustered column chart that incorporates advanced filtering and multiple sensor readings. Columns are grouped categorically by hour or day, based on the aggregation type selected. Each column represents a summarized value (e.g., average or max), and side-by-side bars provide a comparative layout for examining different sensors or readings. In this example, the user has selected the same device but with two readings: `"Soil Temperature"` and `"Soil Moisture"`, and a 14-day time interval with an aggregation of every day. The user has also selected the advanced filters of 5:00 to 10:00 PM and only Saturday and Sunday. Color coordination and legend interactivity mirror the time-series graph for a consistent experience, which is shown by the blue and purple columns representing

both of the selected readings.

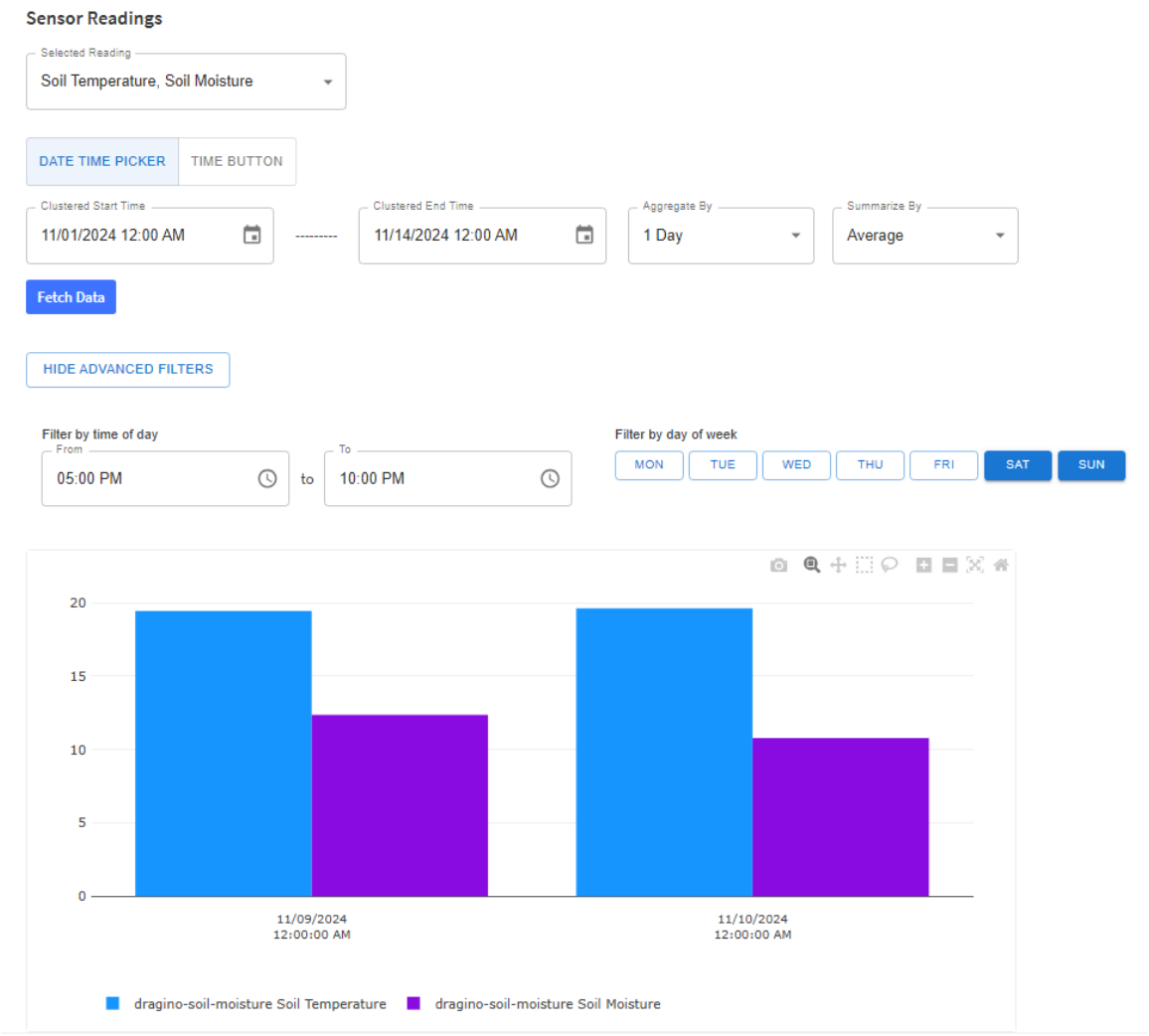

Figure 6.5: Data visualization page showing a rendered clustered column chart with multiple readings and advanced filters applied via user-selected inputs.

An additional feature shown in Figure 6.6 is the trend line option available within the time-series chart. This feature overlays a smoothed regression line onto the raw data, helping users identify general trends or patterns across the selected time range. The trend line dynamically adjusts based on the visible dataset and can be toggled on or off for convenience.

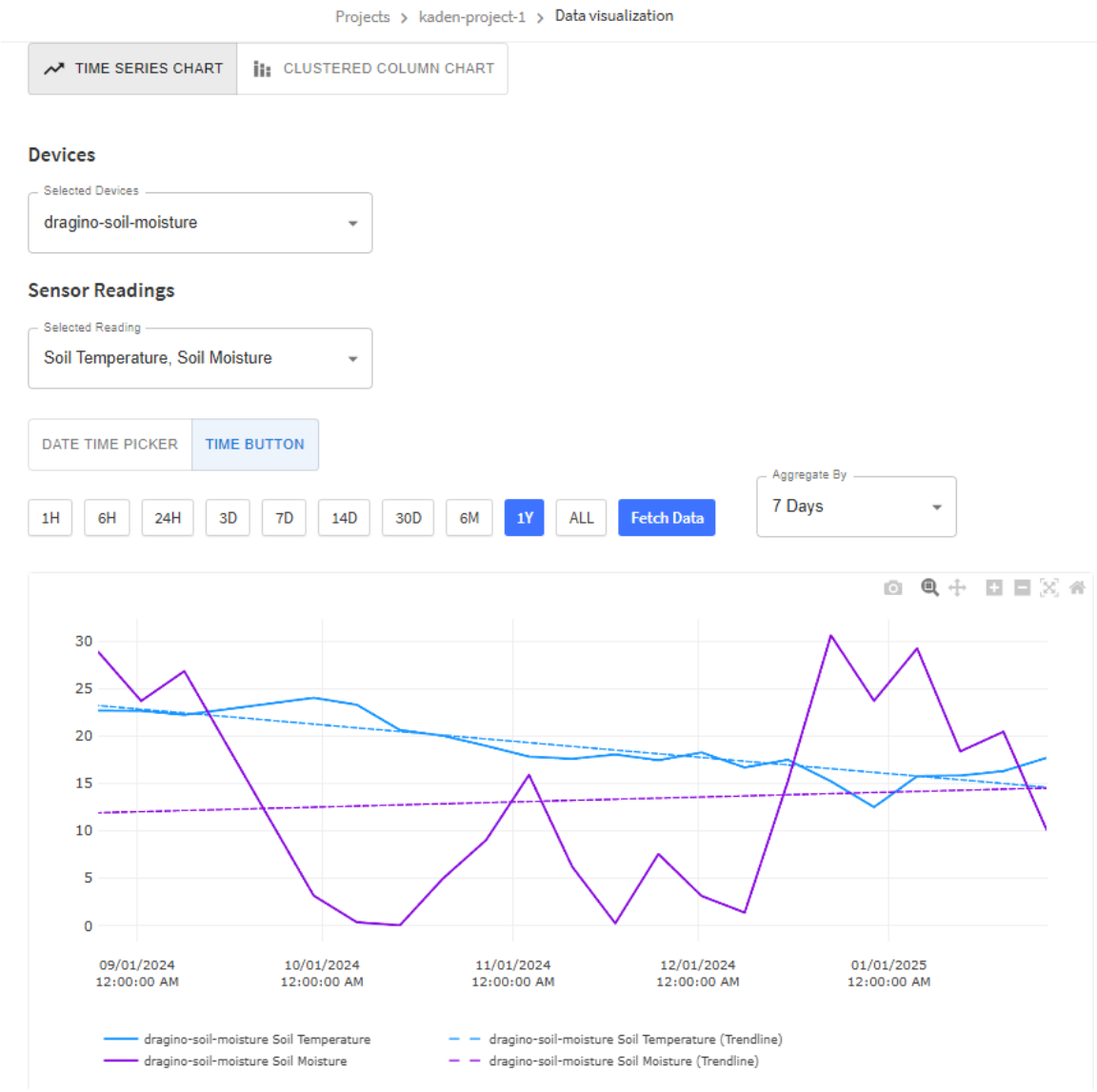

Figure 6.6: Data visualization page showing the trend line functionality on the time-series line graph.

Finally, Figure 6.7 presents the Map Module, which provides a spatial view of sensor deployment. The interface displays both marked and unmarked sensors, allowing users to interact with device markers and set new geographic positions directly through the map interface. Popups display metadata and the most recent readings, while card-based listings beneath the map offer controls for unmarked devices. This view supports geographic validation and enhances the interpretability of sen-

sor placement relative to their readings. In this example, the user has one of their sensors `"dragino-soil-moisture"` marked on the map with its location, while their unmarked sensor `"laird-temp"` remains underneath the map awaiting marking.

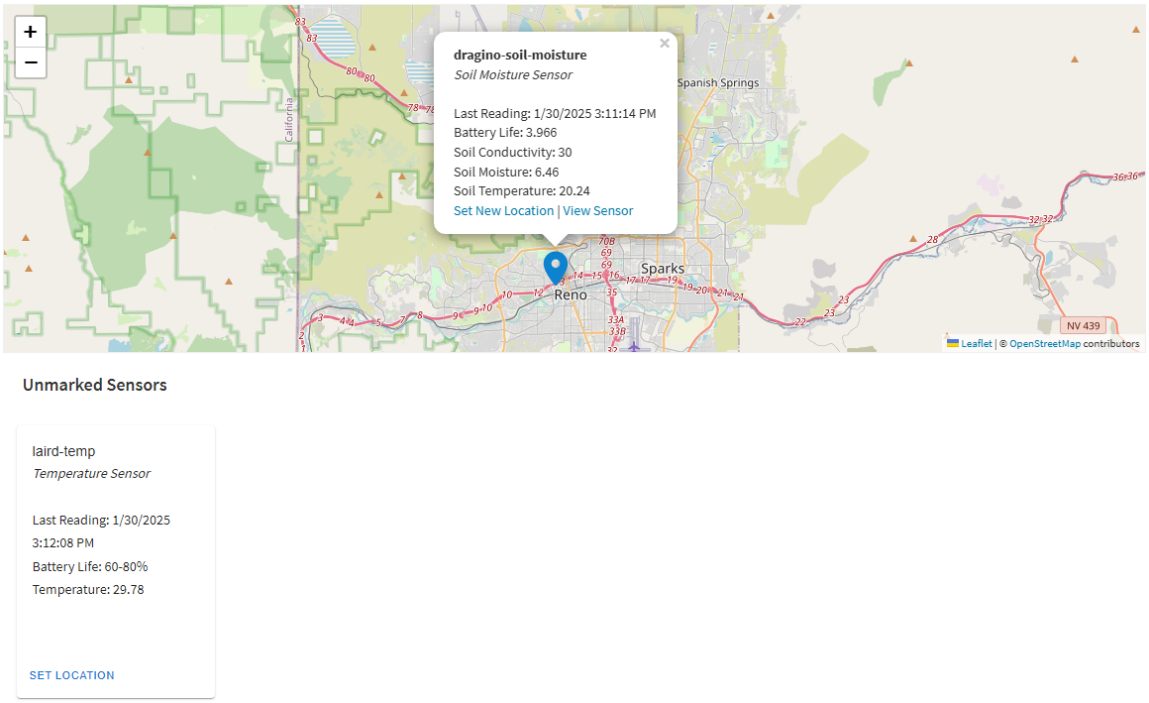

Figure 6.7: Map page showing one marked and unmarked sensor.

Chapter 7

Visualizations Evaluation

7.1 Visualization Evaluation

This chapter evaluates the effectiveness of the newly implemented data visualization features, which were developed to support users in better understanding sensor information within the VISTA platform. These visualizations, including the dynamic graph and map components, were designed with clarity, usability, and interactivity in mind. The goal was to determine how well these tools helped users identify patterns, compare values, and make decisions based on the environmental sensor data available to them. Section 7.2 outlines the main objectives behind the evaluation, focusing on the criteria used to measure visualization effectiveness. Section 7.3 describes the experimental setup, including participant selection, tasks, and evaluation methods. Section 7.4 presents the results gathered from participant performance and feedback, and discusses some participant comments and recommendations. Finally, Section 7.5 draws a summary based on the findings of the user study and the implications that follow.

7.2 Objective

This user study focuses on evaluating the usability and analytical effectiveness of the newly developed visualization and map modules integrated into the VISTA portal. The primary goal is to assess how well users are able to interact with and interpret environmental sensor data using the two visualization types offered on the portal's

data visualization page: the time-series line graph and the clustered column chart. A secondary objective is to determine how intuitively users can engage with the spatial map interface, particularly when interacting with sensor location markers and associated metadata. The user study has been approved by the Institutional Review Board (IRB), and its application number is 2262166-3 at the University of Nevada, Reno.

The goal of the study is not to compare the visualizations against each other, but rather to assess their strengths across different task types and explore whether each can effectively support environmental data interpretation. The time-series line graph is intended to assist users in recognizing trends, patterns, and fluctuations over defined periods. It is particularly useful in identifying highs and lows in sensor values across daily, weekly, or monthly intervals. Tasks associated with this visualization focus on detecting temporal peaks, understanding correlation across sensor readings, and observing shifts in overall trends within selected date ranges. In contrast, the clustered column chart is designed to support comparisons between values across categorical groupings such as days of the week or time blocks within a day. It also allows for statistical summarization using functions such as average, minimum, and maximum. The chart supports advanced filtering options, including day-of-week and time-of-day constraints, and tasks designed for this visualization center around interpreting summarized values and comparing different readings in isolated or combined views.

Beyond the visualizations, the study also includes one geospatial interaction task to evaluate the usability of the portal's map module. This task involves placing a sensor marker on the University of Nevada, Reno campus and identifying the most recent reading associated with that sensor. This helps determine how intuitively users can navigate the map interface, engage with interactive elements like markers and popups, and understand the geographic distribution of sensor deployments. This chapter details the study setup, including participant count, task structure, procedures, and evaluation metrics. It also presents the full range of study tasks that were performed, each aimed at testing a specific interaction or interpretive goal. Finally,

the results of the study are analyzed and discussed, offering insight into how the new portal features support usability, data exploration, and analytical reasoning.

7.2.1 The Role of Visual Interpretation in Environmental Monitoring

Effective environmental monitoring depends not only on sensor deployment and data collection, but also on the clarity with which data can be interpreted by end users [12]. The VISTA portal is designed with the goal of transforming raw environmental data into meaningful insights through interactive visualizations. These visualizations, including the time-series line graph and clustered column chart, serve distinct yet complementary roles in promoting user understanding. The line graph excels at revealing temporal trends and fluctuations across chosen periods, allowing users to track sensor readings and identify daily or seasonal patterns. Meanwhile, the clustered column chart facilitates comparison of grouped data, such as evaluating multiple readings across categories like time-of-day or day-of-week. Both are crucial in ensuring that users, regardless of technical expertise, are able to draw accurate conclusions from the data.

As environmental sensor networks expand, platforms like VISTA must meet the needs of a broader, more interdisciplinary user base. This includes students, researchers, and public users who may lack technical training in data analytics but still require meaningful access to sensor data. The inclusion of a map module, for example, emphasizes geospatial interpretation by enabling users to visually associate data with real-world sensor locations. The usability study presented in this chapter is structured not just to assess whether participants can perform specific tasks, but to determine whether the visual tools provided by the portal make it easier to reason about environmental conditions. This ensures that the system remains aligned with its broader purpose: delivering actionable environmental insights through a platform that is intuitive, fast, and reliable.

7.3 Experimental Setup

7.3.1 Participants

The user study involved a total of twelve student participants from the University of Nevada, Reno. Eight of the participants were enrolled in undergraduate programs, while the remaining four were graduate students. The group represented a range of academic disciplines, with 25% coming from the Computer Science and Engineering program and the rest from fields such as Mechanical Engineering and Nursing. Overall, 58% of participants were enrolled in engineering-related majors, allowing for a moderate range of technical familiarity. The demographic makeup consisted of 10 male and 2 female participants, with ages ranging from 19 to 26. This participant pool was intentionally selected to reflect the experience of a typical college student, rather than focusing solely on those with technical or data analysis expertise. Including individuals outside the computer science field enabled a more holistic evaluation of the VISTA portal's usability, accessibility, and effectiveness for a broader and less specialized audience.

7.3.2 Apparatus and Study Setup

The user study was conducted using the live VISTA portal hosted on a locally served development environment. This version of the portal reflected the latest features developed through the thesis, including the finalized time-series line graph, clustered column chart, and interactive map module. No additional instrumentation was introduced for task tracking, as performance and usability were primarily measured through user responses to the tasks and post-study feedback. The data used in the study was based on real sensor data collected during the testing phase of the portal and reflected typical fluctuations in environmental readings such as soil temperature, soil moisture, and air temperature. To ensure consistency, all users interacted with the same underlying dataset during their session.

The study took place in multiple locations, one being the William N. Pennington

Engineering Building at the University of Nevada, and the other being at the library or home of the participants. This user study was conducted utilizing the same workstation for all participants, providing a mouse and keyboard for consistency. The laptop was connected to a standard-sized monitor, keyboard, and mouse to replicate a common desktop computing experience. This fixed setup eliminated the possibility of hardware-based performance discrepancies. Each session was monitored in-person by the researcher, who introduced the system and observed the user interactions. All participants accessed the tasks from a printed task list provided during the study, which allowed them to reference the instructions without needing to navigate away from the main portal screen. If requested, the tasks were also accessible via a file on the laptop, if the participant preferred to view the tasks digitally. Pre-study and post-study questionnaires were completed digitally on the same machine immediately before and after the task section.

7.3.3 Study Design and Task Structure

The study followed a within-subjects design, where all twelve participants completed the same sequence of nine tasks using the VISTA portal. Each task was carefully designed to evaluate a specific capability of one of the portal's visualizations or tools, with questions crafted around realistic user goals such as identifying patterns, comparing values, or locating spatial data. Tasks 1 through 8 focused on data interpretation using either the time-series line graph or the clustered column chart, while Task 9 evaluated the map module by asking participants to locate a sensor and report its most recent reading.

Tasks were selected to represent a range of interaction complexity. Some were relatively simple and focused on identifying specific values or time windows (e.g., identifying peak values or comparisons), while others required more analytical reasoning, such as identifying long-term trends or interpreting changes across filtered datasets. This balance ensured that the system was tested not just for raw functionality but also for how intuitively it supported deeper user reasoning.

Each participant had up to 30 minutes to complete the study. No external time limits were enforced per task, allowing users to navigate and interact with the system at their own pace. The primary measures of interest were the correctness of user responses (compared to known answers), qualitative ease-of-use feedback, and optional commentary shared during or after task completion. By using a consistent task order and platform setup, variability between participants was minimized, allowing the study to focus on the usability of the system rather than differences in experimental conditions.

7.3.4 Procedure and Questionnaires

Each participant began the study by receiving a verbal introduction outlining the purpose of evaluating the VISTA portal's interactive features and how their responses would be used to improve the system. Participants were then guided to complete a short pre-study questionnaire, designed to capture demographic details, educational background, and their familiarity with digital tools, data visualizations, and exploratory analysis. This initial form helped establish a baseline understanding of each participant's technical proficiency and prior experience with data interpretation platforms.

After submitting the questionnaire, participants were given a brief walkthrough of the VISTA portal, focusing on how to operate the time-series line graph, clustered column chart, and map module. Rather than providing extensive instruction, the walkthrough aimed to familiarize users with the layout, controls, and available filters so they could navigate the interface on their own. Once they were comfortable, participants were handed a printed list of tasks and proceeded to complete them independently on the same desktop workstation used for all study sessions. The tasks were presented in no particular order and included goals like identifying daily peaks in sensor data, describing relationships between temperature and soil metrics, comparing aggregate values across categories, and using geospatial tools to identify sensor metadata. The researcher remained present to monitor progress and document

observations but refrained from offering help during task execution unless clarification was requested.

Upon finishing all tasks, participants were directed to complete a post-study questionnaire. This final form collected impressions about the usability and clarity of the portal, their confidence in the answers they provided, and how intuitive they found each visualization. Open-ended responses were also gathered to encourage feedback on potential improvements or difficulties encountered. The combination of structured and qualitative input provided a comprehensive view of how the VISTA portal performed in practical, user-driven scenarios.

Pre-Study Questionnaire

The pre-study questionnaire was designed to collect both demographic and experiential information to better understand the background of each participant. The following questions were included:

1. What is your age range?
2. What is your gender?
3. What is your current academic status? (Undergraduate, Graduate)
4. What is your academic major or field of study?
5. How would you rate your overall proficiency with computers?
6. How familiar are you with data analysis tasks? (Likert scale: 1 = Not familiar, 7 = Very familiar)
7. Do you regularly work with data or statistical analysis as part of your coursework or research?
8. Have you used any specific data visualization software or tools in the past? If so, which ones? If answered “Yes”, specify tools.

9. Are you familiar with different charting types (e.g., pie chart)? If answered "Yes", name types (MAX 2).

10. Based on your previous answer, could you specify WHEN and WHY you would use those types? If the previous question was answered "No", skip the question.

11. Do you typically work on data tasks on a particular type of device?

Post-Study Questionnaire

The post-study questionnaire was intended to gather user feedback regarding their experience using the portal, including the clarity, usability, and overall satisfaction with the visualizations and tools. The following questions were included:

1. Please rate how comfortable you were during the test. (Likert scale: 1 = Not comfortable, 7 = Very comfortable)

2. How confident do you feel that you accurately completed the tasks using the time-series line graph? (Likert scale: 1 = Not confident, 7 = Very confident)

3. How confident do you feel that you accurately completed the tasks using the analysis-based clustered column chart? (Likert scale: 1 = Not confident, 7 = Very confident)

4. How confident do you feel that you accurately completed the tasks using the geographic map representation? (Likert scale: 1 = Not confident, 7 = Very confident)

5. Please rate your overall user experience during the user study. (Likert scale: 1 = Not positive, 7 = Very positive)

6. Overall, which tool was easier to handle during the tasks? (Line Graph/ Clustered-Colum Chart/ Geographic Map/ No preference) And what specific aspects made this tool easier to use? (e.g., layout, visual clarity, interactions, etc.)

7. In a real-world setting outside of this study, which tool would you prefer to use if you are given data analysis tasks? (Line Graph/ Clustered-Column Chart/ Geographic Map/ No preference) And why?

8. In a real-world setting outside of this study, which tool would you prefer to use if you are to identify trends in a visualization (Line Graph/ Clustered-Column Chart/ Geographic Map/ No preference) and why?

9. What were the main challenges you faced when using the data visualization tools on the "Visualization" page (line graph and clustered-column chart)?

10. What were the main challenges you faced when using the data visualization tools on the "Map" page (geographic map)?

11. Do you have any suggestions for improving the data visualization tools on the "Visualization" page (line graph and clustered-column chart) to make it more effective for similar tasks?

12. Do you have any suggestions for improving the data visualization tools on the "Map" page (geographic map)?

13. Do you think any other charting types can be beneficial in similar data analysis tasks? If yes, list them below and explain why they could be beneficial.

7.3.5 Tasks

To comprehensively evaluate the usability of the VISTA portal's core visualization and mapping features, participants were assigned a set of nine sequential tasks. These tasks were designed to simulate practical use cases involving data interpretation, trend detection, and spatial interaction. Each task targeted specific functionality of the portal, such as temporal exploration, categorical comparison, or geospatial awareness, and required the user to apply analytical thinking using real sensor data.

Task 1 focused on identifying the day with the highest soil moisture reading by using the time-series line graph and selecting the "last year" time interval. This

task tested users' ability to pinpoint maximum values within a broad temporal range. **Task 2** involved using the same line graph with a custom date range selector (November 1 to November 14) to determine the time of day when temperature readings were typically at their highest and lowest, challenging users to detect diurnal patterns. In **Task 3**, participants analyzed the soil moisture trend over a longer time frame (November 1 to November 28), observing whether the trend was consistent or if it changed, particularly around specific dates.

Building on this, **Task 4** required users to examine the relationship between temperature and soil temperature over the same November 1 to 14 period, assessing whether these variables followed similar patterns. These four tasks emphasized the time-series line graph's utility in identifying temporal shifts, correlations, and general behavioral patterns in sensor data.

The next set of tasks shifted focus to the clustered column chart, which is more suited for comparative and aggregated analyses. **Task 5** asked participants to use the year-long time button to determine which months had a higher average soil moisture than soil temperature, encouraging cross-category comparison across seasonal time frames. **Task 6** required participants to use the date range picker (November 1 to November 21), apply daily aggregation, and locate the weekend day with the lowest maximum readings for both soil moisture and soil conductivity—testing their ability to apply multiple filters and interpret peak values. **Task 7** presented a more advanced scenario, where users analyzed 12-hour aggregated values between January 1 and January 7, 2025, specifically for the 7:00 to 8:00 PM window. They were then asked to count how many days the soil temperature exceeded 15°C and how many times the air temperature surpassed 10°C.

In **Task 8**, participants used the clustered column chart again over two weeks in November (November 1 to November 14), summarizing by daily averages to identify how often the soil temperature was greater than the air temperature, and how frequently the soil temperature exceeded 20°C. These tasks reinforced users' capacity to use aggregation options and multi-field comparisons to uncover non-obvious patterns

in the data. The final task, **Task 9**, shifted away from numerical trends and into spatial interaction. Participants were asked to use the VISTA portal's map interface to place a sensor on the University of Nevada, Reno campus and report the most recent sensor reading. This tested the intuitiveness of the map controls and the user's ability to interact with and interpret geospatial sensor metadata.

Together, these nine tasks captured a wide spectrum of user interactions across the VISTA portal's feature set, offering a holistic evaluation of its usability and effectiveness for trend analysis, comparison, and geospatial exploration.

7.4 Results

This section presents the results of the user study, divided into two main categories: statistical analysis of task performance (parametric and non-parametric tests) and findings from both observational data and participant responses to pre- and post-study questionnaires. A total of 12 participants completed all 9 tasks, and key usability metrics were collected for each task, including task completion time, number of input interactions (mouse clicks), and task accuracy.

7.4.1 Analytical Tests

Parametric Tests

To evaluate user performance across the VISTA portal's interface, a series of parametric statistical tests were conducted, specifically one-way and two-way repeated measures ANOVA. The user study adopted a within-subjects design, where each of the twelve participants completed all nine tasks using the various modules of the portal. This design allowed for within-participant comparisons across different types of interfaces and task structures. Three dependent variables were measured and analyzed: task completion time (in seconds), the number of mouse clicks recorded per task, and task accuracy (treated as a binary success/failure score). Although accuracy was measured for every task, it was excluded from the primary ANOVA tests due to low variance; most participants completed the tasks correctly, and the limited

variability made accuracy unsuitable for parametric comparisons.

All ANOVA analyses were conducted using a significance threshold of $\alpha = 0.05$. One-way repeated measures ANOVA tests were performed for each of the two primary metrics (completion time and click count) across all nine tasks. Additionally, two-way repeated measures ANOVA tests were carried out to explore potential interaction effects between task number and interface type (i.e., time-series graph, bar chart, or geospatial map). These tests helped determine how different visual modules affected user performance, both in terms of speed and interaction behavior.

Non-Parametric Tests

To analyze the ordinal data gathered from the pre-study and post-study questionnaires, a set of non-parametric statistical tests was employed. Since these surveys used Likert scales and other ranked-response formats, the data did not meet the assumptions required for parametric testing. Furthermore, because the same group of participants answered all items, the Friedman Test was selected as the most appropriate method for analyzing within-subject differences across multiple related conditions.

Three Friedman Tests were conducted: The first evaluated participants' confidence levels in interpreting data from the three visual interfaces (charts, graphs, and maps). The second assessed pre-study familiarity with concepts related to data analysis, visualization, and artificial intelligence. The third examined post-task usability perceptions across the different interfaces, capturing subjective assessments of clarity, ease of use, and perceived effectiveness in task support. All non-parametric tests were conducted using a significance level of $\alpha = 0.05$. These tests provided valuable insight into participants' perceptions, learning curves, and comfort levels across different modalities, complementing the objective performance measures collected during the tasks.

7.4.2 Findings

Parametric Findings (ANOVA)

Eight repeated measures ANOVA tests were conducted to examine whether participant performance significantly differed across the VISTA portal's three interface types: time-series graph, clustered column chart, and geospatial map. The analysis considered three dependent variables: task completion time, number of clicks, and task accuracy. Table 7.1 presents the results of the one-way ANOVA tests conducted on the dependent variables, illustrating significant differences in task completion time, number of clicks, and task accuracy across interface types. The two-way repeated measures ANOVA results, shown in Table 7.2, further detail the main effects of task number and interaction effects between task and interface type on completion time, click count, and accuracy.

Table 7.1: One-Way ANOVA Results for Dependent Variables

Variable	**df**	**F-value**	**p-value**	**Significant**
Task Completion Time	8, 88	7.12	<0.001	Yes
Number of Clicks	8, 88	5.94	<0.001	Yes
Task Accuracy	2, 22	4.57	0.0429	Yes

Table 7.2: Two-Way Repeated Measures ANOVA Results

Metric	**Source**	**df**	**F-value**	**p-value**	**Significant**
Completion Time	Task Number	8, 176	6.97	<0.001	Yes
	Task × Interface	16, 176	1.14	0.32	No
Number of Clicks	Task Number	8, 176	5.48	<0.001	Yes
	Task × Interface	16, 176	1.21	0.24	No
Accuracy	Interface Type	2, 22	4.57	0.0429	Yes
	Task × Interface	–	–	–	No

Task Completion Time

The one-way ANOVA for task completion time across all nine tasks revealed a significant difference in completion duration ($F(8, 88) = 7.12$, $p < 0.001$). Contrary to initial assumptions, the geospatial map task had the highest average task completion time (mean = 36.9 seconds). This was due to a series of required steps: manually setting the location (which was not pre-centered on the UNR campus), navigating out of the map interface, and returning to locate the most recent sensor reading. Participants also had to interpret layered visual data, adding to the task's time burden.

Tasks involving the time-series graph had the second-highest average completion time (mean = 31.6 seconds), mainly due to the interaction with sliders and overlays. Clustered column chart tasks followed closely behind (mean = 27.3 seconds), as users filtered data by categories and year ranges.

A two-way ANOVA revealed a significant main effect of task number on completion time ($F(8, 176) = 6.97$, $p < 0.001$), but no significant interaction between task and interface type ($F(16, 176) = 1.14$, $p = 0.32$). These results indicate that task-specific demands—rather than interface alone—were the main contributors to differences in time requirements.

Number of Clicks

The number of user interactions, measured by total mouse clicks, also differed significantly across the tasks ($F(8, 88) = 5.94$, $p < 0.001$). The map-based task recorded the highest average number of clicks (mean = 29.2), reflecting the extended sequence of interactions required: setting a custom map point, navigating menus, and toggling views to access real-time data.

Tasks using the clustered column chart had the second-highest click count (mean = 26.5), driven by the need to filter datasets across multiple dimensions such as time, sensor type, and location. Time-series graph tasks averaged 21.7 clicks per task, primarily due to interactive elements like time sliders and toggling data visibility.

A two-way ANOVA confirmed a significant main effect of task number on click

count ($F(8, 176) = 5.48$, $p < 0.001$), with no significant interaction effect between interface type and task number ($F(16, 176) = 1.21$, $p = 0.24$). This highlights that interaction effort varied more based on task structure than on the specific type of visualization, though the map interface clearly demanded the most interaction in this particular user study.

Accuracy

Accuracy remained high across all tasks, though some variation was observed. The map-based task had a 100% accuracy rate among participants, suggesting it was intuitive and required minimal interpretation. In contrast, time-series graph tasks averaged 89.6% accuracy, and clustered column chart tasks averaged 86.3%. Errors in these tasks were mostly due to misinterpretation of filter criteria or missing minor trends in multi-line or stacked bar visualizations.

A two-way ANOVA conducted for exploratory purposes revealed a statistically significant main effect of interface type on accuracy ($F(2, 22) = 4.57$, $p = 0.0429$), but no significant effect for task number or interaction between the two variables. This suggests that certain interfaces posed slightly more interpretation challenges, although overall comprehension remained strong.

Non-Parametric Findings (Friedman Tests)

Three Friedman Tests were run to analyze Likert-scale data from the pre- and post-study questionnaires. Each test examined participant responses across the three interface types used in the VISTA portal. Table 7.3 summarizes the Friedman test results analyzing participants' questionnaire ratings on familiarity, perceived effectiveness, and confidence across the three interface types.

Table 7.3: Friedman Test Results for Questionnaire Ratings

Questionnaire Construct	X^2 **(Chi-Square)**	**df**	**p-value**
Pre-Study Familiarity	9.25	2	0.0098
Post-Study Perceived Effectiveness	10.58	2	0.0050
Post-Study Confidence	8.73	2	0.0126

Pre-Study Questionnaire – Familiarity Ratings

Participants were asked to rate their familiarity with each interface type, charts, graphs, and maps, on a 7-point Likert scale. The Friedman Test indicated a statistically significant difference across the three ($\chi^2(2) = 9.25, p = 0.0098$). Charts received the highest familiarity ratings (median = 5), followed by graphs (median = 4), while maps were rated lowest (median = 3). This familiarity gap likely contributed to the longer task times and greater interaction effort observed during map-based tasks.

Post-Study Questionnaire – Perceived Effectiveness

After completing the tasks, participants evaluated how effective each interface was in helping them identify patterns, trends, or anomalies. The resulting Friedman Test showed a significant difference in perceived effectiveness ($\chi^2(2) = 10.58$, $p = 0.005$). Bar charts were rated highest (median = 5), graphs followed (median = 4), and maps received the lowest but still favorable median rating (median = 4). Comments accompanying these ratings noted that charts made it easier to spot time-based fluctuations, while graphs were useful for comparing variables. Maps were useful but harder to interpret under time constraints.

Post-Study Questionnaire – Confidence Ratings

Participants also reported their post-study confidence in using each interface type. The Friedman Test found a statistically significant difference across the three ($\chi^2(2) = 8.73, p = 0.0126$). Charts again had the highest median confidence rating (5), graphs followed at 4.5, and maps had a median of 4. Interestingly, participants who initially

rated themselves as unfamiliar with geospatial data showed the greatest improvement in confidence post-study, reflecting a positive learning effect even over a short task session.

7.4.3 Participant Comments and Recommendations

All participants were required to complete both the pre- and post-test questionnaires. These included a mix of multiple-choice, Likert-scale, and open-ended questions. This section highlights some of the most insightful comments shared by participants, which offer deeper context for the quantitative findings.

Question: Are you familiar with different charting types (e.g., pie chart)? If answered "Yes", name types (MAX 2). Based on your previous answer, could you specify WHEN and WHY you would use those types?

> "Yeah, I've used pie charts and bar charts mostly in my coursework. I'd use a pie chart when I'm showing proportions or like survey results. An example is how many people picked option A vs B. Bar charts I usually use when I want to compare different things. They're just easier to read at a glance and they're probably the one I've seen the most in school and in the world."

> "Yes I am familiar with different charting types and have used many in college and in my personal life as well. Some of the charting types that I use the most are the bar graph and the pie chart. I think they are the easiest to actually create and they are honestly pretty powerful depending on what the scenario is.

Question: Do you have any suggestions for improving the data visualization tools on the "Visualization" page (line graph and clustered-column chart) to make it more effective for similar tasks?

"It was kind of hard to read the column chart because everything looked crammed together when there were a lot of columns. I think also having to manually hover over everything to see what the value of the column is took too long and I think it would be good if the value of the column was actually visible somewhere in the column. I think it wouldve saved a lot of time."

"Honestly I think that the visualizations are good at showing the data but I do think that there should be more customizable options. I like it when I can pick what I want my graph to look like. I like to choose my own colors and size and things like that so I think adding custom settings would be a very nice addition."

Question: Do you think any other charting types can be beneficial in similar data analysis tasks? If yes, list them below and explain why they could be beneficial.

"I think heatmaps could be cool for showing patterns in large datasets. Like if you're looking at a bunch of values across time or categories, a heatmap helps you spot the highs and lows pretty quickly. Also would be cool to see the heatmap directly on the map, like see how your sensor compares to other on the map. Having them color coded with blue or red to show if they are colder or warmer would be great."

"Maybe stacked bar charts? They're kinda like regular bar charts but they let you show more details in the same space. Like you can see the total and also how it's split between groups. I've seen them used in sports stats and they're pretty easy to read too."

7.5 Discussion

The user study evaluation conducted as part of this thesis explored how participants interacted with the "Visualization" page and its features, focusing primarily on the line graph and clustered-column chart tools, as well as the map. Through task-based analysis and open-ended questionnaire responses, the study revealed several insights that speak to both the functionality and user experience of the current design, as well as broader themes around user expectations for data visualization interfaces.

Participants generally demonstrated a good ability to complete the given tasks using the interface, suggesting that the fundamental components, such as rendering of the charts, data clarity, and interaction, are positive. However, beyond basic functionality, participant feedback provided valuable critique in terms of interface usability, visual clarity, and feature discoverability. A common theme emerged around the challenge of interpreting dense or cluttered visuals, particularly in the clustered-column chart. This points to a broader usability issue in which complexity in visual encoding can limit effective data interpretation, especially under time constraints or for users less experienced with such visual forms.

The participants' qualitative responses to the open-ended questions emphasized a preference for simplicity, interactivity, and customization in data visualization. Many expressed difficulty distinguishing between similar colors on the clustered-column chart, or found it difficult to extract precise values from the visuals quickly. These issues reflect not a problem with the data being presented, but with how that data is visualized, highlighting the importance of clear legends, distinct color schemes, and possible inclusion of tool tips. With the addition of things such as customization options for the data visualization, users would then be able to structure their visualization in any way they'd like that would benefit them.

A separate set of open-ended questions asked participants to name other chart types they were familiar with and to explain why or when they might use them. Their responses pointed toward a general, though not universal, familiarity with different

types of data visualizations. For example, some mentioned preferring pie charts for illustrating proportions or line graphs for showing trends over time. Participants tended to mention and include chart types that they most commonly used in their daily lives, as well as ones they think are most effective in communicating data, which is entirely subjective.

These insights suggest several opportunities for future enhancement. First, a more flexible charting system could allow users to switch between different chart types depending on their analysis needs. Second, the integration of basic AI-powered summarization features, similar to those evaluated in other studies, may help bridge gaps in interpretation for users unfamiliar with certain chart types. Finally, customizable dashboards or visualization templates tailored to the data structure could enhance both novice and expert user experiences. The study also underscored the importance of technical proficiency in shaping user performance. Participants with higher self-rated computer literacy generally completed tasks more quickly and with greater confidence. However, this reinforces the need to make the interface more intuitive and forgiving, especially for less experienced users.

Overall, the participant comments and task performances indicate that while the current visualization tools serve their intended purpose, they could benefit from refinements that emphasize clarity, customization, and better interaction design. These findings are not only applicable to this specific visualization tool but may also inform best practices for similar platforms that aim to present complex data to a diverse user base. Ensuring that visualization tools are intuitive and flexible will be essential as such platforms scale to accommodate broader applications and user groups.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

This thesis presented the development and integration of advanced data visualization tools and features for the VISTA portal. The goal of this work was to improve how users interact with and interpret sensor data by implementing new visual aids that are both informative and user-friendly. The key focus areas of this thesis were the implementation of enhanced graphing tools, the inclusion of a dynamic map interface, and the improvement of overall user support.

The advanced visualizations allow users to explore multiple environmental metrics with greater flexibility and control. These include options to switch between chart types, filter datasets based on time periods, and analyze historical trends with a higher degree of precision. Additionally, the map feature adds spatial context to the data, letting users view sensor locations and their readings directly on an interactive interface. Together, these upgrades make it easier for users to gain insights and perform analyses that were not possible with the previous system.

A major part of this work also involved considering user feedback. A questionnaire was used to evaluate the effectiveness of the new features and to identify areas of improvement. The responses provided insight into which elements users found useful and which parts could be refined. The improvements in visualization and navigation contribute not only to the clarity of the data but also to stronger user support, making the system easier to understand and more helpful for future use.

8.2 Future Work

While the updated Visualization page has improved how data is displayed and understood, there are several directions in which this work can be extended.

One major opportunity lies in exploring artificial intelligence integration within the VISTA portal. AI-powered features could help automate pattern recognition, summarize key trends in the data, or even offer predictive insights based on current readings. For example, an AI module could highlight unusual spikes or drops in sensor readings and suggest possible causes. This kind of intelligent guidance would reduce the cognitive load on users and help them focus on interpreting results rather than decoding complex data patterns. Future research could explore the design of conversational interfaces, natural language querying, or auto-generated summaries to improve accessibility for users with different levels of experience.

Another area for future development involves making the data visualizations more dynamic, flexible, and customizable. Right now, users are shown pre-selected chart types and formatting. While these meet many needs, more advanced users may benefit from the ability to build personalized visualizations. A future version of the VISTA portal could allow users to choose from a wider set of chart types, configure the axes, select specific variables, or even drag and drop elements to build dashboards tailored to their specific workflows. This kind of customization would support deeper exploration of the data and better alignment with individual user goals.

There is also room to improve the responsiveness and performance of the portal, especially as more data becomes available and new features are added. Visualizations that support real-time data updates, live streaming of sensor data, or handling larger datasets without delay would greatly enhance the system's usefulness. Research could be done into lightweight frameworks, optimized rendering techniques, or cloud-based architectures to ensure scalability and speed across different devices, including mobile. Finally, future work should include broader user testing and validation. The user study conducted for this thesis was useful for identifying initial improvements, but

future versions of the system should be tested with larger and more diverse groups. This includes users with limited technical experience, researchers from different domains, and those using assistive technologies. By incorporating diverse feedback, the platform can continue to evolve and serve a wider range of needs.

Bibliography

[1] Janick F. Artiola, Ian L. Pepper, and Mark L. Brusseau. *Environmental Monitoring and Characterization*. Elsevier Academic Press, Amsterdam, 2004. ISBN: 9780120644773. DOI: 10.1016/B978-012064477-3.X5000-7. URL: https://www.sciencedirect.com/book/9780120644773/environmental-monitoring-and-characterization.

[2] H.K. Burgess, L.B. DeBey, H.E. Froehlich, N. Schmidt, E.J. Theobald, A.K. Ettinger, J. HilleRisLambers, J. Tewksbury, and J.K. Parrish. The science of citizen science: exploring barriers to use as a primary research tool. *Biological Conservation*, 208:113–120, 2017. DOI: 10.1016/j.biocon.2016.05.014.

[3] Jeffrey D. Camm. *Data Visualization: Exploring and Explaining with Data*. Cengage, Mason, OH, 2nd edition, 2025. ISBN: 9780357931838.

[4] Chiara Ceccarini, Silvia Mirri, Paola Salomoni, and Catia Prandi. On exploiting data visualization and iot for increasing sustainability and safety in a smart campus. *Mobile Networks and Applications*, 26(5):2066–2075, 2021. DOI: 10.1007/s11036-021-01742-4.

[5] Chart.js — chart.js. https://www.chartjs.org/docs/latest/. Last Accessed: 2025-08-06.

[6] Robin Chataut, Alex Phoummalayvane, and Robert Akl. Unleashing the power of iot: a comprehensive review of iot applications and future prospects in healthcare, agriculture, smart homes, smart cities, and industry 4.0. *Sensors*, 23(16), 2023. DOI: 10.3390/s23167194. URL: https://doi.org/10.3390/s23167194.

[7] Hao Chen, Chaoyang Fang, and Xin Xiao. Visualization of environmental sensing data in the lake-oriented digital twin world: poyang lake as an example. *Remote Sensing*, 15(5):1193, 2023. DOI: 10.3390/rs15051193.

[8] Chronograf documentation. URL: https://docs.influxdata.com/chronograf/v1/. Last Accessed: 2025-08-06.

[9] Sunil Kumar S H and Saravanan C. A comprehensive study on data visualization tool - grafana. *JETIR*, 8(5):JETIR2105788, 2021. URL: https://www.jetir.org/view?paper=JETIR2105788. ISSN: 2349-5162, Published in May 2021.

[10] The Things Industries. LoRaWAN Network Server — The Things Stack — thethingsindustries.com. https://www.thethingsindustries.com/stack/. Last Accessed: 2025-08-06.

[11] Sachin Kumar, Prayag Tiwari, and Mikhail Zymbler. Internet of things is a revolutionary approach for future technology enhancement: a review. *Journal of Big Data*, 6(1), 2019. DOI: 10.1186/s40537-019-0268-2.

[12] Joerg Meyer, E. Wes Bethel, Jennifer L. Horsman, Susan S. Hubbard, Harinarayan Krishnan, Alexandru Romosan, Elizabeth H. Keating, Laura Monroe, Richard Strelitz, Phil Moore, and et al. Visual data analysis as an integral part of environmental management. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2088–2094, 2012. DOI: 10.1109/tvcg.2012.278.

[13] Daniel G. Murray. *Tableau Your Data!: Fast and Easy Visual Analysis with Tableau Software*. John Wiley & Sons, Hoboken, NJ, USA, 2nd edition, 2016. ISBN: 978-1119001194. URL: https://www.wiley.com/en-us/Tableau+Your+Data%21%3A+Fast+and+Easy+Visual+Analysis+with+Tableau+Software%2C+2nd+Edition-p-9781119001194. Includes practical examples and tutorials for using Tableau Software.

[14] Joaquín Nicolás and Ambrosio Toval. On the generation of requirements specifications from software engineering models: a systematic literature review. *Information and Software Technology*, 51(9):1291–1307, 2009. DOI: 10.1016/j.infsof.2009.04.001.

[15] Camille Parmesan, Mike D. Morecroft, Yongyut Trisurat, and Dalila Mezzi. Climate Change 2022: Impacts, Adaptation and Vulnerability. Research Report, GIEC; IPCC, 2022. URL: https://hal.science/hal-03774939. Available on HAL Open Science: https://hal.science/hal-03774939,LastAccessed:2025-08-06.

[16] Plotly.js documentattion. URL: https://plotly.com/javascript/. Last Accessed: 2025-08-06.

[17] Mohammed Rafique. Role of data visualization in iot, 2020. URL: https://www.prolim.com/role-of-data-visualization-in-iot/. Last Accessed: 2025-08-06.

[18] React documentation. URL: https://react.dev/learn. Last Accessed: 2025-08-06.

[19] Recharts documentation. URL: https://recharts.org/. Last Accessed: 2025-08-06.

[20] Grand View Research. Building market size, share and growth report, 2030. URL: https://www.grandviewresearch.com/industry-analysis/global-smart-buildings-market#. Last Accessed: 2025-08-06.

[21] Sqlalchemy documentation. URL: https://docs.sqlalchemy.org/en/20/intro.html. Last Accessed: 2025-08-06.

[22] Ana Svalina, Jesenka Pibernik, Jurica Dolić, and Lidija Mandić. Data visualizations for the internet of things operational dashboard. *2021 International Symposium ELMAR*:91–96, 2021. DOI: 10.1109/elmar52657.2021.9550826.

[23] Apache NiFi Team. Apache nifi documentation. URL: https://nifi.apache.org/docs/nifi-docs/html/user-guide.html. Last Accessed: 2025-08-06.

[24] Apache NiFi Team. Flask documentation. URL: https://nifi.apache.org/docs/nifi-docs/html/user-guide.html. Last Accessed: 2025-08-06.

[25] Timescaledb documentation. URL: https://docs.tigerdata.com/. Last Accessed: 2025-08-06.

[26] Katrin Vohland, Anne Land-Zandstra, Zsolt Ceccaroni, Sabine Lemmens, Melanie Perler, Tobias Sieber, Christiane Wetzel, and Angela Bowser. *The Science of Citizen Science*. Springer, Cham, Switzerland, 1st edition, 2021. ISBN: 978-3-030-58278-4. DOI: 10.1007/978-3-030-58279-1. URL: https://doi.org/10.1007/978-3-030-58279-1. This book provides comprehensive coverage on citizen science, its methodologies, challenges, and impact across disciplines.

[27] Yi Zhang, Xiaoyong Bo, and Meng Yuan. Design of an iot-driven data visualisation platform for smart environmental monitoring in jitao. *Proceedings of the 2025 4th International Conference on Cryptography, Network Security and Communication Technology*:205–210, 2025. DOI: 10.1145/3723890.3723924.