HELLENIC MEDITERRANEAN UNIVERSITY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

THESIS

DEVELOPMENT OF BIG DATA PROJECTIONS WITH MAPLIBRE AND D3.JS TECHNOLOGIES ADAPTED TO A REACT ENVIRONMENT

GIANNIS GEROLYMOS

SUPERVISOR
PROFESSOR ATHANASIOS G. MALAMOS

HERAKLION
SEPTEMBER 2023

Abstract

The rapid proliferation of interconnected devices within the Internet of Things (IoT) ecosystem has led to an exponential surge in data generation and consumption. This expanding phenomenon has highlighted the importance of efficient management and effective representation of big data. A prominent contributor to this surge of big data is sensor networks which generate geotagged data covering a wide range of environmental and electromechanical conditions. This thesis aims to handle this expanding data landscape by developing an application that incorporates powerful cartographic and graphical features. It was deemed necessary for the existence of two distinct visualization technologies that would be harmoniously integrated under one framework. In view of this need, research was conducted to select the optimized technologies. Subsequently, an in-depth architectural analysis of the application will be carried out with the purpose of capturing the system's fundamental architectural qualities. This analysis will demonstrate how these architectural components precisely interconnect and augment the application's functional requirements. Concluding this thesis, the application's implementation will be meticulously carried out, offering a detailed roadmap for seamless operations throughout its interface. To elaborate further, distinct iterations of the user's navigational experiences will be methodically analyzed, facilitating the exploration of the system's features intuitively.

Σύνοψη

Η ταχεία εξάπλωση των διασυνδεδεμένων συσκευών στο πλαίσιο του Διαδικτύου των Πραγμάτων (IoT) έχει οδηγήσει στη ραγδαία αύξηση της παραγωγής και της κατανάλωσης δεδομένων. Αυτό το εκτεταμένο φαινόμενο έχει αναδείξει τη σπουδαιότητα της αποδοτικής διαχείρισης και της ορθής αναπαράστασης των μεγάλων δεδομένων. Ένας εξέχων συντελεστής αυτής της έξαρσης μεγάλων δεδομένων είναι τα δίκτυα αισθητήρων που παράγουν δεδομένα με γεωγραφικές ετικέτες που καλύπτουν ένα ευρύ φάσμα περιβαλλοντικών και ηλεκτρομηχανικών συνθηκών. Η παρούσα διατριβή αποσκοπεί στη διαχείριση αυτών των μεγάλων δεδομένων αναπτύσσοντας μια εφαρμογή που ενσωματώνει ισχυρά χαρτογραφικά και γραφικά χαρακτηριστικά. Κρίθηκε απαραίτητη η ύπαρξη δύο διαφορετικών τεχνολογιών απεικόνισης που θα ενσωματώνονται αρμονικά κάτω από ένα πλαίσιο (framework). Λαμβάνοντας υπόψη αυτή την ανάγκη, διεξάχθηκε έρευνα για την εύρεση των βέλτιστων τεχνολογιών. Ακολούθως, πραγματοποιήθηκε μια εις βάθος αρχιτεκτονική ανάλυση της εφαρμογής με σκοπό την αποτύπωση των βασικών αρχιτεκτονικών ιδιοτήτων του συστήματος. Η ανάλυση αυτή θα αναδείξει τον τρόπο με τον οποίο αυτά τα αρχιτεκτονικά στοιχεία διασυνδέονται επακριβώς και επαυξάνουν τις λειτουργικές απαιτήσεις της εφαρμογής. Ολοκληρώνοντας την παρούσα διατριβή, θα πραγματοποιηθεί επιμελώς η παρουσίαση της υλοποίησης της εφαρμογής, προσφέροντας έναν λεπτομερή οδοιπορικό χάρτη για όλες τις διεπαφές της. Πιο συγκεκριμένα, θα αναλυθούν μεθοδικά όλες οι διακεκριμένες πλοηγήσεις του χρήστη, διευκρινίζοντας ενορατικά την εξερεύνηση των χαρακτηριστικών του συστήματος.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my sincere gratitude to my parents, partner, family, and friends for their unwavering psychological support over the years, particularly during the process of writing this thesis.

I am deeply grateful to my supervisor Professor Athanasios G. Malamos, Professor at the Department of Electrical and Computer Engineering of the Hellenic Mediterranean University, for the trust he showed me in assigning me this thesis and for the opportunity to be part of his team in the Multimedia laboratory. I would also like to thank my co-supervisor Mr. Markos Fragkopoulos for guiding me through the whole coding and writing process of this thesis. Finally, my gratitude extends to the rest of the Lab members for any guidance that I received and for fostering a positive and collaborative atmosphere within the laboratory.

1 Introduction

It is a privilege to live in the information era also known as the Information Age, the Computer Age, the Digital Age, or New Media Age which brought several important inventions and innovations and led to an economy centered on information technology. This era established the use of computers to produce, store, and exchange abundant data. The emergence of the Internet and the constant need for copious newly generated data daily which individuals, organizations, and societies rely on for their growth and development led to the creation of big data.

Big data is a massive collection of data that has a more varied and complex structure with the difficulties of storing, analyzing, and visualizing for further processes or results [1]. The technique of analyzing enormous amounts of data to uncover obscure patterns and hidden correlations is named big data analytics [1]. Big data is also known as the three Vs of big data [2].

i. **Volume:** This is maybe the most expected characteristic of big data. Big data is about volume, as it reaches unprecedented heights, the likes of which cannot be predicted [2].
ii. **Velocity:** This refers to big data rapid production. Big amounts of data are constantly generated, and time is of the essence. Companies make the most of that speed to either generate or process that streaming data to meet their demands and to avoid creating a bottleneck [2].
iii. **Variety:** This is referring to the nature of data. Big data can be generated with heterogeneous sources, either by humans or by machines. There are at least 3 types of big data that can be categorized [2]:
    a. **Structured:** Structured data requires a data model and a data repository, which is typically a database. Its main characteristic is that it is classified as quantitative data, which also includes a precise and strict format. It is highly organized and easily decipherable by machines which makes this type of foundation preferable in machine learning and artificial intelligence. SQL (Structured Query Language) is the most common language to manage structured data in databases [2].
    b. **Unstructured:** Unstructured data is unorganized information that mainly leads to a foundation that evades a predetermined data model or schema. Furthermore, it consists of different types of formats that can be generated by either humans2

such as emails (message fields) and text files (Word processing, spreadsheets), or by machines such as scientific data (atmospheric data) and digital surveillance (surveillance video or photos). Non-relational databases are often the most common choice for storing unstructured data [2].

c. **Semi-Structured:** As the name reveals, semi-structured data is a combination of the above categories. Although it cannot be stored in a relational database, it has some organizing properties, tags, and semantic markers, thus allowing the separation of data elements. All in all, it is a more flexible foundation that evades a fixed schema and contains some structural elements that enable information grouping and hierarchies [2].

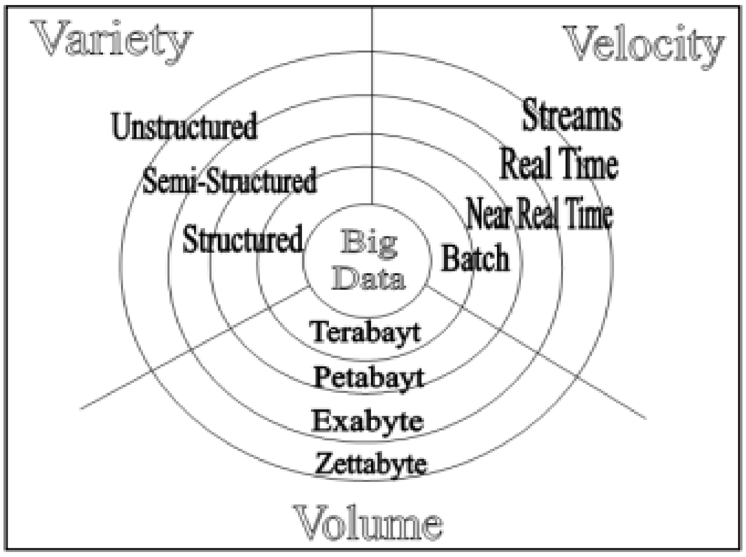


Figure 1. The three Vs of big data

Big data has been integrated into society to such a degree that it can now appear in various sectors with a wide range of applications [3]. Listed below are three of the main sectors:

i. **Healthcare:** The analysis of enormous volumes of medical records, clinical trial data, and research data is largely attained through big data, having a tremendous impact on the healthcare industry. Healthcare professionals utilize big data analytics to seek out patterns, trends, and correlations in patient data, resulting in far more accurate diagnoses, individualized therapies, and enhanced results for patients. Big data analytics is additionally essential for overseeing population health, monitoring diseases, and maximizing the implementation of healthcare assets [3].

ii. **Banking and Security:** Big data has evolved the banking and security sector, as the accumulation of all this massive information held by banks for so many years could now be harnessed using big data analytics. The utilization of these historical and real-time data, having regard to the fact that the accounting framework is verifiable, offered several advantages. In particular, they can identify patterns of human behavior and transactions to detect and prevent fraud, develop depth in customer relationships, and enhance decision-making regarding investments, lending, and overall risk exposure [3].

iii. **Agriculture:** The agricultural sector has an important role to play in contributing to the global supply chain and economic development. With the existence of big data and the tools to analyze them, data is in digital form, providing farmers the opportunity to derive value from it, improving their productivity. More specifically, big data technologies empower farmers with weather forecasts and weather data in general, real-time decisions and alerts, optimized seeds and livestock, and methodologies that will overall facilitate the forecasting of yields. As a result, products are transferred from production to consumption with precision, consistency, and rapid delivery [4].

As mentioned above, big data emerged to respond to the needs of modern society. Currently, there are a lot of organizations and companies that use and even depend functionally on big data. Two of the most famous organizations are Google and McDonald's [5].

In this field, Google should be mentioned first, as it is considered one of the largest innovators and implementers of big data technologies. A few of them that should be mentioned are Google Maps, Google Analytics, Machine Learning, Search Engine, and Advertising [6]. Google Maps uses big data to determine the best routes, compute real-time traffic and propose close-by businesses and activities. Google Analytics is a web analytics service that utilizes big data to provide website owners the opportunity to track user behavior and thus deduce their site's performance. Google also operates in machine learning in a vast range of applications, from voice recognition and image classification to language translation and spam filtering. This company also has an advertising platform nominated as Google Ads, which also specializes in observing user behavior conducive to the best possible advertising. Lastly, arguably their best technology is their search engine. Google manufactured a search engine that thinks and responds like a human. The company collects and analyzes vast amounts of data from user behavior, websites, and web pages to calculate the best relevant search results for each query.

Another approach to big data utilization is through McDonald’s. McDonald's is a global fast-food chain that relies heavily on big data to maintain its growth and territorial expansion [7]. Their goal is to deliver quality fast food at a reasonable price, in a convenient location, and in a handy way; to illustrate this point of view:

i. **McDonald’s Menu Optimization**: McDonald's uses big data analytics to determine customer preferences and purchasing patterns to optimize its menu. They track sales data, customer reviews, and demands to discover the value of all menu items and therefore prioritize interests on their best sales, adjust the number of ingredients they need to order, and discover if an item needs improvement or removal.

ii. **Drive-Thru Experience:** McDonald's drive-thru method is a very successful strategy to serve millions of customers globally daily. They use big data to optimize the architecture of the drive-thru, estimate traffic, and calculate the average waiting time of each customer. Consequently, they staff their employees appropriately to avoid traffic congestion even during rush hours.

iii. **Mobile App**: McDonald's mobile app allows customers to place orders, pay for food, and earn rewards. The app urges its users to create an account, rewarding them with special deals and ease of user interaction. McDonald’s benefits from its application by collecting data on customer behavior, including order history and location data, to offer personalized suggestions and promotions, thus guaranteeing positive customer experience and marketing optimization.

Depending on the sector and organization, big data can include information gathered from a variety of internal and external sources, including transactions, social media, enterprise content, sensors, and mobile devices. Among the above sources, it is worth highlighting the contribution of sensors for their efficient and massive production of real-time data. In essence, the purpose of a sensor is to operate as a device that senses and captures various types of data, whether they are related to environmental conditions (e.g., chemical, temperature, humidity, etc.), biological, physical, or mechanical [8]. Although sensors are capable of generating and detecting big data, the current needs regarding the production, acquisition, and analysis of this data, especially in real-time, require the presence of a very important technology, the Internet of Things.

The term Internet of Things is a cluster of infrastructures linking interconnected objects enabling procession, data analysis, and access to the output data. These connected objects could be sensors and/or actuators programmed with a particular functionality and designed to interact with other devices [9]. The IoT has expanded in multiple fields, some of the most prominent are health care, agriculture, resource management, energy, and asset tracking [9], [10]. These fields are accompanied by impressive technologies such as RFID, Wireless Sensor Network (WSN), and Smart Objects that consequently provide new developments, technological achievements, and opportunities [9]. Regardless of the numerous fields of application for IoT and their accompanied technologies, the use of IoT devices i.e., connected objects, lean towards the same kind of architecture: Data required to be moved, stored, processed, and then made accessible. Considering the data field, the Internet of Things generates massive quantities of data every second (Big Data) [11].

Gathering such massive data integrates the need to visualize it. Big data visualization is the procedure of transforming large and complicated data visually, to fathom and gain insights into that information. It is a matter of high significance as it enables organizations to convert raw data quickly into essential information that will eventually improve their business acumen[12]. Each organization can use a variety of methods to enhance this practice as there are currently several data visualization tools and techniques. Some of the most common techniques are plots, maps (heat maps, tree maps), and charts, e.g., line charts, bar charts, pie charts, etc. Therefore, emphasizing the importance of visualizing a data set should provide additional benefits. Some of them are:

i. **Detect errors and estimate risk:** It is always in the brand's or company's best interest to identify threats and mistakes that will eventually jeopardize their security and longevity. Analyzing those threats will provide a scale of safety, delineating the risk measures.
ii. **Real-time Monitoring:** The ability to constantly observe your system's performance or a part of it yields a cutting-edge opportunity. From Network security to employee productivity, real-time monitoring will improve incident response time, whether a system breach needs to be resolved quickly, down to an update in a team project. It can also have an essential role in health care as continually monitoring a patient via a wearable or injectable body sensor can save his life.
iii. **Perceive patterns and form a prognostication**: This feature is integral to any business industry. Big data visualization improves the comprehension of data as it

simplifies complex data into visual information that allows its analyzers to observe patterns and forge predictions. Additionally, they are enabled to develop a prospective strategy that will optimize their entire business/brand structure.

1.1 Scope and Objective

The existence of big data has created new challenges and new opportunities. By focusing on the stages of production, collection, analysis, and visualization of big data, the dual nature of this phenomenon is more clearly illustrated. Starting from the production stage, big data, depending also on the industry, is formed from different sources such as sensors (especially with the presence of IoT), emails, documents, etc. Integrating all these sources into a final data set is no easy task. Moving on to the analysis and visualization stage, many tools and techniques are available, but finding the optimal one to meet the necessary needs renders this process time-consuming and expensive. Finally, an implicit challenge is the requirements that thrive at the initiation of Big Data projects: an application that incorporates interoperability, adaptability, interactivity, and real-time capabilities.

Taking into consideration the aforementioned necessities we present as a solution a web-based framework, powered by React and React Native, designed to revolutionize the traditional methods of agriculture. Through IoT technology and the integration of sensors, our application offers farmers distant surveillance of their fields, real-time data observations, and innovative analytics, moving toward the new era of smart agriculture.

In this thesis, the subject of big data visualization will be explored, which is implemented by the technologies of MapLibre and D3.js adapted to a React environment. Moreover, the farmer will be granted data projections with advanced capabilities in both map and graphical formats. In brief, the farmer has the opportunity to define his fields and crops, to monitor and interact with them visualized on the map of the application, which was implemented with the MapLibre library. At the same time, projections of interactive graph visualizations will be developed, using the D3.js library.

1.2 Thesis Overview

This thesis is structured in 5 chapters as follows:

Chapter 1: The introduction chapter provides the context and establishes the scope and objectives of this thesis. Furthermore, it gives an overview of the thesis structure, summarizing each chapter.

Chapter 2: The background chapter conducts research on each implemented technology, showcasing every aspect of its features in comparison with similar technologies, to validate it as the optimal choice.

Chapter 3: The application architecture acts as the mediator between the requirements and the coding process. Specifically, this chapter unveils user interactions, introduces blueprints of key components, and establishes both functional and non-functional requirements in detail.

Chapter 4: The implementation chapter acts as a user manual of the application. It has been developed to document and illustrate every possible navigation option available to the user that aims towards the application's visualization technologies.

Chapter 5: The conclusions chapter marks the final chapter of this thesis. This section evaluates the extraction of substantiated conclusions through rigorous analysis resulting from the theoretical as well as the practical processes. Additionally, it culminates with suggestions for future development and potential integrations.

2 Background

Software development is a practice that constantly has new technological features, achievements, and ambitions. Its growth is keener now more than ever and this is evident from the coining of the term global software development, where distributed teams, consisting of individuals from all kinds of backgrounds and origins collaborate and deliver high-quality software solutions to a global audience [13]. In this context, a considerable share of the evolution of software development is due to the contribution of open-source software, culminating in Linux and Apache where thousands of participants from all over the world contributed long after its initial release [14]. As a result, a variety of programming languages are released almost every month, incorporating new versions and frameworks. This plethora of methods and available tools offers freedom of choice but also complicates the search for the optimal strategy [15].

The project at hand addresses the area of big data visualization, which involves processing and presenting massive volumes of data in a relevant and interactive manner. This profound initiative demands integration and coexistence with several cutting-edge technologies. In pursuit of these requirements, it is necessary to establish a thoughtfully crafted approach. This necessitates meticulously choosing the frameworks, map libraries, and data visualization tools that will serve as the foundation of our implementation.

2.1 Web Development

The World Wide Web, frequently referred to as the Web, is a techno-social system developed to improve human cognition, communication, and cooperation through technology networks [16]. Although the terms "web" and "internet" are not interchangeable, the web is by far the most visible component of the internet and has completely changed how people communicate in the modern world. The philosophy underlying the web as a techno-social system is its ability to serve as a platform enabling far more than technological connectivity. On the contrary, it functions as a means of transforming humankind's intellect by facilitating greater access to vast amounts of information and knowledge. Users possess the ability to investigate, educate, and interact with a broad range of subjects and concepts on the web, drastically improving their intellectual abilities. The web additionally renders it feasible for individuals from across the world to communicate, exchange ideas, and collaborate on a variety of projects and initiatives. There are four generations in the development of the web, each with its features and functions [16].

Web 1.0: Web 1.0, the first phase of web development, was created as a read-only platform for businesses to share data with users, with just a small amount of interactivity. The core protocols of Web 1.0 were HTTP, HTML, and URI. The majority of the content was static, however, there were hyperlinks to access it and bookmarking to store the links. HTML forms were used to send emails. The main purpose was to exchange content online with limited user engagement or developing content throughout the initial interaction. The website's content was kept in files on the server, from which it was fetched and shown to the client when a request was made. To prevent the website from loading slowly, the visitor comments were added to a standard guestbook page rather than the straight content[16], [17].

Web 2.0: Significant technological advances defined the growth of web development from the static Web 1.0 to the dynamic Web 2.0. The introduction of databases, upgraded servers, and elevated connection speeds permitted a shift from passive data consumption to active user participation. Users became content creators with Web 2.0, adding live material through comments, tweets, videos, and posts. Furthermore, the integration of Application Programming Interfaces (APIs) allowed for the effortless import of data from one website into another. This functionality broadened web developers' options, letting them construct interlinked and integrated web experiences. All in all, the second generation, evolved into a read-write web, shifting the focus on users to create content, engage in social activities and establish online communities [16], [17].

Web 3.0: Following the arrival of Web 3.0, sometimes referred to as the semantic web or the intelligent web, features an abundance of revolutionary technologies and development processes that have fundamentally altered the way websites are built. Semantic web technologies, which allow machines to grasp and process information in a more meaningful way, are at the center of this generation. The incorporation of distributed databases alongside distributed computing facilitated the effective management and analysis of vast quantities of data in a variety of locations. Microformats serve as a key for coordinating and assembling data, thereby rendering it easily available and practical for humans as well as computers. To derive substantial knowledge from data, automate procedures, as well as improve decision-making processes, natural language processing (NLP), data mining, and machine learning have been pivotal. In addition, the emergence of artificial intelligence (AI) technologies has entirely altered the user experience. These sophisticated systems are highly adapted to comprehend user habits and patterns, thus providing personalized and relevant suggestions. To brief, the intention was to offer machine-readable material, thereby minimizing human duties and decisions and facilitating efficient human-machine cooperation [16], [17].

Web 4.0: The symbiotic web, also known as Web 4.0, is envisioned as an intelligent and dynamic platform where computers and human minds coexist in perfect harmony. The web has developed throughout these decades into a potent instrument that continuously alters how we see, access, and exchange information online [16].

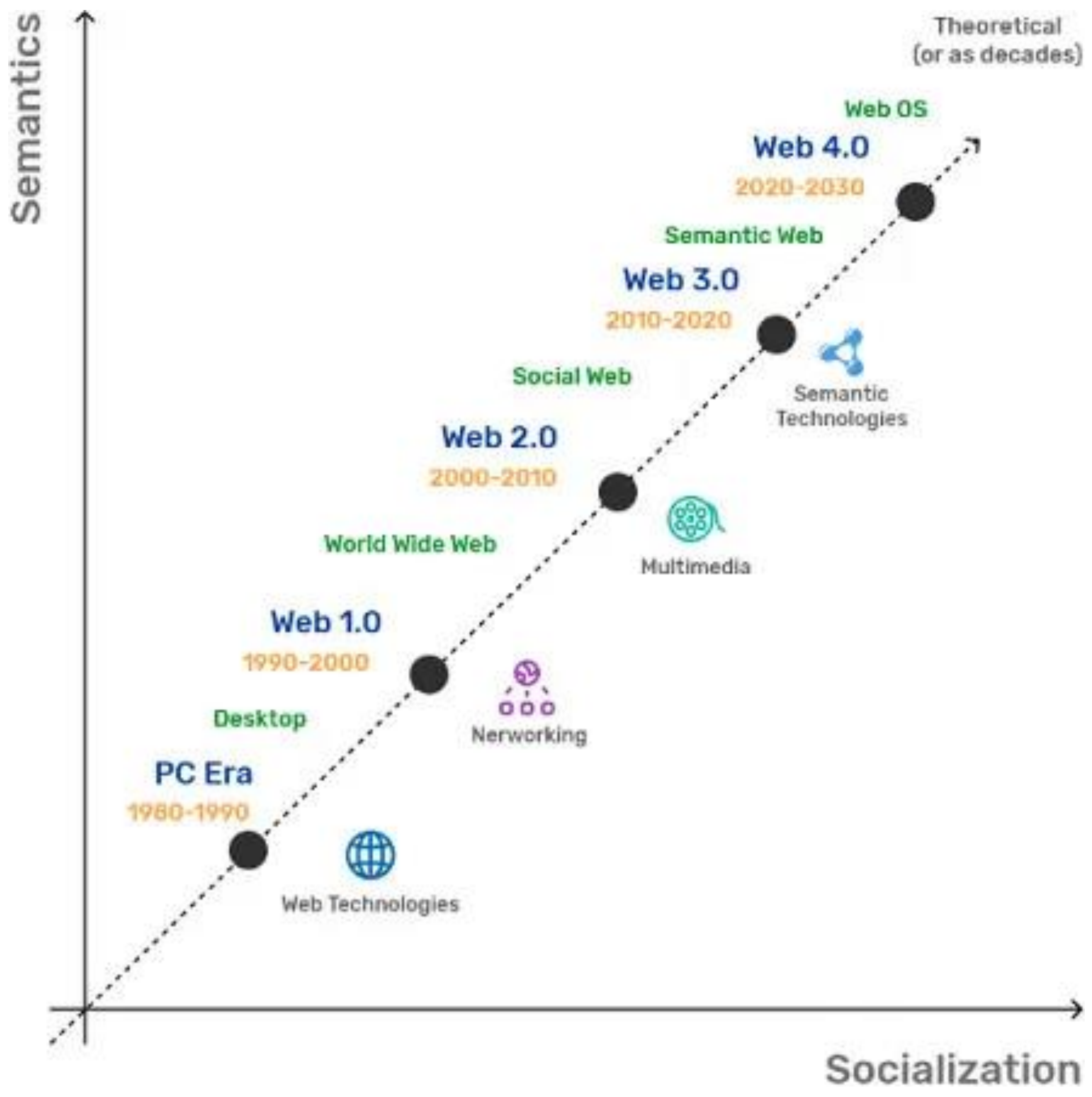


Figure 2. Web Transition over the years

2.2 JavaScript

As indicated web development has made substantial progress over the last decades, enabling the development of dynamic and interactive web applications. Undoubtedly, the most mesmerizing transition of web generations was the transition from the static Web 1.0 to the interactive Web 2.0. JavaScript (JS) turned out to be a key programming language even though it wasn't initially designed for that impact. In the early stages, approximately in 1995, when Netscape introduced both Java and JavaScript, the initial plan was for Java to be the primary language, grasping the role of constructing sophisticated, interactive web applications, meanwhile JavaScript was meant to be used within form-based applications. The launch of Web 2.0 and AJAX (Asynchronous JavaScript and XML) marked a significant shift in the role of JavaScript in Web development.

JavaScript or ECMAScript is a user-friendly object scripting language designed to develop dynamic web applications that seamlessly connect objects and resources on both clients and servers [18]. JavaScript originated as a tool for HTML page authors and corporate application developers to dynamically script the behavior of objects carried out on either the client or the server. Now, JavaScript is undoubtedly perhaps the most widely used programming language around the globe. The World Wide Web has grown to be ubiquitous, with over a billion websites accessible via billions of internet-connected devices. Each of these devices runs a Web browser or a comparable program that can handle and display pages from these websites. Additionally, a substantial number of these websites integrate or fetch source code generated in the JavaScript computer language. Some numerous characteristics and features put JavaScript in favor of other worldwide languages, some of them are:

1. **Dynamic:** Since the first release of JavaScript 1.0/1.1, it is considered a dynamically typed language, featuring five fundamental data types: number, string, Boolean, object, and function. In essence, declaring it as "dynamically typed" stands for declaring the type of a variable or data at runtime (execution time), in contrast to statically typed languages where types of variables or data are declared in compile-time. In brief, types are associated based on the store value and not the type itself, meaning that the usage of the let or var keyword before indicating the variable name, results in creating a variable regardless of its type, with the ability to change its value from e.g., a string to object [18].
2. **Prototype-based:** JavaScript is a prototype-based language, a term based on its distinctive perspective on object inheritance [18]. This technique contrasts with other object-oriented languages such as Java, in which objects depend on class inheritance, creating classes that operate as templates to produce instances (objects) with predetermined attributes and behaviors. Nevertheless, JavaScript utilizes a prototypal inheritance, thereby defining the object's prototype and using it to produce more objects of the same type. In a nutshell, each object contains an internal reference to an additional object that is labeled as its prototype. When a new object is generated, using the constructor functions or object literals, it also derives the attributes and methods from its predecessor prototype. Upon accessing an element of an object, that element will additionally be searched on its prototype, the predecessor of that prototype, and so forth, forming a chain until either a property with a matching name is found or the end of the prototype chain is reached [19].
3. **Imperative and structured:** The syntax of JavaScript was based on the syntax of the C programming language, implementing their expression statements (e.g., the if conditional statement, the for and while iteration statements, the continue, break, and return, etc.). However, one major difference is that it didn't follow C's declaration style as its prementioned is a dynamically typed language, and its statements are not limited to residing inside the body of a function. JavaScript uses a script containing statements and declarations, that are embedded within HTML documents, defined by a <script> </script> tag [18].
4. **Async Processing:** JavaScript is considered an asynchronous processing language, allowing particular operations to be concurrently initiated, without the need of waiting for the completion of pre-operations, thus fully supporting the asynchronous I/O model of Web browsers [18]. This is achieved by the usage of Promises and Async

functions. In Promises, the programmer makes a request attaching with a .then() clause which indicates that the clause will only be executed upon the completion of the promise. As for the Async functions the programmer can use the async-await to write asynchronous code that has similarities with the synchronous code, with the particular characteristic of its functions being executed simultaneously rather than sequentially, optimizing the overall performance [20].

Figure 3. A JavaScript Logo

2.2.1 JS Frameworks

Through the intricate nature of AJAX-style apps, as well as browser interoperability challenges, application frameworks, and libraries have sprung up to ease the development of web applications [18]. JavaScript is the unparalleled stronghold in the immense web market, bringing a plethora of sophisticated frameworks and libraries that provide structure and ease to the web development process [17]. It is ubiquitous in the whole web, masterfully coordinating both client-side code and server-side logic. The sumptuous collection of feature-rich libraries and frameworks enables developers to easily infuse improved capabilities into their applications, minimizing the necessity for complex coding. These JavaScript frameworks offer a streamlined blueprint for complex operations, simplifying them into elegant segments of code to accelerate the development procedure. JavaScript frameworks are constantly expanding and improving, bolstering their dominance in the web domain. JavaScript frameworks are organized according to their key use cases and characteristics. JavaScript frameworks are classified into various categories some of the most common applications are: Front-end, Back-end, Data layer, and JS automation testing frameworks and test runner environments [17].

Since the application is web-based, priority must be given to the selection of the optimal framework. Among the several revered JavaScript framework categories, selecting a framework from the front-end JS framework is the one that is essential for the implementation of the application, because it will play a key role in facilitating the development of the application and integrating various technologies efficiently. Moreover, the web framework must be able to offer the best possible interactions with the user while at the same time collecting and managing the large volume of large data to be visualized at will. This implies that it should update the application in real-time to reflect any changes in the data immediately. Indirectly, therefore, it needs to have mechanisms and techniques to optimize its performance, i.e., methods to respond and support all stages of visualization (collection, processing, visualization, etc.) quickly. At the same time, bearing in mind that big data is expanding, the framework must adapt successfully. Having fulfilled the aforementioned conditions, it should provide or support a gamut with powerful visualization and mapping libraries. The following is the chosen framework used for the development of the system.

2.2.1.1 React

React is an open-source JavaScript framework with the exception that it is not a standard "framework, but rather a JavaScript library that was initially developed by Facebook engineers to address the issues associated with creating elaborate user interfaces with dynamic datasets [21]. Despite that, React stood out as the pinnacle of popularity and functionality and evolved to become as extensive and component based as any other framework. Acknowledging the challenging nature of the task, the engineers sought a solution that would additionally be maintained nonetheless scaled to satisfy the necessities of Facebook's extensive platform. The initial development of React has been traced back to Facebook's ads organization, which utilized a traditional client-side Model-View-Controller (MVC) methodology. However, React brought revolutionary changes to web development, dramatically altering the way apps were produced. When it first appeared in 2013, it attracted both interest and skepticism in the web development world. Aside from its fundamental shift to front-end development, React offers an abundance of features that democratize the manufacturing of single-page applications and user interfaces for developers of diverse ability ranges [22].

At its core, React handles the intricate issue of displaying data in a user interface, which has been already addressed by preceding frameworks. However, React was deliberately designed to suit the needs of large-scale interfaces, dealing effectively with dynamic data. As projects become more sophisticated maintainability becomes an urgent challenge. Small projects that initially surfaced doable could grow into cumbersome endeavors involving the addition of patched approaches to secure data binding and interface presentation. Unforeseen business requirements may further aggravate issues, resulting in fragile, interdependent, and unmaintainable user interfaces [22]. Such are the issues that React aims to overcome, relying on the key elements of its architecture.

To gain a thorough understanding of React's architecture one has to address its relationship with the MVC pattern. There are currently several frameworks that are inspired by the MVC pattern and implement it or a variant of it (MVVM, MV*). A basic MVC architecture is composed of the model, view, and controller. The model is in charge of controlling the state of the program and notifying its changes to the view. The view represents the user-facing aspect of the system, including all the interactions, with the potential of distributing events to the controller and, in certain situations, the model. Lastly, the controller serves as the primary event dispatcher, thus acting as an intermediary between the model and the view, informing the model of state changes, and triggering changes in the view's presentation. That MVC structure doesn't represent React's architecture as it was designed to be just the view of that pattern [22]. The following features form the foundation of React's architecture.

Virtual DOM: The Virtual DOM (VDOM) is a programming technique that involves storing an ideal or virtual representation of a user interface in memory and syncing it with the actual DOM using a library such as ReactDOM [23]. This process is named reconciliation which is an algorithm that React uses to distinguish between trees to identify which parts need to be updated [24]. The Virtual DOM acts on top of the real DOM or another rendering target and uniquely approaches the state manipulation problem. When changes occur in the DOM, the virtual DOM efficiently propagates these changes across its virtual tree. This technique benefits from React's declarative API, allowing developers to declare the desired state of the UI, and in return, React guarantees that the real DOM replicates that state. As a result of that, attribute manipulation, event handling, and manual DOM updating are kept separate from developers, which would otherwise be necessary to develop an application. Given that "virtual DOM" refers to a pattern rather than a specific technology, its meaning may differ based on the context. In React, "virtual DOM" relates primarily to React elements, which are the objects that define the user interface. React also uses internal objects known as "fibers" to store further details about the component tree. React Fiber's primary goal is to improve its adaptability and performance in areas such as animation, layout, and gestures. React Fiber incremental rendering capabilities, via enabling the division of rendering tasks into smaller portions and distributing them across several frames [24]. These "fibers" can be regarded as an aspect of React's "virtual DOM" implementation. Since manual DOM manipulation is inefficient and difficult to optimize, this technique of DOM manipulation leads to elevated performance [23].

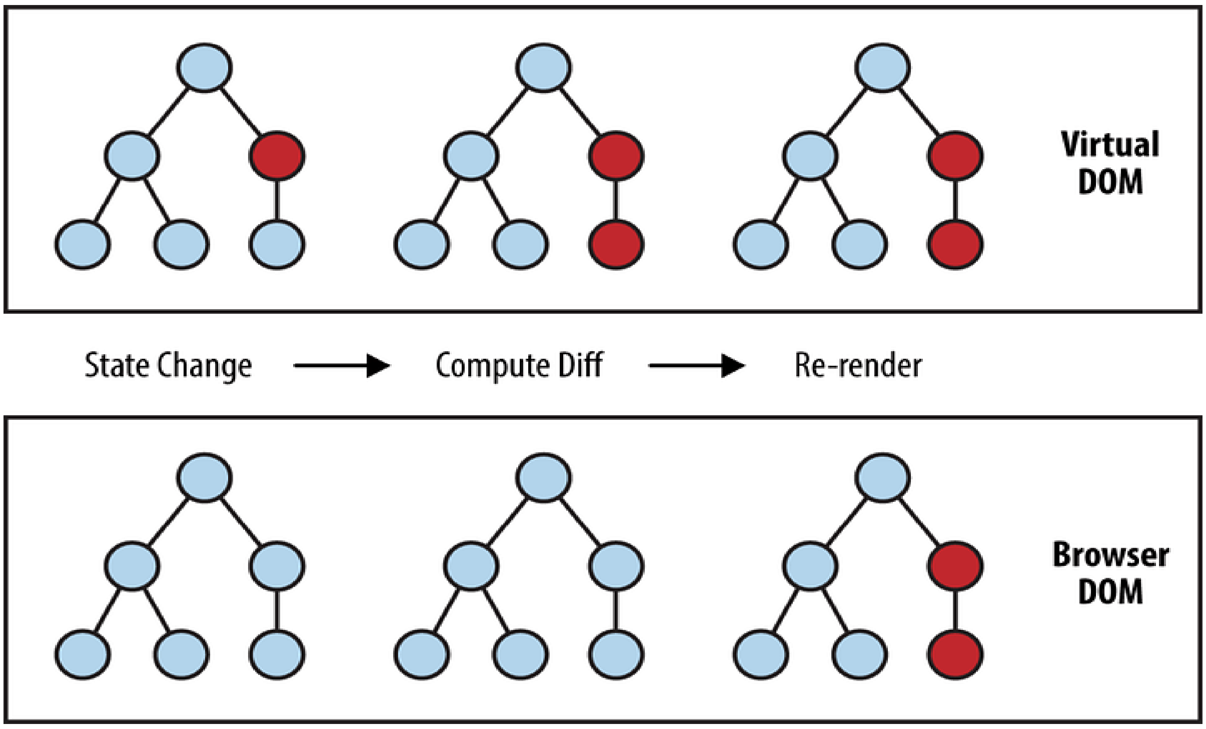


Figure 4. Performing calculations in the Virtual DOM limits rerendering …

Components: React components are the basic building blocks of any React application, which collectively form the entire system. A React component can be expressed as a JavaScript class or function that receives properties (defined below) as input and returns a React element that defines the layout of an area of the User Interface (UI). React components, featuring a completely independent structure, methods, and APIs, offer reusability and can therefore be effortlessly incorporated into interfaces as desired. The UI is assembled mostly with these components, which form a hierarchical tree structure with the initial component functioning as the root and distinct components branching out, further separating into sub-branches. This arrangement safeguards that data and state changes flow properly from the root to its branches and sub-branches. Notably, React components provide direct client-side server calls, resulting in dynamic DOM modifications eliminating the need for page refreshes. Each component has its interface, allowing server calls to be made independently, guaranteeing that refreshing one component will not impact others. Furthermore, React components have high encapsulation, reusability, and composability, allowing developers to conceptualize, acquire knowledge, and construct user interfaces more straightforwardly and elegantly. Once a component is generated, lifecycle methods are used to guide the application through various stages of its lifespan, such as mounting, updating, and unmounting, to enable efficient reuse and management [23].

Properties: Properties in React are typically accessed as "this.props" since this represents the most typical manner of referring properties within components. Properties are an assortment of features that a component has and can be expressed in React as a simple JavaScript object. It is vital for one to understand that props are immutable since they remain unchanged during the component's lifecycle. In case an update is needed within the component, then the state object is the best practice it reflects the alterations in the component's functionality and presentation. Through utilizing this approach, developers can preserve the architectural integrity of React by ensuring the proper handling of properties and states within their components [23].

States: States in React are established upon component initialization and may alter through the component's lifecycle. It ought to be noted that the state shall not be attainable via an outside component absent a parent component being responsible for determining or modifying the child component's initial state. For the purpose of keeping components organized and manageable, it is preferable to use state objects as sparsely as possible. The complexity of components increases when states are added, and it may affect the behavior of the component. As a result, it is advisable to use states with prudence while at the same time attempting to create components that rely on them as minimally as feasible. Instead of overloading a single component with states, it's frequently wiser to divide the UI into smaller, more manageable components, each with its own specialized responsibility and state management [22].

JSX: JSX, which is short for JavaScript XML, allows developers to formulate HTML-like syntax within React. It simplifies and enhances the process of integrating HTML code into React components. Although JSX frequently behaves and resembles HTML, it is important to understand that its primary purpose is to be converted into JavaScript. JSX started as a research project at DeNA and developed into an expanded version of JavaScript syntax. It is a statically typed, object-oriented programming language designed for contemporary web browsers. Given that it has an XML-like structure, preprocessor operations can be used to introduce XML syntax into JavaScript. A more elegant and polished coding technique is produced as a byproduct of the incorporation of JSX into React. JSX syntax contains opening and closing tags as a pair containing names, attributes, and child elements just like XML [23]. JSX serves as an intermediary layer, translating the XML-like syntax necessary for generating React components towards the JavaScript syntax utilized by React to render elements. Whilst JSX is not essential in React development, it has tremendous value and considerably simplifies application development. JSX syntax does not solely support custom React classes; it also supports standard HTML tags[22].

Data Flow: In React, data flows horizontally across components in a unidirectional flow, allowing each component to update the other. This implies that data cannot explicitly gain access to or alter the state or props of other components or their parents when parsed through a component. Callbacks, on the contrary, allow programmers to create interactions among components. By receiving a call back from a child component, the parent component can update its data and fetch the revised data back to the child. The architecture of React is fundamentally based on this unidirectional flow of data. The parent passes the data down the component tree, ensuring that the data continues to flow downward regardless of callback scenarios. This offers important advantages when developing user interfaces since it eases the comprehension of how data flows through the program, resulting in a more logical and well-structured development process [23].

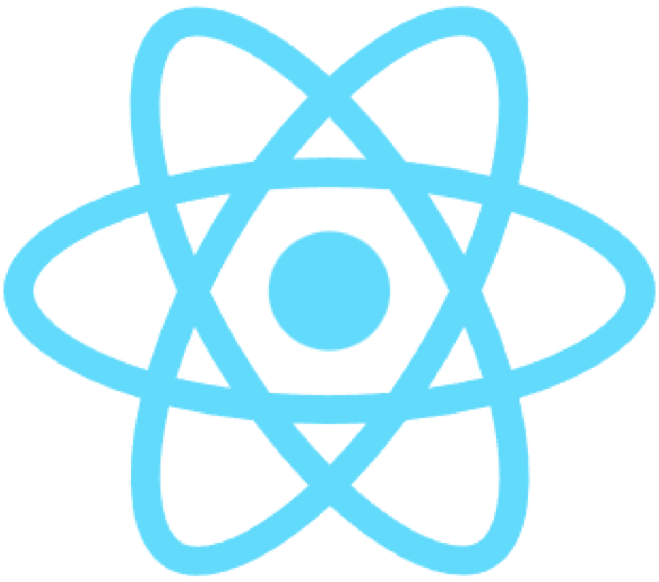


Figure 5. React Logo

2.2.1.2 React vs Angular

Angular is Google's robust JavaScript framework for swiftly creating complex single-page web apps. It uses TypeScript, a superset of JavaScript that adds features like types, interfaces, async functions, and decorators before compiling into plain JavaScript code interoperable with all browsers. Angular applications are built using modular programming techniques, which entail separating program functionality into logically distinct and interchangeable modules. The modular system used by Angular is known as "NgModules." Each NgModule wraps components or service providers within a coherent unit, with the contained NgModule defining the scope of these pieces. Every Angular application requires at least one module, referred to as the "AppModule," which acts as the root module. A NgModule is represented by a class that has been annotated with the @NgModule() decorator and contains various properties that describe the module [25]:

- "declarations": Module-specific components and directives.
- "exports": Components and directives that other modules should be able to access.
- "imports": Other modules that this module requires.
- "providers": Services exported by this module to the global collection of all services.
- "bootstrap": The foundational component.

Angular is available in two separate versions, AngularJS, and Angular 2+. The elder sibling, AngularJS, is based on JavaScript and, nevertheless, is no longer in active development. On the contrary, Angular 2+ which debuted in 2016 heralded a new age by integrating TypeScript, a powerful superset of JavaScript. This move marks AngularJS's first TypeScript-based rewrite, distinguishing it from its predecessor along with other frameworks in the market. Angular 2+ became a more popular and rightfully preferred version over its predecessor. Implementing TypeScript unleashes a slew of benefits and performance boosts, thereby offering an appealing option for developers looking for a modern and simplified development experience. While it is technically possible to utilize Angular without TypeScript, doing so is frequently difficult and, as a result, not advised in principle. Given the similarities between the two versions and considering that AngularJS is no longer actively maintained, the study of searching for the optimal framework does not provide an in-depth examination of AngularJS. Instead, the emphasis is on examining the innovations and capabilities provided by Angular 2+ versus React [26].

Figure 6. Angular Logo

There are several aspects in which frameworks can be compared in search of finding the optimal for this thesis use case:

Popularity and Community: In the fast-paced world of web development, measuring a framework's popularity and community's support is essential. It not only represents developer preferences but also throws insight into technological developments and improvements. Three reliable parameters have been carefully assessed for this analysis [25]. To begin, a review of Google's online searches over the last five years revealed significant information about people's curiosity and interest in these frameworks. With its dynamic and efficient nature, React emerged as the clear frontrunner, collecting the most searches (see Figure 8). Additionally, its excellent growth rate in the course of these 5 years demonstrates the growing interest in this framework among developers [27]. Furthermore, study results on Stack Overflow, a well-known developer community, contributed to a better understanding of the frameworks' practical use and importance. React once again stole the show by having the most questions asked and answered on this platform (see Figure 9). Angular was close behind, having the upper ground between 2016 to 2018, indicating its continued appeal, even though it must be mentioned that React skyrocketed after 2018 while Angular started losing ground [28]. The amount of GitHub repositories was used to calculate community support and open issues, and the results suggest that React has the most community support gathering currently more than 211k stars, with Angular coming in second with 89.4k. However, it must be mentioned that when Angular 2+ was released, a unique repository different from Angular.js was created starting the number of stars and commits from the beginning. Also, Angular has more open issues than React showcasing its complexity and React's ability to overcome its issues. These frameworks' dynamic and involved communities play a critical role in their ongoing growth and enhancement, providing useful tools, plugins, and active discussions to aid programmers across the globe [25].

Performance: The Angular and React frameworks were evaluated using the Js-framework-benchmark, with an emphasis on startup metrics and memory allocation. The comparison results indicate that Angular has the highest overhead and requires the most RAM allocation of these two frameworks. To evaluate the concrete applications developed in each framework, key features such as user login, user registration, and product purchase in an online shop with an integrated products cart were assessed. Compiling these project sizes and the number of code lines for both apps results in React being the more lightweight framework with the prospect of further lowering code lines with the Hooks concept. Meanwhile Angular has the highest overhead due to its vast node modules[25]. Two deciding factors that lead to the aforementioned results include React utilizing the Virtual Dom, therefore, achieving rapid performance but also simplifying the development process, and secondly their data managing techniques. Specifically, React uses a unidirectional data flow as opposed to Angular's two-way binding in which data flows into the template from a component via property binding and then out of the template using event binding, a process that is sophisticated and may stall the program [29].

Learning Curve: A framework's Learning Curve includes numerous fundamental factors that contribute to its acknowledgment and ease of use. Some factors that affect the learning curve of a framework are the available documentation, the knowhow requirements, and the human resources and recruiting. A well-written and thorough documentation is essential for a library or framework to be adopted, and it ought to point out that both frameworks have vast and thorough documentation. Carrying on, with the knowhow requirements React is plain sailing for developers with experience in or substantial knowledge of JavaScript given that almost everything has been converted to JSX. On the other hand, Angular is way more complicated, and acquiring a full picture of all the available options takes time. Yet another time-consuming feature is that Angular requires the full adaptation of Typescript. Moreover, it provides less freedom, predefining decisions for particular scenarios. This feature is described as "opinionated", meaning that most Angular projects have similar structures but also this is considered as a limitation. Finally, in terms of human resources and recruitment, both React and Angular are supported by gigantic organizations of Facebook and Google respectively, offering several jobs either in their offices or by other companies that base their development on their technologies. All in all, Angular has a rather steep learning curve meanwhile React is way easier to adapt [29].

Migrations: Migration is the process of moving from earlier versions of a framework to newer ones. The primary objective of JavaScript frameworks is to render the migration process easier for developers. Angular typically provides version changes every six months, which can be difficult for developers. However, they provide migration support through their website to help with the process. React, on the other hand, seeks to keep the number of significant releases to a minimum. When such releases are made, they include automatic scripts that are run during the migration process, making the transfer easier for developers[25].

Flexibility: In terms of flexibility, Angular follows a tight structure in which everything is offered within the Angular package and developers are forced to conform to the stated code architecture. React, on the other hand, provides greater freedom because it doesn't put limits on the application structure and can easily connect with other libraries [25].

2.2.1.3 React vs Vue

Vue.js is a lightweight JavaScript developed by Evan You who formerly worked at Google and Meteor. Evan You, who formerly worked at Google and Meteor, created Vue.js, a lightweight JavaScript library. Throughout his time at Google, Evan observed that, although Angular is powerful, it may be cumbersome in several circumstances. Driven by the ambition to create a more efficient solution, he embarked on an endeavor to extract the quality aspects of Angular to form his own lightweight library. Vue.js, abbreviated Vue, may be regarded as the most recent framework in this comparison, despite being released two years before Angular 2. This claim stems from the fact that Angular 2 is based on the foundations of its predecessor, AngularJS. Vue is frequently lauded as an innovative framework for creating web user interfaces. Although not restricted to the Model-View-ViewModel (MVVM) pattern, Vue's design concepts were influenced by it. Vue's adaptability allows it to be used in a wide range of contexts, from simple projects combining the core library with other technologies to complex Single-Page Applications (SPAs) [29].

Vue's key features rely on its reactive data binding architecture, which allows for seamless data and visual synchronization. Following the data-driven view approach, Vue binds data from the model via specific interpolation in HTML, comparable to Angular's property name code for property access. Vue also makes use of a number of directives, prefixed with v-, to give HTML elements specific functionality. The v-if, v-for, v-model, and von: click are some of the available directives. Every Vue application starts with the creation of a Vue instance, and giving an options object to this instance adds properties to Vue's reactivity system. Any changes to these properties result in automatic updates to the associated view. Vue like React is a component based framework. Vue components are merely Vue instances containing data, templates, and optional methods. To ensure that each instance has precise data component's data needs to be a function. Templates for components require a single root element, whereas single-file components created with the Vue extension are used for more complicated applications. Single-file components, similar to React's approach, consist of a template, script, and style making it easier to handle smaller logical units of components within a single file [25].

Figure 7. Vue logo

In order to compare React with Vue, there will be employed the same criteria that were used to compare React with Angular:

Popularity and Community: Vue is notable for being a product exclusively created by the open-source community, rather than being pushed by a huge corporation. Vue began as a personal hobby project for its founder, Evan You, and hit a critical milestone when he devoted himself full-time to its improvement. Patreon, a community site that allows individuals to support specific projects or authors through monthly subscriptions, was used entirely to fund the project. Surprisingly, Vue received tremendous financial support from its patrons, with monthly payments reaching $4,000 initially and rising to nearly $15,000 by April 2018. Vue's growing popularity might also be attributable to its strong presence on Google trends, GitHub, and Stack Overflow.

Vue has acquired quite the community considering its origins but React still holds a larger popularity and community support, having an everlasting growth since its release. Be it on any of these 3 massive parameters, React holds the upper hand, gathering the most web searches on Google (see Figure 8), the most stars in GitHub (211k over 205k), and the most activity in Stack Overflow (see Figure 9).

Performance: To analyze the performance of these frameworks, it should be indicated first that both of them are designed to be lightweight technologies. Following the same performance techniques that were implemented in React vs Angular sector, React has less script bootup time but it requires a bit more memory allocation. Moving on to the project sizes, Vue is the most lightweight framework [25]. To analyze these frameworks further there is a need to determine their data processing. Vue supports both two-way and one-way data binding, offering developers more freedom in data processing. Vue excels over React in rendering and processing efficiency, given a reduced codebase. However, due to its development team's scale constraints and unexpected official upgrades, Vue's technical assistance could prove less reliable. Moreover, the difference in volume suggests that Vue includes only essential features. Considering these factors, Vue is particularly suitable for small and medium web projects that favor flexibility and easy development meanwhile React thrives for more sophisticated projects with everchanging datasets as in this thesis project [30].

Learning Curve: In this field, Vue seems to have the upper hand. Since both frameworks have exceptional documentation, the comparison should be spread among the knowhow requirements. Vue doesn’t necessitate an adaptation to any new technologies as its whole existence depends on simplicity, ease of use, and basic front-end technologies such as HTML, CSS, and JavaScript. In the interim, React requires an adaptation to JSX and hooks but is also considered an easily adaptable language. Also, regarding human resources and recruitment, React has a massive community and a powerful organization, Facebook, supporting it, factors that yield more job offers [29].

Migrations: As previously reported React opts to minimize releases based only on significant occasions. Vue throughout development, maintains a consistent Application Programming Interface (API), guaranteeing that around 90% of the API remains intact. However, in the event of minor adjustments, they also give migration tools to aid developers during the migration process [25].

Flexibility: React and Vue both demonstrate impressive flexibility in the development of applications. To be more precise, both frameworks avoid putting limitations on their application structure, granting developers the freedom to maneuver according to their own interests. Furthermore, they contribute effortlessly to a variety of different libraries, increasing their versatility to a wide range of project requirements [25].

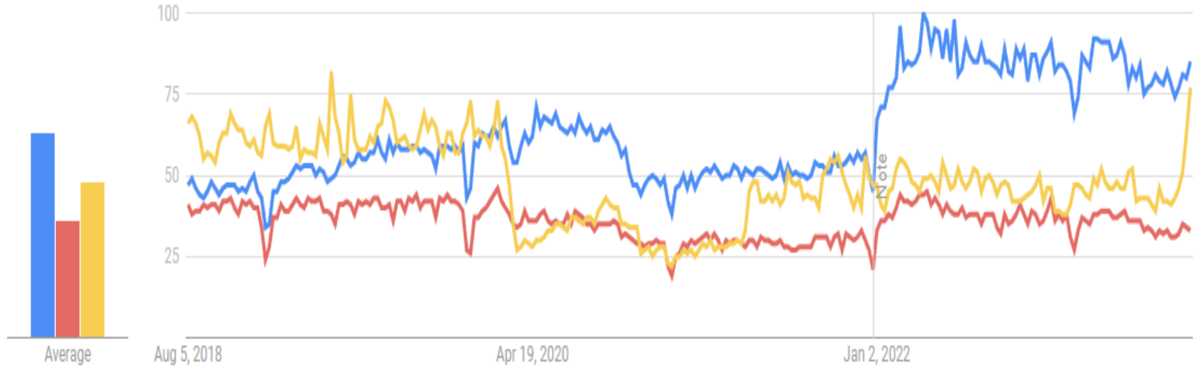


Figure 8. Google Trends over React (blue), Angular (red), Vue (yellow)

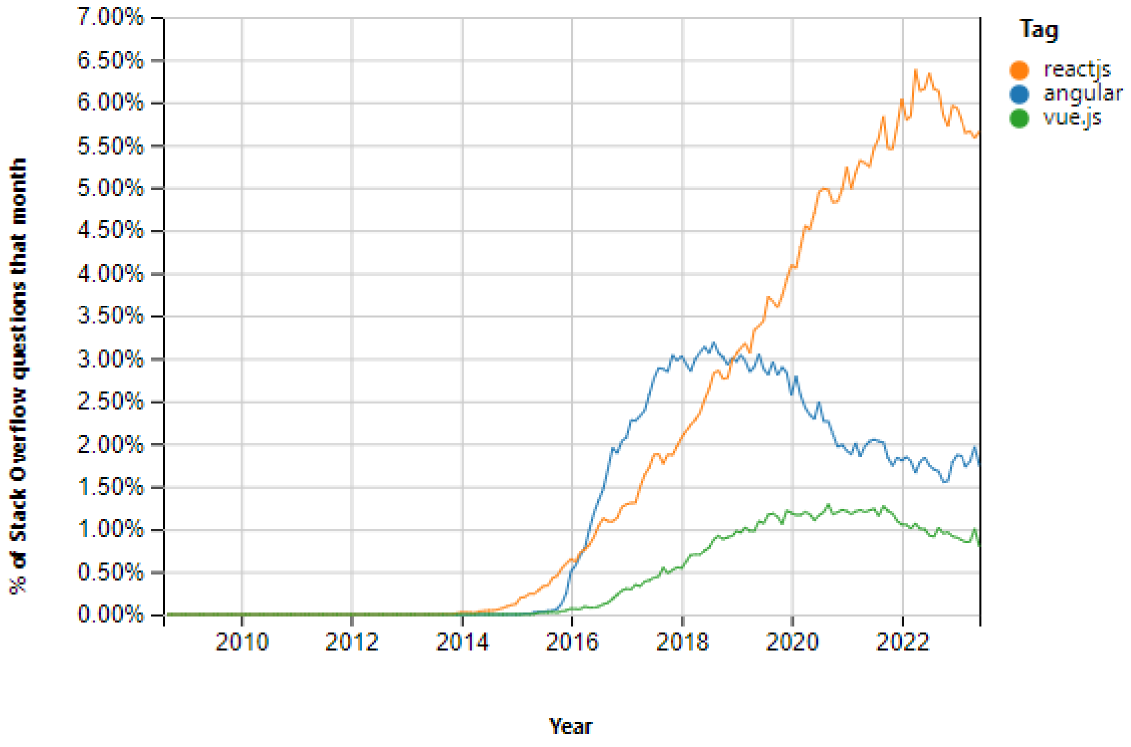


Figure 9. Stack Overflow Trends over React, Angular, Vue

2.3 Hybrid Development

As the optimal framework is being decided (React) one more initial factor must be taken into consideration and that's the possibility of a hybrid development. The hybrid app development process is concentrated on producing an application using a single project that is capable of handling several platforms such as Android, iOS, and Windows. The appeal of hybrid software lies in its ability to run an application on various platforms using a single codebase. Aside from certain areas that require platform-specific customization, such as the user interface, developers can write the code once and reuse it for several target platforms. The increasing demand for hybrid apps is driven by two undisputed factors: their relative ease of development compared to native apps and the ability to expand a company's presence across numerous platforms, including the web, using the same source code [31]. The following are two distinct types of hybrid development:

1) **Hybrid Mobile App Development:** Developers in this approach utilize web technologies such as HTML, CSS, and JavaScript to create mobile apps that can operate on many platforms such as iOS and Android. These programs are wrapped within a native container, allowing them to be distributed through app stores in the same way as native apps are. One of the best Hybrid App frameworks, if not the best, is currently React Native.

   React Native is a strong cross-platform mobile app development framework built on React that was first launched by Facebook in 2015. This open-source JavaScript framework uses JSX to make it easier to create multi-platform user interface components. These components are then linked to native code and turned into Android and iOS native views, allowing developers to create mobile applications that work flawlessly across platforms. Although it contains the term native, React Native is a hybrid framework, which distinguishes it from the properties of the term native. Native app development is specific to a single platform, focused solely on Android or iOS. This technique necessitates devoted efforts to create unique applications for each platform, lengthening the development process in comparison to cross-platform solutions. Despite that React Native enables native development that focuses attention on a single platform, its primary role is to provide cross-platform development that also allows developers to write platform-specific code if necessary, allowing them to optimize the app's performance on each platform [31].

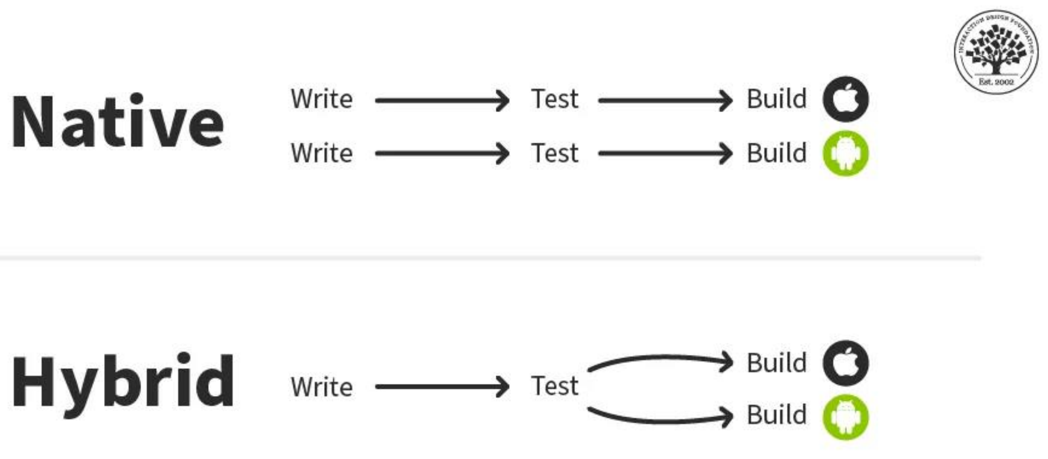


Figure 10. Hybrid App Development vs Native App Development

2) **Hybrid Web App Development:** Hybrid Web App Development entails creating web applications that are designed for both desktop and mobile platforms. These apps are accessible via a web browser and do not require installation through app stores. Frameworks such as React Native Web can be used to create web applications that look and feel similar to native apps[32].

Meanwhile, React isn’t a hybrid development framework, it can easily coordinate with React Native if the needs of this system require a mobile application or both mobile and web implementation using React Native Web. Finally, React is provided with the capability to create a Progressive Web App (PWA) which offers a compelling set of benefits similar to responsive websites and native mobile apps. They are an all-encompassing solution that allows for the seamless delivery of a site or application across all devices, eliminating the need for app-store distribution difficulties. PWAs are fundamentally webpages; nevertheless, they have been painstakingly built to mimic the appearance and feel of a native app. PWAs provide an immersive and engaging user experience by leveraging progressive enhancement and sophisticated web technologies, blurring the barrier between standard web pages and native applications. PWAs are an appealing solution for businesses looking for an adaptable and efficient approach to accommodating varied user devices while also providing a rich and app-like user experience [31].

2.4 Map Systems

Moving onward to the system's necessities, the next step is to decide on an appropriate map system. In order to pick the best-suited map library, it is essential to initially identify and classify the correct map system category. Various categories of map systems are used for various purposes and they tend to be classified into three main types:

1. **General Purpose Maps:** General Purpose Maps, often known as basemaps or reference maps, depict natural and man-made features of broad interest. They are intended for general public usage and provide a thorough overview of geographical aspects [33].
2. **Thematic Maps:** Thematic maps, also known as special purpose, single topic, or statistical maps, showcase particular characteristics, events, or phenomena. The data that these maps project can be qualitative, quantitative, or a combination of both. As a result of the above characteristic thematic maps are further classified into two types: qualitative thematic maps, which show the spatial extent of categorical or nominal data (for example, soil type, land cover, and political districts), and quantitative thematic maps, which show the spatial patterns of numerical data (for example, income, age, population) [33].
3. **Cartometric Maps:** Cartometric maps, a subset of maps, are intended for precise measurements and cartometric analysis. These maps are designed to be efficient for mathematical activities like counting, measuring, and estimating. Aeronautical and nautical navigational charts used for routing over land or sea are examples of cartometric maps, as are USGS topographic maps, which are commonly used for tasks requiring precise distance calculations, such as surveying, hiking, and resource management [33].

   Listed below are a handful of well-known map systems:

   1. **Political Map:** Political Map is a style of cartographic representation that illustrates the borders of countries, states, counties, towns, and cities, as well as notable water bodies, usually portrayed in blue. This map depicts administrative units and their corresponding borders, as well as political divisions and territorial demarcations. A Political Map, for example, may display the borders of states within a country alongside the international

borders between countries. It is a valuable instrument for comprehending the geopolitical landscape and the distribution of administrative bodies on a regional or worldwide scale [34].

2. **Physical Map:** A Physical Map is a type of cartographic representation that focuses on displaying the geographical attributes and natural elements of a specific location. Such maps provide detailed visual information about the topography of the land, highlighting key features such as mountains, rivers, and lakes. In the context of physical maps, different hues are often used to designate various characteristics; for example, blue is commonly used to depict water bodies, green represents lower elevations, and brown represents greater elevations. In addition, physical maps are frequently embellished with labels to identify significant geographical phenomena such as rivers, mountains, and lakes, which improves the viewer's comprehension of the area. These maps are useful for geographers, travelers, researchers, and anybody else interested in discovering and comprehending a country's natural terrain [34].
3. **Topographic Map:** A topographic map is a cartographic representation that emphasizes the vertical dimension of the Earth's surface. It provides useful information on the heights of the terrain, allowing users to quantify and interpret the height variations in a certain region. Topographic maps, like physical maps that highlight diverse natural landscape elements, may include contour lines to represent elevation changes rather than depending entirely on colors. These contour lines, which are meticulously spaced at regular intervals, provide a clear visual representation of the landscape's ups and downs, allowing people to grasp the topographical profile effectively. Topographic maps have several applications in geology, geography, engineering, surveying, scientific education, and military operations. Topographic maps, particularly in navigation-related applications, serve an important role by providing accurate descriptions of both natural and man-made elements visible from various views. This richness of information assists navigation by allowing users to fully comprehend their surroundings and plan their trips accordingly [34].
4. **Climate Map:** A Climate Map is a visual representation of the predominant climatic conditions in a specific area. These maps are extremely useful for displaying the region's numerous climatic zones, which are mostly determined by temperature trends. Climate maps efficiently represent the various climatic areas by using a variety of hues, and a Map Legend is included with the map to assist in recognizing each zone based on its associated colors. A climate map additionally includes a series of charts that show the changes in crucial variables such as temperature and precipitation during a specific period. To illustrate the temperature gradient over the displayed location, these charts use a color spectrum, frequently ranging from cool colors like blue to warmer tones like dark brown [34].

5. **Navigational Chart Map:** Navigational charts, often known as nautical charts or navigation charts, are cartographic representations that are specifically adapted to the navigation needs of sailors and other watercraft users. These charts meticulously portray the shoreline or seafloor of a body of water, providing critical information for safe marine transportation. Notably, navigational charts include a plethora of critical details, such as depth contours, precise depths at various points along a designated route, tidal elevations indicating water levels on specific parts of the shoreline, and tidal streamlines indicating tide direction and speed. Additionally, these charts display critical landmarks, navigational aids like waypoints and beacons, and important information on potential hazards such as shoals or reefs. Therefore, navigational charts are essential tools for sailors to identify safe passage routes, providing safe and efficient travel across the ocean [34].
6. **Road Map:** Roadmaps have developed as one of the most common and widely used map kinds in modern times. These cartographic representations highlight both large and minor roadways, as well as other important transportation features such as airports and ports. On top of that, roadmaps contain useful information about the locations of cities and items of interest such as parks, hotels, recreational areas, and historical monuments. The adaptability of roadmaps allows them to be used for a wide range of purposes, from trip planning and tourism to navigation aid. Roadmaps help travelers understand the environment and navigate through unknown areas by effectively illustrating roads and notable landmarks. Roadmaps have developed as one of the most common and widely used map kinds in modern times. These cartographic representations highlight both large and minor roadways, as well as other important transportation features such as airports and ports. On top of that, roadmaps contain useful information about the locations of cities and items of interest such as parks, hotels, recreational areas, and historical monuments. The adaptability of roadmaps allows them to be used for a wide range of purposes, from trip planning and tourism to navigation aid. Roadmaps help travelers understand the environment and navigate through unknown areas by effectively illustrating roads and notable landmarks. Therefore, roadmaps have proven to be vital tools, allowing people to explore new places with assurance and ease [34].
7. **Cadastral Map:** Cadastral maps are essential instruments for registering and documenting land ownership within a country, state, or municipality's territorial boundaries. These maps, known for their comprehensiveness, include a plethora of information, making them valuable resources for a variety of reasons. The term "Cadastre" derives from the concept of "division," which refers to the allocation of property ownership for taxes purposes. Given that they provide a comprehensive view of land utilization and ownership patterns, they are particularly useful to revenue agencies, land surveyors, and land co-operatives. Cadastral maps enable stakeholders to make educated decisions, promote equitable taxation procedures, and facilitate effective land

management initiatives by methodically documenting critical land-related data. Cadastral maps are essential components of modern land administration and governance systems because they are long-lasting and dependable cartographic representations [34].

2.5 Geographic Information System (GIS)

A geographic information system (GIS) is a sophisticated system for creating, controlling, analyzing, and cartographically representing various datasets [35]. It traces back to the 1960s, a time defined by the introduction of computers and the growth of quantitative and computational geography. During this period, pioneering research by the academic community provided the framework for GIS. Notably, the National Center for Geographic Information and Analysis, led by Michael Goodchild, was instrumental in formalizing research on critical areas in geographic information science, such as spatial analysis and visualization. These scholarly efforts sparked a radical quantitative revolution in the field of geographic research, laying the groundwork for the following evolution and advancement of GIS. GIS seamlessly connects data to geographical maps, combining spatial information (location-based data) with detailed descriptive information (attributes). Because of this integration, GIS has become a critical tool in scientific study and nearly every business. Users acquire insights into patterns, correlations, and spatial context by using GIS, which leads to improved communication, efficiency, and more informed management and decision-making processes. GIS's multifarious benefits extend across all disciplines, making it a critical component in modern data-driven initiatives. GIS has transformed the practices of innumerable companies in a wide range of industries around the world. GIS is used by hundreds of millions of organizations to create useful maps, supporting effective communication, data analysis, and information exchange. These firms solve complex difficulties by leveraging the potential of GIS, resulting in significant improvements in the way the world runs. GIS has become an invaluable instrument that continues to alter and reinvent numerous parts of modern society due to its broad range of applications and transformative potential.

Geographic Information Systems (GIS) in modern applications operate through the seamless coordination of numerous technologies, consisting of maps, data, analysis, and apps.

Maps: Maps are the primary geographic containers for storing data layers and supporting analytical processes in spatial Information Systems (GIS). These GIS maps provide a simple and quick way of sharing and embedding crucial spatial data into a variety of applications. GIS maps become accessible to a large audience, overcoming geographical borders, and reaching people all over the world. GIS maps' adaptability and user-friendliness enable people from all walks of life to engage with geospatial data and harness its insights for better decision-making, collaboration, and problem-solving efforts [35].

Data: Geographic Information Systems (GIS) are defined by their capacity to connect disparate data layers based on their spatial placements. Almost all data has a geographic component, making it suitable for incorporation into GIS. This integration includes a wide range of data, from images and geographical features to foundational basemaps that are closely linked to tabular datasets and spreadsheets. GIS data's spatial context allows for more in-depth knowledge of relationships and patterns, allowing for more thorough analysis, informed decision-making, and better communication across various fields and sectors [35].

Analysis: Geographic Information Systems (GIS) spatial analysis enables the evaluator to assess the suitability and capacity of sites, create estimates and projections, comprehend complicated phenomena, and get a deeper understanding of spatial patterns. Its adaptability allows for the acceptance of new views, enriching insights, and promoting informed decision-making processes. GIS goes beyond mere visualization by using spatial analysis to become a powerful tool for discovering, unwinding, and grasping the complicated interactions between geographic variables, thereby boosting innovation and efficiency across multiple domains [35].

Apps: GIS apps provide purpose-driven user experiences, allowing individuals to fully utilize GIS capabilities. These applications work as dynamic tools, bringing GIS functionality to life, and are available on a variety of devices and platforms, including mobile phones, tablets, web browsers, and desktops. GIS apps provide ubiquitous accessibility integrating with numerous devices, guaranteeing users utilize GIS features wherever they may be, adapting to the demands of modern workflows, and improving productivity [35].

2.6 Web-Based Platforms for GIS

Web-based GIS platforms have altered the way geographic information is accessible, processed, and visualized. These platforms employ the internet and modern web technologies to provide users with easy access to geographical data and geospatial tools. These platforms enable users to build interactive maps, do spatial analysis, and communicate geospatial information with a large audience by integrating GIS functionality into web applications. Web-based GIS platforms have democratized geographic data, allowing individuals and organizations all over the world to access it. Web-based GIS applications cover a wide range of areas, from fundamental globe maps to sophisticated tools for understanding spatial distributions and processes. These applications use various technologies and database platforms, with preference determining the overall performance of the web-based system. Web-based GIS applications, at their most basic, provide user-friendly maps, while at the other end of the spectrum, they enable comprehensive spatial analytics. The adaptability of GIS-enabled web apps enables them to accommodate a broad spectrum of requirements, from plain visualization to comprehensive geospatial modeling. Given the variety of technologies available, developers can customize these applications to provide an optimal user experience while meeting specific data management and analysis requirements. As requests for geographical data and analysis expand, web-based GIS applications remain at the forefront, providing an accessible and robust way of comprehending and exploiting geographic information.

Considering that the application is indeed a web-based app that after careful consideration is powered by React, the appropriate library needs to be identified, which will meet the requirements of the application but at the same time will be compatible with React. The optimal map library should be tuned for handling huge datasets with multiple data points and complex shapes. It should efficiently render the map and data layers to provide smooth interactions and responsiveness, even when dealing with large amounts of geospatial data. The library should include a plethora of data visualization capabilities such as heatmaps, clustering, point density maps, choropleth maps, and custom overlays as data visualization variants. These visualizations should be adaptable and capable of successfully expressing big data insights. From the UI perspective, it should provide a user-friendly environment enriched by interactive features such as click events, hover interactions, and custom tooltips, to engage the user to trigger a change or to collect useful information from the input forms, enhancing his overall experience. For the purpose of achieving that, the library should be able to cope with dynamic data loading and visualize data on demand, particularly in a big data project that incorporates massive datasets, whose insufficient render could lead to overwhelming the application. Finally, for developers to understand and efficiently utilize the map library, a thorough and well-documented API, as well as active community support and ongoing updates is mandatory.

2.6.1 MapLibre GL JS

MapLibre GL JS is a cutting-edge TypeScript library utilizing WebGL for displaying immersive and interactive maps and it is deemed as the optimal library for covering this system's necessities. Its outstanding performance is due to the use of GPU-accelerated vector tile rendering, paving the way for instantaneous map display even with massive datasets. The library complies with the MapLibre Style Spec, allowing significant map styling customization possibilities. It collaborates with MapLibre Native, which is built for mobile, desktop, and servers, as part of the MapLibre ecosystem. MapLibre GL JS began as an open-source fork of Mapbox-GL-JS v1, nevertheless, its development has resulted in considerable improvements and modifications that exceed the scope of its forefather. Notably, with Mapbox's change to a non-OSS license in December 2020, it emerged as an open-source alternative. MapLibre GL JS has grown in popularity as a resilient and feature-rich solution for providing developers with dynamic and visually appealing web-based mapping services [36]. MapLibre has an abundance of capabilities and services to offer, some of the most prominent are:

Vector Tiles Rendering: The map is divided into a grid of square tiles, with each tile carrying geometric data for a specific location. This segmentation is consistent across all zoom levels used in the application, ensuring that map data is rendered consistently at different scales. These tiles can be provided in an image format to MapLibre GL JS to represent the map's properties. Notably, the vector-based data within these tiles encompass the subtleties of geometries, encompassing city names, height curves, and other geographic attributes. This vector data provides incredible versatility, allowing for customization and styling for a variety of use cases prior to rendering as a tile. This versatility extends to displaying the same data at different resolutions for different zoom levels without blurring or loss of clarity. This method allows map data to be adapted to individual zoom levels, with each data file corresponding to a given geographic area at a specific zoom level. A single file contains all of the necessary information for generating a tile for those zoom levels with specified data, speeding the map visualization process and improving overall performance [37].

Style Document Structure: A MapLibre style is a document that specifies how a map should look stylistically, including essential characteristics such as which data to display, the order in which to draw the elements, and how to accurately style them. This style document is organized as a JSON object, with particular root-level properties and hierarchical settings that affect the map's appearance and functionality in minute detail. The aforementioned description targets three key groups:

- **Cartographers and advanced designers:** This particular method gives assistance in manually building map styles for expert designers and cartographers seeking aesthetic control, allowing them to create distinctive and visually compelling map designs.
- **Developers:** MapLibre GL JS and MapLibre Native for Android and iOS developers will get useful insights into working with style-related features programmatically. They may effectively integrate and adjust map styles to provide end-users with customized and interesting map experiences.
- **Authors of software:** Authors of software that design or process MapLibre styles can use this specification as a reference. It ensures that its product adheres to industry standards, allowing for easy interoperability and consistency when working with MapLibre styles.

Delving deeper into the core elements of MapLibre styles it is composed of root properties, each serving a specific function. A MapLibre style is a comprehensive collection of root properties, each with its own function. Several root properties represent single global attributes, while others are intended for nested attributes. Several root properties, such as "version," "name," and "metadata," provide information without affecting the map's appearance or behavior. Instead, they provide mandatory detailed information about the map. In contrast, root attributes such as "layers" and "sources" are important since they indicate the presence of map features and define their visual representation. These attributes play an important role in creating the overall map design and functionality. Furthermore, characteristics such as "center," "zoom," "pitch," and "bearing" provide default values to the map renderer, acting as a basic guide throughout the first display of the map [36].

Events and handlers: MapLibre features a powerful event system with a diverse set of event types and handlers, which improves interactivity and user engagement in map applications. MapLibre events are triggered by user activities including mouse clicks, touch movements, and keyboard inputs, allowing developers to respond to these actions and adapt map behavior accordingly. Event handlers can be registered by developers for specific event kinds, allowing them to precisely collect and respond to user events. For example, "click" events can be used to detect and select map features, and "move" events can be used to dynamically update map items as the user navigates the map. MapLibre even supports custom events allowing developers to generate and trigger their own events for specific circumstances. This extensibility enables more versatility and diversity when dealing with map interactions [36].

MapLibre

Figure 11. MapLibre Logo

2.6.1.1 React Map Gl

In the context of the remarkable technological convergence with a primary focus on the efficient integration of the library into the React environment, an integration between MapLibre GL JS and React Map GL has emerged. React-Map-GL is an ensemble of React components designed to be compatible with Mapbox GL JS-affiliated libraries. At the moment, its compatibility extends to three main libraries: MapLibre GL JS, Mapbox GL JS v1, and Mapbox GL JS v2. It is deemed even possible to implement React-Map-Gl with other Mapbox-GL forks but it is not a recommended use case. This library provides a seamless connection within the React environment, allowing MapLibre GL JS to be utilized as a completely controlled and responsive component. Furthermore, it demonstrates inherent flexibility by providing auxiliary React APIs, such as context and hooks, allowing for the incorporation of customizable components. Remarkably, it is part of VIS.GL's Framework suite; is a comprehensive set of tools designed for web-based geospatial visualization and analytics with GPU acceleration [38].

Uber's Visualization team spearheaded the creation of React-Map-Gl, employing Mapbox GL JS-compatible libraries as a key component for building powerful web tools encapsulating complicated geospatial analytics and self-driving data visualization applications. To properly address such applications' complexities a thorough dedication to the React framework and its fundamental reactive programming was established. The basic Mapbox-Gl APIs are imperative, which means that orders such as map.flyTo are issued, and the map performs these activities autonomously and at its own rhythm. Nevertheless, when dealing with multiple components that require synchronization, this strategy becomes inadequate. For instance, a pair of maps may be displayed side by side, necessitating synchronized updates of both cameras when interacting with one map. Furthermore, React-based UI elements that adapt to camera motions are incorporated outside the map container. WebGL visual overlays are also layered on the map. Shared states must be maintained through React to enable proper synchronization across all components. The authoritative data source may reside in a parent component state, Redux store, or hooks and it cascades down to the map and its related peers. Following the tenets of the reactive programming paradigm, a constant data flow pattern is maintained, in which data continually flows downward. While Mapbox-Gl controls its own state, there is a possibility of component disparities if this internal management is relied on alone [38].

2.6.2 MapLibre GL JS vs Leaflet

Leaflet is a popular open-source JavaScript library that delivers interactive maps suitable for mobile devices. With a small JavaScript size of about 42 KB, it has a complete set of mapping features that satisfy the bulk of developers' demands. Designed by Vladimir Agafonkin, it is widely utilized for many web mapping applications given its simplicity, flexibility, and devoted development community. Its effectiveness extends across major desktop and mobile platforms, displaying seamless operation. Furthermore, its extensibility is boosted by a large number of plugins. The API is naturally navigable, and thoroughly documented for straightforward usage. The basic and understandable source code that underpins its qualities fosters a great environment for collaborative contributions [39].

Given the circumstances that both libraries are equally valued and appreciated by the general public, the decision between Leaflet and MapLibre was parsed based on this thesis project criteria. Leaflet specializes in lightweight projects to appeal to a more general audience. Its simplicity, cross-platform compatibility, and ease of use are considered its strengths. However, its integration in this project may even be considered misguided considering the potential negative performance it may face when managing huge datasets. Conversely, MapLibre GL JS thrives in visualizing and optimizing massive datasets via the utilization of GPU-accelerated vector tile rendering. MapLibre may also have a steeper learning curve but makes up for it by handling sophisticated visualizations and interactions effortlessly.

Figure 12. Leaflet Logo

2.6.3 MapLibre GL JS vs Google Maps

Google Maps is Google's web-based cartography platform and consumer application. It includes satellite imagery, aerial photography, comprehensive street maps, immersive 360° panoramic views of streets via Street View, real-time traffic data, and itinerary planning for various modes of transportation such as walking, driving, cycling, and public transportation. By the year 2020, Google Maps has amassed a monthly user base of over one billion people worldwide. Google Maps began as a C++ desktop software built by the Rasmussen brothers, Lars and Jens, while they were working at Where 2 Technologies. The acquisition of the company by Google in October 2004 represented an important turning point in the software's evolution into a web-based application. Following investments in a geospatial data visualization firm and a real-time traffic analysis entity, Google Maps was officially launched in February 2005. In September 2008, Google Maps was released for Android and iOS devices, bringing GPS-assisted turn-by-turn navigation and specialized parking guidance capabilities. It had the distinction of being the most extensively used smartphone application in the world as of 2013, with a remarkable 54% of global smartphone users engaged with it [40].

Considering the impact that Google Maps has in modern society it isn't an exaggeration to indicate that is the most widely used SDK for incorporating a map into an application. It undeniably encompasses a range of functionalities such as traffic conditions, street view, immersive view, etc. Nevertheless, the comparison is established based on custom implementation cases such as this thesis. Google Maps is not open source, offering only the usage of the map itself as cost-free. Moreover, all the other features are considered premium and payment must be fulfilled. Visualizing data is also included as a premium feature and needless to say in a big data visualization project that would be highly costly. Rendering also can be resource-intensive, resulting in brief freezes throughout the rendering process. Finally, markers and shapes are limited to a single static display. On the flip side, MapLibre GL JS is an open-source library and it inherited several of the technologies that Mapbox GL JS has specialized in over the years, notably vector tiles. In conclusion, MapLibre GL JS is the optimal choice for this project offering great performance, flexibility, big data adaptability, thoroughly crafted documentation, and an active community [41].

Figure 13. Google Maps Logo

2.7 Data Visualization Tools

In the sphere of providing a complete data visualization perspective, a synergy between maps (MapLibre) and specialized tools unfolds, reaching a more comprehensive and coherent display of information. Building upon the preceding chapter's assessment of the significance of big data visualization, the following inquiry focuses on the tools that aid this process. As aforementioned, data visualization is the graphical depiction of information through illustrative design with the aim of delivering complicated information in a visually comprehensible manner. Data visualization approaches can be grouped into two types: univariate and multivariate data visualizations. The primary focus of the univariate category is on plotting a single variable to discern distribution patterns and scatter plots. Multivariate approaches, on the other hand, investigate the complicated interactions between numerous datasets and elements. The domain of data mining tools is categorized into three fundamental types: programming languages, statistical tools, and visualization tools [42].

Prior to delving into the optimal tool, it is necessary to establish the project demands for this tool. Starting with the most important aspect the scalability. A powerful big data visualization tool must be able to handle massive amounts of data without sacrificing performance or responsiveness. It should be able to adapt to changing data quantities and growth rates with ease. Furthermore, it should have a variation of customization features, allowing users the flexibility of modifying the chart types, color schemes, labeling, etc., according to their preferences. Compatibility is also an essential consideration, requiring the chosen library to be implemented efficiently with other technologies employed in this project, placing greater emphasis on its compatibility with React. Ultimately, having an intuitive and interactive user interface while maintaining comprehensive and enriched documentation, becomes imperative.

2.7.1 D3.js

D3, also referred to as D3.js, is a free and open-source JavaScript library designed for data visualization. Its innovative technique, based on web standards, provides unprecedented flexibility for generating dynamic, data-driven graphics. D3 has been at the vanguard for well over a decade, delivering pioneered and award-winning visualizations. It has developed into a core component for more advanced chart libraries, and it has fostered an active worldwide community of data practitioners. Mike Bostock created D3 in 2011, a journey that led him to co-author the D3 paper with Jeff Heer and Vadim Ogievetsky at Stanford. Jason Davies also made significant contributions between 2011 and 2013, particularly in defining D3's geographic projection framework. Philippe Rivière has been a substantial contributor to both D3 and its extensive documentation since 2016. Throughout the years, there has been a huge contribution from users offering guidance via sharing code and ideas, uniting a community to improve the field of visualization [43]. D3 pioneered data visualization through its cutting-edge features that set it apart from the rest of the data visualization libraries:

Low-level toolbox: D3 exceeds the traditional definition of a charting library. Instead, it functions as a diverse toolset with no preset concept of "charts." D3's visualization approach is constructed by merging several essential features. To create a stacked area chart, for example:

- a CSV parser would be used to load data,
- time and linear scales would be utilized for positioning,
- ordinal scales and categorical schemes would be employed for color differentiation,
- a stack layout would be applied for data arrangement,
- a linear curve with an area shape would be used to generate SVG path data,
- axes would be used to explain position encodings,
- and selections would be used to create SVG elements.

It is critical to understand that D3 is a collection of 30 separate modules rather than a single monolithic entity. These modules are grouped together for convenience rather than necessity, allowing designers to access their toolbox as they iterate through the design process [43].

Flexibility: Since D3 lacks an overarching "chart" abstraction, building a simple chart may necessitate writing a few dozen lines of code. However, this approach has the advantage of explicitly presenting all of the components, allowing complete control over the visualization process. This assortment of customization allows the user to precisely adjust the visualization to their interests. D3 is not provided with a predefined data presentation format; instead, it relies on code written by the user or derived via examples. D3 could be regarded as an alternative to a high-level charting library for attaining full customization. When conventional tools are judged insufficient and the thought of creating personalized charts utilizing SVG, Canvas, or even WebGL occurs, investigating D3's toolbox can be beneficial. It is highly feasible that it has an asset capable of facilitating the production of the desired chart while additionally permitting the user to express their innovative abilities [43].

Web Compatible: D3 preserves a unique approach By refraining from new graphical representations. Rather, it's compatible with web standards like SVG and Canvas. The term "D3" is an abbreviation for "data-driven documents," where "documents" signifies the Document Object Model (DOM) standard that dictates the representation of a webpage. Although certain D3 modules (for example, selections and transitions) interface with the DOM, others (for example, scales and shapes) only change data. D3 can also be connected with web frameworks such as React which is also vital for this application. The integration of D3 with web standards provides various benefits. External stylesheets, for example, can be used to adjust chart appearances in response to media queries, simplifying the building of adaptable charts or accommodating dark mode. The use of the debugger and element inspector aids in the review of code execution. When opposed to frameworks with complex asynchronous runtimes, D3's synchronous, imperative evaluation model—in which executing selection.attr immediately mutates the DOM, thus simplifying debugging [43].

Bespoke Visualization: D3 enables possibilities, though not always with simplicity. In cases where it pertains to maximizing the expressive potential of a bespoke graphic, D3 is specialized. D3 is an excellent match for media outlets like The New York Times or The Pudding, where one graphic may captivate a large audience and a collaborative editorial team can cooperatively expand the domain of cutting-edge visual communication [43].

Dynamic Visualization: D3 adds a novel idea known as the "data join." Provided with a dataset and a set of Document Object Model (DOM) elements, data join supports the implementation of distinct procedures for entering, modifying, and exiting elements. The fundamental objective of the data join is to provide precise control over the repercussions when data changes. This direct oversight produces highly efficient updates that alter only the particular components and attributes that require modification, eliminating the need for extensive Document Object Model comparison. As a result, smooth animation transitions between states are finally conceivable [43].

Figure 14. D3 Logo

2.7.1.1 Collision between D3 and React

The integration of React and D3 provides several beneficial outcomes. This collaborative association enables the establishment of a customizable repository including reusable charts, which is enhanced by React's features. However, the challenge that lies in that merging is the nature of D3 and React, two technologies based on data-driven modification of the Document Object Model (DOM). Considering DOM manipulation is inherent in both libraries, merging them requires prudence. This DOM "competition" is also highlighted in D3's documentation, dedicating the whole chapter of "D3 in React" to segregating the D3 modules that interact with the DOM and those that don't. It had a precautionary approach to that issue that also ended with proffering a preventive measure to their fellow programmers [43]. Through substantial research and evaluation conducted by the active community of D3 and React, three alternative solutions have been formulated:

1. **To each its (DOM) land:** The simplest and most straightforward solution entails merely deactivating React within the structure of D3. This may be accomplished in a variety of ways, including rendering an empty div/> with React to serve as D3's scope or returning false from the "shouldComponentUpdate()" lifecycle method. The main concern with this technique, which is really effective, is that it strips away the beneficial qualities that React brings to the D3 environment. This method, in particular, resulted in decreased rendering performance due to heavy DOM modification, a situation in which React's reconciliation algorithm may have optimized processing by saving milliseconds of execution time. Furthermore, the use of React's tooling and developer experience (DX) benefits is undermined [44].
2. **D3 for the math, React for the DOM:** The second solution which is presently the most popular, entails solely utilizing D3's utility functions to preprocess data, configure axes, and do other relevant operations. Following that, the processed data is passed to React for rendering. The traditional D3 data binding mechanism is not used in this case; rather, data processing is organized through React, which frequently necessitates the

definition of keys for SVG elements. In D3, such an approach is occasionally required, particularly when complex data binding circumstances exceed D3's autonomous resolving. Notably, a noteworthy divergence from the traditional D3 is the incorporation of JSX to draw SVG elements, rather than the usual "`d3.(...).append()`" method [44].

2.a. Enter/Exit with React, Update with D3: This solution has also an enhanced version that improves its foundations by incorporating ideas from the first solution. The basic idea is to render SVG elements using D3's helpers and JSX. A significant distinction resides in the fact that React elements are purposefully absent of attributes. D3 instead steps in to add these properties. Notably, the distinction between React and D3 switches from an element based as implemented to the first solution to an attribute level. While apparently insignificant, this distinction is critical in reintroducing D3 transitions. All in all, this technique greatly aids in maintaining a clear line of ownership between the two, with React managing the structure and D3 managing the attributes. By doing so, D3 can transition elements – change their fill color, location, and size — without interfering with React's principles [44].

3. **Feed D3 a fake DOM that renders to state:** The core idea is to provide a fabricated Document Object Model (DOM) to D3, to include the entire range of methods in regular DOM interactions. D3 performs modifications that are in accordance with its standard process within this fake DOM. These alterations are then rendered automatically, turning into React elements that are stored within the component's state, where React can manage changes and trigger an update [44].

The second methodology and in particular «2.a» was considered the best for the implementation of this project, as it has multiple implementations, stable results, and a clear preference among the general community. The first solution had performance issues and the third has proved inefficient due to the lack of implementations that indicate it may still be in the experimental stage. In a nutshell, offering React DOM control using memorized methods and parsing the elements as JSX, while allowing D3 to perform the bulk work for calculations and interactivity, establishes a synthesis that balances both entities' strengths.

2.7.2 D3.js vs Chart.js

Chart.js is an open-source JavaScript library that operates as a powerful data visualization tool. This library efficiently makes use of the HTML5 canvas to render various chart types, positioning itself as one of the most prominent JavaScript charting libraries on GitHub. Chart.js popularity stems from its simplicity, thereby making it accessible to any individual with a basic understanding of HTML, CSS, and JavaScript. It provides comprehensive documentation for learners which is distinguished by its straightforward articulation and clarity. However, this simplicity comes with certain constraints in terms of breadth and features. Currently, the library contains eight different chart types: bar, line, area, radar, polar area, scatter, bubble, and pie. The process of creating visualizations with Chart.js consists of three simple steps [45]:

- Create the canvas element that will render the chart.
- Setting up chart-specific options.
- Parsing data into the chart for visualization.

Chart.js was developed and officially launched in 2013, but its evolutionary journey has taken a long way since then. This library operates within an open-source principle, accommodated by the MIT license, and is overseen by an active and engaged community. Chart.js features a robust default setup, allowing for an easy start-up process that results in a production-ready application. Even with the lack of explicit customization, the outcome will probably be a visually appealing chart. As an illustration, Chart.js employs animations by default, instantly focusing attention on the narrative weaved by the underlying data [46].

Figure 15. Chart.js Logo

Prior to initiating a comparison, it must be clarified that for the purpose of data visualization, both D3.js and Chart.js are exceptional libraries. Chart.js focuses on establishing a more user-friendly environment requiring significantly less effort in chart generation and its implementations should suffice in the vast majority of scenarios. On top of that, it provides an array of default features reaching a far better first-time experience, establishing a highly-level chart right from the start. These features include an interactive environment followed by Legend, animations, and responsiveness automatically without any need for programming. However, those features that deemed it as a starting-level data visualization tool also contribute to its limitations. Chart.js has a limited variety of chart types which are currently estimated at eight and although it offers several traits, they are distributed as a standard chart [47]. D3.js, on the other hand, operates without the constraints of predetermined chart types, instead relying on the developer's creativity. While it can undoubtedly handle basic charts, its capability expands beyond displaying data in a variety of unique ways. Furthermore, Chart.js only supports Canvas to visualize charts in contrast with D3 which supports both Canvas and SVG. That feature can indulge in huge performance differences particularly when the scene must be updated. Since SVG utilizes the DOM to monitor elements, individual elements can be updated without affecting the entire scene. Nevertheless, in the Canvas situation, any change involves the complete redrawing of the entire scene. Overall, D3.js has a much more pronounced learning curve, demanding full coding to achieve even a standard outcome, however, this translates into a heightened flexibility that will ultimately stipulate a better performance in a big data visualization scenario [45].

2.7.3 D3.js VS Tableau

Tableau is a proprietary business intelligence solution that boasts widespread popularity as a visualization tool. It is recognized for producing interactive visualizations swiftly according to its straightforward drag-and-drop capability. Its broad toolbox includes a variety of visualization options, including maps, heat maps, scatter plots, bubble charts, pie, bar, and bar charts. This diversity renders it remarkably simple to generate insightful dashboards from diverse datasets. Notably, Tableau skillfully handles aggregations, highlighting or drilling down within charts, even assisting novice users to uncover insights within massive data sets. The tool seamlessly integrates with data stored in formats including Excel, CSV, and text file recognizing fields and formats. However, in cases where data is stored in various databases using the right database connections and having the necessary knowledge is crucial. Tableau's strength comes in its ability to integrate with a variety of platforms, such as the powerful Big Data framework Hadoop, the statistical analytics tool R, and its impact on the Google Big Query API, despite the fact that it only offers limited analytical capabilities. Although it offers direct native support for proprietary Big Data companies such as Amazon Red Shift and Teradata, it falls short in terms of integrating with popular open-source NoSQL databases such as MongoDB. Tableau Software provides a variety of options that satisfy the diverse requirements of users, including Tableau Desktop, Tableau Server, Tableau Mobile, and Tableau Public. Tableau Desktop is designed for individuals and small businesses, whereas Tableau Server is designed for larger companies. Tableau Mobile supports tablet and mobile phone users, which is a useful feature considering the ubiquitous usage of smartphones.

Tableau Public, on the other hand, is free and provides small-scale visualizations, but it has less functionality than Tableau Desktop, making it unsuitable for complicated big data visualizations [48].

Figure 16. Tableau Logo

In the domain of data visualization, Tableau and D3.js stand out as powerful innovative tools, each with diverse methods and capabilities. Tableau is a business intelligence tool; a visualization software package that is distinguished for its remarkable features and user-friendly experience. This tool is proprietary and the majority of its essential features for sophisticated big data visualizations are accessible only through a paid subscription. On the other hand, D3.js is an open-source JavaScript library, that doesn't require any form of payment, encouraging its users to implement all its features and experimentalize them, free of cost. Furthermore, Tableau has a small learning curve, initiating Drag and Drop techniques so its users can experience its ease of use right from the start while are also provided with a variety selection of built-in charts and maps. Conversely, D3 has a steeper learning curve due to requiring everything to be built from scratch, but that's actually one of its major strengths as it grants the freedom to its implementors to develop any imaginable visualization that is possible to code. In essence, Tableau guarantees convenience, whereas D3.js provides limitless creativity for creating detailed and customized data displays, making it the ideal selection for this application [48].

3 Application Architecture

Software architecture is the framework for establishing, implementing, and managing application components that accommodate user functionality [49]. Inevitably software architecture serves a vital role as a link across demands and code. The architecture broadcasts particular elements whilst concealing others by providing an abstract description (or model) of a system. This illustration gives a strategically comprehensible orientation to the broader system, enabling architects to make assumptions about a system's ability to meet particular needs, and proposes a blueprint for system development and synthesis [50].

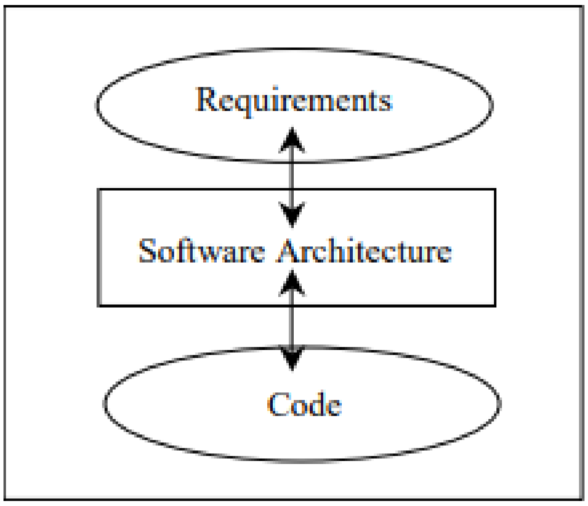


Figure 17. Software Architecture as a Bridge

The subsections go through key elements that contribute to the application's operating dynamics and structure. The initial element corresponds with input and output analysis, delving into user interactions with the program and the outcomes that occur from it. The class diagram offers a blueprint of key components and their interrelationships, as well as a glance at the coding process. Moreover, functional requirements address the system’s features and functionalities. Finally, the non-functional requirements are implemented to ensure the application’s prosperity and longevity.

3.1 Input/ Output

As regards this thesis implementation, the first input originates from the user and it can be obtained either by navigating through the components of the home page to inspect his property that consists of fields, crops, and stations, or by accessing the map. Subsequently, users can utilize the system's beneficial characteristics to observe their property, evaluate the displayed data, and facilitate their field's and crop's condition. Starting from the first scenario, the user’s navigation through the home page components culminates in a final page. Upon reaching that final page the system triggers a data request from the server via a socket call. After the acquisition of data and further processing, interactive graphical projections provide thorough insights into the current state of his fields as an output.

The system adeptly captures the user’s mouse position as a second-level input during his interaction with the charts. As the user moves his mouse through the SVG area, that input notifies the system, and as an output, it returns precise information about that mouse coordinate in a D3 text, detailing the name, value, and date based on the closest available data point of each line. Another aspect that completes this particular event is the addition of circles to these aforementioned data points along with a line that is formed to occupy the full height of the chart that also passes through the circles, highlighting the event's purpose. Furthermore, another noteworthy feature is the “Brush”, a mini version of the main chart located right underneath it, which also serves as a second-level input. The system captures the user’s mouse movement inside the mini chart, changing the cursor's appearance dynamically in response to the user's interactions, altering its symbol to reflect the precise activity being performed. Moreover, the system also captures any click events inside the brush area, initiating the main “brush” event that enables users to click and drag to choose a specific location within the chart. As an output, this designated area can be used for a variety of purposes that affect the main chart including zooming in on a specific time range or highlighting a portion of data.

A significant input/output association is also established through the map page. Furthermore, the system takes the user’s input to load the map, via the drawer menu that is always displayed in the up-right corner of the app or via the map button that appears by navigating through the home page. The only factor that distinguishes these two access methods is the data visualization in which the first displays all available data, and the second showcases only the data relevant to the user's current view on that specific page. A subsequent feature of the second method is the exhibition of a data symbol on the map page that will redirect the user to the specific page where he clicked the map icon. Upon rendering the map component there are multiple events regarding the user’s activity through the map that assembly the second-level input to the system. The user can interact with his property which consists of polygons and markers that translate to his fields, crops, stations, and collectors. These events apprehend:

- **mouse clicks:** This action yields specific information about the clicked polygon or marker.
- **mouse hovers:** It triggers the display of a pie chart giving a quick analysis of his properties condition.
- **mouse scrolls:** It demonstrates a marker clustering that organizes the visualization of markers. The nearby markers are grouped and become clusters based on the map’s zoom level. Whenever users zoom in or out on the map, the clustering dynamically changes to display more or fewer individual markers and clusters, giving a seamless experience and enhancing the overall performance of the map.

Finally, one last major second-level input is through the control panel that is rendered alongside the map. As its name suggests it’s a container that comprises checkboxes, numeric inputs, and sliders, and as an output it provides a customized map based on the user’s preferences. One major highlight of that customization is the ability to switch between heatmap data, while it's also noteworthy that it can display a mini D3 chart upon the initial click on a polygon area, reminding the user to assess the state of their fields and crops. Overall, these input/output features enhance a user-friendly environment while also maintaining a highly interactive experience.

3.2 Class Diagram

Class diagrams are an important aspect of the Unified Modeling Language (UML) among the six fundamental types of structural diagrams. They are an essential component in the object modeling process, outlining the static composition of a system. Depending on the system's complexity, an individual class diagram can model the entire system or several class diagrams can be employed to encapsulate the various components of a system. Class diagrams serve as the blueprints of a system or its subsystems. They facilitate the ability to depict the system's constituent objects and the interrelationships that bind them while also offering an overview of their functions and services. Class diagrams are quite beneficial at various stages of the system design lifecycle. First, class diagrams can serve as a navigational tool throughout the analysis phase, assisting with grasping the requirements of the issue domain and identifying its core components. In the context of object-oriented software development, the initial class diagrams generated during the project's early phases frequently transform into actual software classes and objects as the coding process unfolds. As the project progresses, these class diagrams can be further refined to demonstrate explicit components of the system, such as user interfaces and logical implementations. As a result, the class diagrams represent a snapshot that delicately elucidates the system's mode of operation, the intricate interdependencies of system components across different layers, and the roadmap devised for their implementation [51].

The implementation of this system’s class diagram has some unique approaches in order to fully showcase React’s characteristics. Even though React is based on JavaScript, which is considered an object-oriented language, it differs from traditional Object Oriented languages. Moreover, React is considered a combination of Object Oriented and Functional programming, which is currently evolving toward a more Functional approach. However, this still indicates that there are some parallels between React and OOP that will direct the formation of the diagram [52]:

- **Classes:** React is declarative and component based. Within the conventional Object-Oriented Programming (OOP) pattern, React facilitates the construction of objects that incorporate and operate data without expressly dictating their application. React components serve as the fundamental building block of a React application. Those components will be represented as classes in this diagram along with their props as these classes’ arguments, as well as key features or state manipulator functions as its operations.
- **Relationships:** Regarding the matter of relationships between classes, this diagram is based on the component relationships. Moreover, it is based on the parent/child relationship in which one component is nested within another component and in some

rare cases the owner-based relationship wherein a component sets the props for another. Factoring in those above relationships as well as contemplating that the child doesn't inherently know who their parent component is, the most suitable relationship turned out to be the unidirectional association. In unidirectional association, two objects are connected, but only one of them is aware of that relation [53]. Moreover, that is an extra way to display the data flow in the diagram as indicated in the previous chapter React has a unidirectional data flow as the parent passes the data down the component tree, ensuring that the data continues to flow downward.

- **Multiplicity/Cardinality:** In the context of class diagrams the representation of multiplicity serves as a means for indicating a more in-depth overview of the relationship that links the two components [53]. Furthermore, the cardinality of each relationship is based on the number of initial mounts that the parent component may render the child component.
- **Utility Classes:** In traditional class diagrams, a utility class is distinguished by the presence of only class-scoped static attributes and actions. In essence, a utility class often lacks instances because it functions primarily through its static parts [54]. Utility classes are linked with the dependency relationship. In React there are utility functions/classes that are closely associated with helper functions/classes. In that use case, it offers much more freedom in the structure of the utility files without requiring a full static approach. Moreover, it is noteworthy that the helpers and utility functions may differ in the implementation purposes as the first offer assistance or aid to different areas of the program, and the latter is used for general tasks that are not bound to any particular area of the program [55]. Nevertheless, both types provide methods and functions that will enhance the coding process, by organizing it, significantly reducing the amount of code, and increasing its clarity. As a result, both fall within the <<utility>> class type.

Presented below is the class diagram that is developed based on the aforementioned characteristics, accompanied by a thorough study of each component.

Map.web:

This is the initial phase of one of the two main visualization techniques that were conducted in this thesis, the map component. The ".web" extension indicates that it is designed for web purposes and that can also have an identical name file with the ".native" extension. This is due to React allowing its users to define a module that can have various implementations and dependencies based on the platform. Then the user can import the file name without extension and React will decide which version to render based on the platform, paving the way for a native approach in the future. The map component can be accessed by either the drawer menu or through the map icon, which will be displayed after navigating the home page. This file is responsible for notifying the system that the user is utilizing the map along with the method he reached based on the route ID. Finally, this component renders the actual map of the system, the "DisplayPolygons" file, with the route ID provided as a prop. Given that it initially renders it once, it exhibits a multiplicity of "1".

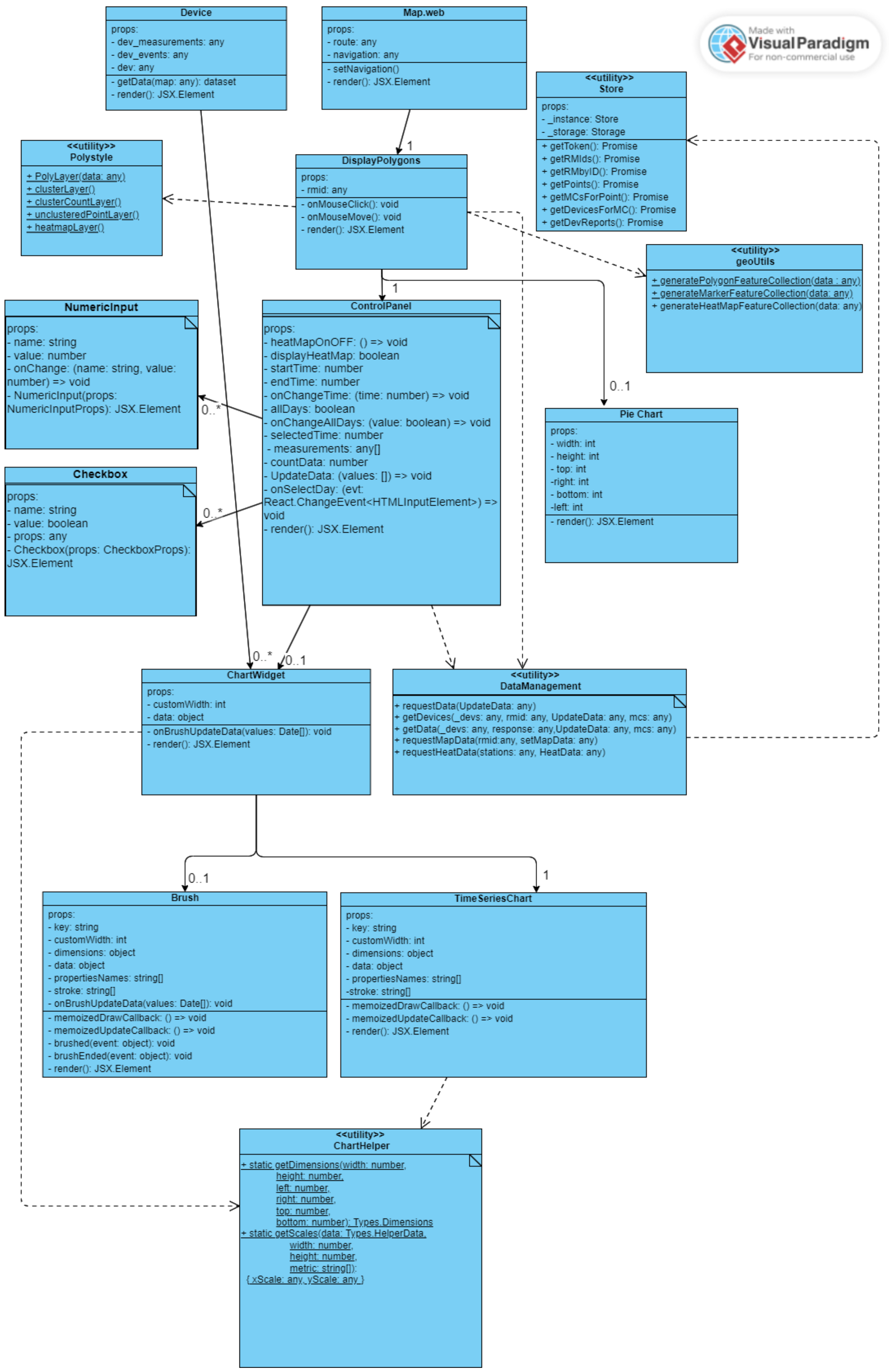

Device
props:
- dev_measurements: any
- dev_events: any
- dev: any
- getData(map: any): dataset
- render(): JSX.Element
Map.web
props:
- route: any
- navigation: any
- setNavigation()
- render(): JSX.Element
Made with Visual Paradigm For non-commercial use
<<utility>> Store
props:
- _instance: Store
- _storage: Storage
+ getToken(): Promise
+ getRMIds(): Promise
+ getRMbyID(): Promise
+ getPoints(): Promise
+ getMCsForPoint(): Promise
+ getDevicesForMC(): Promise
+ getDevReports(): Promise
<<utility>> Polystyle
+ PolyLayer(data: any)
+ clusterLayer()
+ clusterCountLayer()
+ unclusteredPointLayer()
+ heatmapLayer()
1
DisplayPolygons
props:
- rmid: any
- onMouseClick(): void
- onMouseMove(): void
- render(): JSX.Element
<<utility>> geoUtils
+ generatePolygonFeatureCollection(data : any)
+ generateMarkerFeatureCollection(data: any)
+ generateHeatMapFeatureCollection(data: any)
1
NumericInput
props:
- name: string
- value: number
- onChange: (name: string, value: number) => void
- NumericInput(props: NumericInputProps): JSX.Element
ControlPanel
props:
- heatMapOnOFF: () => void
- displayHeatMap: boolean
- startTime: number
- endTime: number
- onChangeTime: (time: number) => void
- allDays: boolean
- onChangeAllDays: (value: boolean) => void
- selectedTime: number
- measurements: any[]
- countData: number
- UpdateData: (values: []) => void
- onSelectDay: (evt: React.ChangeEvent<HTMLInputElement>) => void
- render(): JSX.Element
0..1
Pie Chart
props:
- width: int
- height: int
- top: int
-right: int
- bottom: int
-left: int
- render(): JSX.Element
0..*
Checkbox
props:
- name: string
- value: boolean
- props: any
- Checkbox(props: CheckboxProps): JSX.Element
0..*
0..*
0..1
ChartWidget
props:
- customWidth: int
- data: object
- onBrushUpdateData(values: Date[]): void
- render(): JSX.Element
<<utility>> DataManagement
+ requestData(UpdateData: any)
+ getDevices(_devs: any, rmid: any, UpdateData: any, mcs: any)
+ getData(_devs: any, response: any,UpdateData: any, mcs: any)
+ requestMapData(rmid:any, setMapData: any)
+ requestHeatData(stations: any, HeatData: any)
0..1
1
Brush
props:
- key: string
- customWidth: int
- dimensions: object
- data: object
- propertiesNames: string[]
- stroke: string[]
- onBrushUpdateData(values: Date[]): void
- memoizedDrawCallback: () => void
- memoizedUpdateCallback: () => void
- brushed(event: object): void
- brushEnded(event: object): void
- render(): JSX.Element
TimeSeriesChart
props:
- key: string
- customWidth: int
- dimensions: object
- data: object
- propertiesNames: string[]
-stroke: string[]
- memoizedDrawCallback: () => void
- memoizedUpdateCallback: () => void
- render(): JSX.Element
<<utility>> ChartHelper
+ static getDimensions(width: number, height: number, left: number, right: number, top: number, bottom: number): Types.Dimensions
+ static getScales(data: Types.HelperData, width: number, height: number, metric: string[]): {xScale: any, yScale: any }

Figure 18. Class Diagram of the system

Display Polygons:

This is the actual map of the system. The main task of this file is to provide a precise view of the user's properties with an interactive approach. To achieve that, it initiates an asynchronous data request through the "DataManagement" utility file, based on the route ID that was parsed as a prop from the "Map.web" component. Then, accompanied by the utility files "Polystyle" and "geoUtils" generates and renders polygons and markers throughout the map area. The displayed polygons represent the user's fields and crops, and the markers his stations. The map system ensures an ease of use and interactive experience providing various events that will trigger once the user engages with his property. More specifically, a plethora of mouse events are registered so that he can observe his property from different angles and perspectives using drag techniques with his mouse, gain insights through a popup for every displayed part that relating to his property by clicking on a polygon or marker area. Additionally, D3 charts are also distributed across the map and are also triggered by mouse events. More precisely a mini version of a time series chart emerges within the map's control panel, upon the user's initial click within a polygon area. Besides that, a pie chart materializes near the user's cursor, as he hovers through a polygon or marker area. The D3 integration in the map's system serves as an elevated visualization that aims to facilitate the user's experience and also stands as a prompt to explore the system's sophisticated charts for a more in-depth analysis of his properties' condition.

Regarding the relationships of "DisplayPolygon" with the rest components, they consist of a variation of unidirectional and dependency interconnections. More precisely, the utility files that are used to request data, structure, and display the polygons and markers are linked with a dependency. The "ControlPanel" has a unidirectional association with a multiplicity of one since it is accompanied by the map, thus it renders once. Lastly, the "Pie chart" also has a unidirectional association but in this case, it exhibits a "0..1" multiplicity since it will be only rendered if the user decides to hover in a polygon or marker area.

Polystyle:

This utility file is tasked with providing the layers that are required by the map. Essentially this file is in charge of preparing the polygons, markers, and heatmap layers. In the case of polygon layers, it separates the fields from the crops, based on their ID, it ensures that each crop and each field have a unique color and it finally fills that color. As for the marker layers, they consist of three different layers: the "clusterLayer", the "clusterCountLayer", and the "unclusteredPointLayer". Prior to analyzing those layers, it is noteworthy that in order to establish the marker clustering, inside the source tag that the map component utilizes to render the markers, it must be added that the cluster is true. This automatically generates Mapbox's special property called "point_count" which is designed for marker clustering. Then, Clusters are calculated dynamically based on the zoom level of the map and the proximity of markers, assigned to "clusterMaxZoom" and "clusterRadius", accordingly. The point_count attribute is assigned to a cluster feature when markers are grouped together. This attribute indicates the number of markers in the cluster. Continuing with the layers, the clusterLayer is the intermediary for clustering, and the clusterCountLayer displays the "point_count" as a text inside the cluster. Both layers filter the source to include only features with the 'point_count' property and use that value to indicate the cluster radius and color. The unclusteredPointLayer is the actual marker layer that is tasked with displaying the markers when the “clusterMaxZoom” or “clusterRadius” is out of bounds, filtering to avoid any “point_count” properties and fetching an icon image with a predetermined color. Finally, the “heatmapLayer”, as its name indicates, generates, and paints the heat map values based on the frequency of a given property and the zoom level of the map.

Geo Utils:

This is the file that generates every data structure that is required by the map be it polygons, markers, or even the heat map. It is also a utility file and thus it exhibit’s a dependency relationship with “DisplayPolygons” component. Each of its functions accept raw data as an argument and they return a GeoJSON object. GeoJSON is an independently available standard based on JavaScript Object Notation (JSON), for the interchange of geographical data with a format precisely intended to incorporate plain geographic elements as well as their nonspatial properties [56]. Mapbox web services and APIs deliver geospatial data in the form of GeoJSON. On the substance, MapLibre GL JS requires a GeoJSON object that would be recognized through the declaration of type feature or feature collection if the structure contains multiple features together. Apart from the specification of type, it requires a geometry indication which provides vital information about the spatial or shape of the geographic feature being represented. Moreover, it requires the declaration of the feature’s geometry type which can be a point, line, polygon, etc., and the coordinates which correspond to it. Finally, a feature offers an additional field, the properties. Properties are an optional feature that the programmer can utilize for conveying additional information or attributes related to the geographic feature. Properties’ supplementary information can contribute to map rendering, interaction, and analysis.

Store:

This is maybe the most crucial utility component of the entire application. It was created as a tool to form all the necessary transactions with the server, via socket calls. It is used to fetch authentication tokens, as well as to facilitate requests, storage, or deletion actions for the user's credentials and his associated data (properties). For this thesis implementation purposes, it primarily utilizes the data requests that relate to the user’s properties. More precisely, the store follows the fundamental data flow of the system which begins with a resource manager that leads to its points which at the current time contain one microcontroller per point. The microcontrollers manage different types of sensors and provide reports of each sensor’s measurements. The store offers separate functions for all the above stages giving the freedom to access multiple parts of the data for different purposes. Finally, since it’s a utility file it is noteworthy to indicate that it will be linked with a dependency relationship.

Data Management:

“DataManagement” is one more utility component, developed to request and manage data. It is mainly used in “DisplayPolygons” and “ControlPanel” components, thus forming a dependency relation between them, and its responsibility is to request data from the “Store” component and convert it into employable data for these components. It can be separated into three main functions that correspond to different purposes. The first part consists of three main functions, “requestData()”, “getDevices()”, and “getData()”. The first two of the aforementioned functions navigate through the store’s functions, to access their data pattern of resource manager, points, and microcontrollers to derive the devices reports. These reports are then fetched into the “getData()” in order to get sorted out and prepared for visualization purposes. The second part of the component refers to the generation of the initial heat map dataset using the “requestHeatData()” function. This function utilizes the three functions of the first part but also provides some extra features that conclusively generate the heat map dataset. Finally, the last key function is called “requestMapData()” and as its name indicates, it also uses the store functions to return a dataset that contains valuable information about the user’s fields, crops, and stations that in DisplayPolygon’s use case translates as polygons and markers.

Pie chart:

This file renders a pie chart that is currently being displayed on the “DisplayPolygons” component. It is implemented to act as one of the map’s interactive features, as it only gets displayed when the user hovers over a polygon and becomes invisible when his cursor leaves a polygon area. It is linked with a unidirectional association that exhibits a “0..1” cardinality since it depends on the user’s interaction with the polygons which might never happen, thus avoiding the rendering of a pie chart.

Control Panel:

This is practically the complementary companion of the map component. It is created as a means to enhance the map’s interactivity and satisfy the user’s needs, displaying numerous variables that will customize the map on his behalf. It is associated with a unidirectional relationship with the map and given that it initially renders once, along with the map it has a cardinality of “1”. It is orchestrated to enable and disable the heatmap feature as well as general map features such as the scroll zoom, the double click zoom, the touch zoom rotate, etc. It also currently provides four numeric input boxes for the user to determine his preferred min and max zoom and pitch, respectively. Lastly, it also provides a glimpse of this app’s second visualization technique with the D3 library, displaying a mini chart at the bottom of the ControlPanel’s domain.

Numeric Input and Checkbox:

These two components provide the checkboxes and the numeric inputs that are presented on “ControPanel”. They are developed inside the “ControlPanel’s” file but since they render a JSX element, they act as individual components. They have a unidirectional association with the “ControlPanel” with a cardinality of “0..*” meaning it can be either zero or more. This is based on the fact that “ControlPanel” renders them based on a property that it receives from the map component and since it can't possibly know the precise number of checkboxes and numeric inputs, it exhibits this cardinality.

Device:

This file is dedicated to the second visualization technique the presentation of charts powered by D3. This file is the result of the user navigating through the home page’s recourse manager, to further navigating through his points and accessing a station that has a functional microcontroller. This routing triggers the devices and the collection of their reports. These data are fetched to the “Device” component that renders chart diagrams based on the exact number of the received devices.

Chart Widget:

The “ChartWidget” component, in accordance with its terminology, functions as a distinct widget constructed particularly for the presentation of charts. It gets rendered by the “Device” and the “ControlPanel” components, connected with a unidirectional association. As for their multiplicity the “Device” exhibits a “0..*” cardinality, given that it receives the actual devices as props which may vary from zero to many. The “ControlPanel” on the other hand has a cardinality of “0..1” since it depends on user interactivity to render. Furthermore, this file utilizes the “ChartHelper” to prepare the dimensions that will be parsed as a prop to the actual charts which are the “TimeSeriesChart” and “Brush” components. In essence, ChartWidget has the roles of rendering and mediating these components.

Time Series Chart:

This is the main chart component. It is connected with a unidirectional association with the “ChartWidget” and has a cardinality of “1”. This component also implements the “ChartHelper” component for the construction of its x and y scales. It is a time series chart that presents as many lines as the types of datasets it gets as a prop. Furthermore, it provides mouse events, as a breach for an in-depth analysis by specifying precise information about each data point that is closest to the current mouse coordinates.

Brush:

This is the supplementary chart that is positioned right below the main chart. “Brush” also has a unidirectional association but this one exhibits a cardinality of “0..1” as it is excluded in the mini chart use case in “ControlPanel”. The main purpose of this component is to create a brush effect that will allow the user to zoom in and out of the main chart, giving more detailed information about a specific area of values or timelines. This component holds the whole dataset that is also presented to the user at all moments. The user can interact with that chart and create a brush effect, then the component triggers a "ChartWidget" function that is parsed as a prop, which results in fetching a new dataset for the "TimeSeriesChart", thus creating the brushing event.

Chart Helper:

"ChartHelper" serves as the last utility file of this implementation. This component also demonstrates a dependency relationship with the components that integrate it. It consists of two main static functions, the "getDimensions()" and "getScales()". The first prepares the dimensions of the charts and the second generates the scales of each chart's axis.

3.3 Functional Requirements

A Functional Requirement (FR) is an overview of the services that the software must deliver. It illustrates a software system or one of its fundamental components. In this context, a function comprises the software system's inputs, operational behavior, and outputs. This function might entail calculations, data processing, business operations, user interactions, or any other specialized functionality that defines the system's expected operational scope. A Functional Requirement in software engineering and systems engineering ranges from broad abstract pronouncements of the sender's requirements to minutely detailed mathematical descriptions of functional prerequisites. These functional software requirements serve to encapsulate the system's intended behavior [57].

- **Device**
  This is one of the two visualization methods of the application, the presentation of charts. The user navigates through the home page, accesses his recourse manager, and then his points to arrive at the presentation of the devices.

Section/Requirement ID	Requirement Definition
FR1.0	The system shall allow users to navigate through numerous amounts of charts based on the available devices, each providing an in-depth analysis of their properties' conditions

Table 1. Device requirements Group 1

- **Chart Widget**
  This widget is responsible for generating every time series chart that this system requires. It is rendered either by the "Control Panel" (map) or by the "Device"

components. Reaching from the "Device" component the user will get a sophisticated time series chart with interactive capabilities dedicated to exhibit the measurements of his stations. Conversely, if the chart is accessed through interactions with the map component, a condensed mini chart is provided, serving as a prompt for users to explore the detailed charts available in the "Device" component.

Section/Requirement ID	**Requirement Definition**
FR2.0	The system shall render either a basic chart or an advanced features chart based on the user's navigation through the app.
FR2.1	The system shall enhance user interactivity by incorporating a function dedicated to the advanced features chart, serving as an intermediary that facilitates communication between the rendered charts.

Table 2. Chart Widget requirements Group 2

- **Time Series Chart**
  This is the main part of the chart presentation. It is a time series chart that is designed to handle large datasets and present the values over time.

Section/Requirement ID	**Requirement Definition**
FR3.0	The system shall be able to present to the user the detailed reports that are generated by the station that is positioned to his fields and crops.
FR3.1	The system shall display each time series chart with a horizontal x-axis representing time progression, and a vertical y-axis signifying the corresponding values. The number of lines will be indicated based on those values IDs.
FR3.2	The system shall capture the user's cursor inside the chart area to provide mouse events.

FR3.2.1	The system shall allow the user to view each data point of the chart with detailed information, regarding the position of that specific point and its corresponding ID, value, and date.

Table 3. Time Series Chart requirements Group 3

- **Brush**
  The supplementary chart that is positioned right beneath the time series chart. It is only displayed on the "Device" component. It is the replica of the time series chart, with the exception of keeping only the x-axis, which is converted into an area chart format.

Section/Requirement ID	**Requirement Definition**
FR4.0	The system shall showcase to the user a condensed version of the time series chart that is transformed to an area chart.
FR4.1	The system shall capture the user's mouse coordinates inside it to initiate mouse events.
FR4.1.1	The system shall provide the user with various mouse events, in order to enhance the brush events which is practically a transparent layer that is positioned in the chart and can cover its whole extent.
FR4.1.2	The system shall create a new dataset based on the user's brush extension, capturing the timelines inside that domain, and fetching it to the time series chart as a new dataset to produce a zoom-in/out effect.

Table 4. Brush requirements Group 4

- **Map**
  The "Map" component is the second visualization technique that the user can access either by the drawer menu which is always displayed on the right side of the app or by routing through the home component's resource manager which will display a map icon on the navigation bar.

Section/Requirement ID	**Requirement Definition**

FR5.0	The map system shall grant the user full accessibility to the map, enabling him to drag, rotate, zoom in and out, etc., offering him an interactive experience.
FR5.0.1	The map system will capture the user's interaction with the left side icons which represent the Geolocate, Fullscreen, and Navigation controls of the map which will either determine the user's current location and center the map on it, toggle the map between Fullscreen and normal view, zooming in and out of the map or reset the map's rotation to north.
FR5.1	The map system will render and display all the available data of the user's properties seamlessly translating them into distinct polygons and markers spread across the geographical representation, each carrying mouse events.
FR5.1.1	The map system shall capture user's mouse hover on polygon areas to demonstrate a D3 pie chart.
FR5.1.2	The system shall create a marker clustering based on the user's zoom level and the distance between the markers. Each cluster will have a count that will indicate the exact number of markers it holds inside.
FR5.1.2.1	The system shall allow the user to click on the clusters which will result in zooming to that specific cluster, unfolding smaller clusters, and/or revealing the markers that it used to contain.
FR5.1.3	The map system shall capture all user's clicks inside a polygon or a marker area, revealing precise information about their titles.
FR5.1.3.1	The map system shall also present a mini chart on the "Control Panel" if it isn't already displayed.

Table 5. Map requirements Group 5

- **Control Panel**
  The "ControlPanel" is the component that renders next to the map and will allow user to have a more advanced experience with the map component.

Section/Requirement ID	Requirement Definition
FR6.0	The system shall allow the user to enable or disable the heatmap.
FR6.1	The system shall allow the user to customize the map component based on his needs.
FR6.2	The system will allow the user to close the mini chart, to save additional space for the “Control Panel”.

Table 6. Control Panel requirements Group 6

3.4 Non-Functional Requirements

A Non-Functional Requirement (NFR) defines the qualitative characteristics of a software system [58]. Infractions with non-functional requirements can result in systems that fall short of satisfying user requirements. Non-functional requirements are used in Software Engineering to impose constraints and limitations on the system's architecture, echoing across numerous agile backlogs. NFRs act as evaluation benchmarks for the software system, encompassing characteristics that are vital to the system’s longevity such as:

1) **Scalability:** Scalability requirements include how a system's capacity ought to be expanded while retaining optimal performance. This includes accommodating a larger user base, managing larger data quantities, and processing a higher volume of operations. Scalability concerns apply to both the hardware and software realms [58].
2) **Maintainability:** The extent to which a system can be easily adjusted and improved over time is referred to as maintainability. It captures the system's ability to accept updates, changes, and alterations while limiting complexity and interruption. A highly maintainable system ensures that modifications to the software might be made without resulting in significant difficulties or disruption. This facilitates the addition of new features, bug repairs, and upgrades. This non-functional criterion helps ensure the software's longevity by enabling development teams to adapt swiftly to ever-evolving user needs and technology developments [58].
3) **Availability:** Availability is ascertained through the period of time that the system's features and services are accessible and functional, supporting all necessary processes. This measure is directly influenced by the time slots for planned maintenance.

Developing plans for limiting the effects of maintenance interventions is of utmost importance. The development team is tasked with identifying the crucial system components that demand constant accessibility while formulating the availability requirements [58].

4) **Usability:** The usability requirements primarily address the system's degree of user-friendliness. These requirements define how much the system's interface and interactions contribute to a smooth and gratifying user experience. To be more precise, the interface should be developed to ensure that users can effortlessly navigate through its components. It should have a clear purpose for its features so that the user will be able to comprehend them. Finally, it should try to predict and exceed the user’s expectations to guarantee an enhanced overall experience [58].

4 Implementation

This chapter delves into the comprehensive implementation of this application. This section methodically dissects each individual component, showing the dense tapestry of functionality and design that serves as the foundation for the application's architecture. The analysis reveals an arrangement of components, each customized to meet the specific project requirements. A visual excursion through the user interface is presented, accompanied by illustrations, demonstrating the harmonic blending of advanced capabilities that enable users to easily engage with geospatial data and visualization tools. This chapter serves as a user manual, revealing the technical foundations that provide the application its performance, responsiveness, and overall user involvement. Since this thesis is based on an already existing application and aims to enrich it through the integration of big data visualization, part of this implementation must be consistent with the ideology and architecture of the application.

4.1 Log-in and Sign-up

The login page (see Figure 19) is the first page that the user will see upon running this application. He needs to have an account to access this application and if he doesn’t, he needs to create one by clicking the sign-up button that is placed below. Given that this chapter illustrates all possible outcomes the case of a new user will be assumed. After the user clicks on the sign-up button, he will be redirected to the signup page (see Figure 20). From there he can fill in his credentials, which are his email, username, and password. After filling these and clicking the sign up button he will be redirected back to the login page. In order to access the rest of the application he will need to remember those credentials that he filled in and accurately type them on the login form. The login page requires a username and a password. After filling in his credentials and clicking the login button, those inputs will be fetched to the server in order to authenticate them. In case the user fails to match his credentials, a popup error message will be displayed informing him that either the username or the password is wrong. However, if he is authenticated, the system will request his token from the server and fetch it to the socket, for further use.

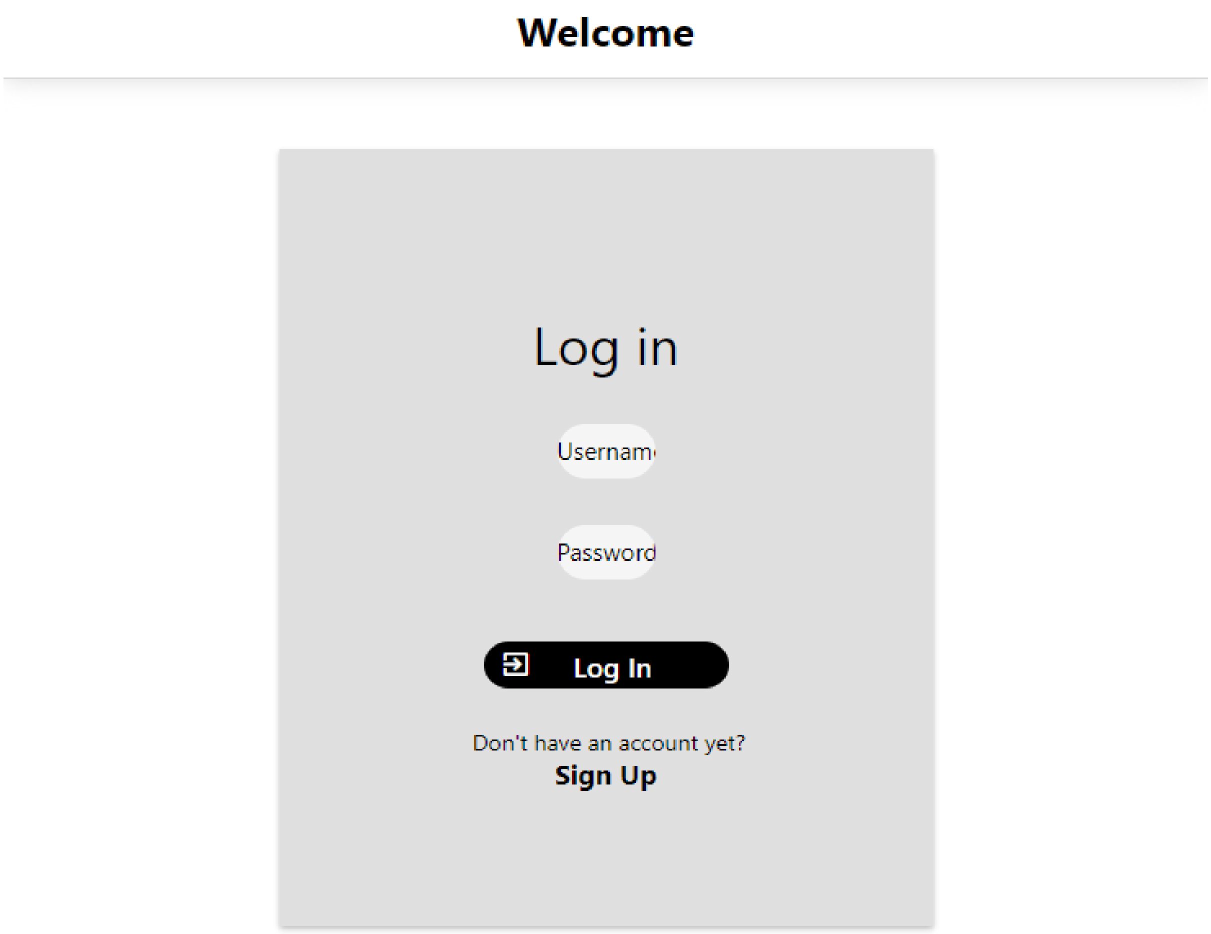


Figure 19. Log-in page

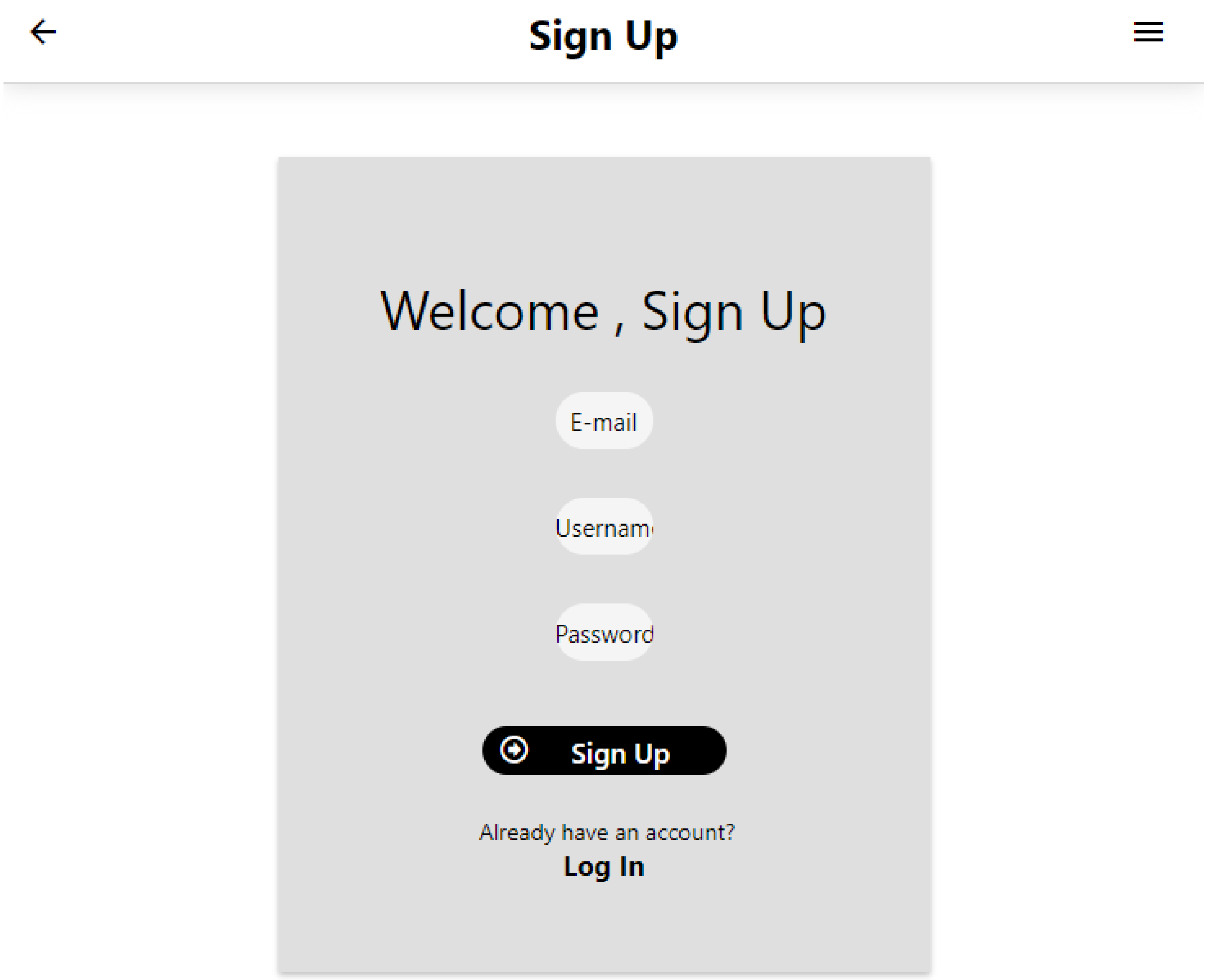


Figure 20. Sign-Up page

4.2 Home Page

The home page (see Figure 21) is the redirected page that the user will see after successfully logging in. It includes a resource manager component displaying its ID, along with an edit and delete buttons that are located at the very bottom of the component. At the very top of the page, a navigation bar has been placed with a drawer menu (see Figure 22) sitting fixed on the right side of the screen. The user can either navigate through the recourse manager or select a different page through the drawer menu. Since the navigation of the home page will result in one of the two visualization techniques of this thesis, priority will be given to that particular routing path.

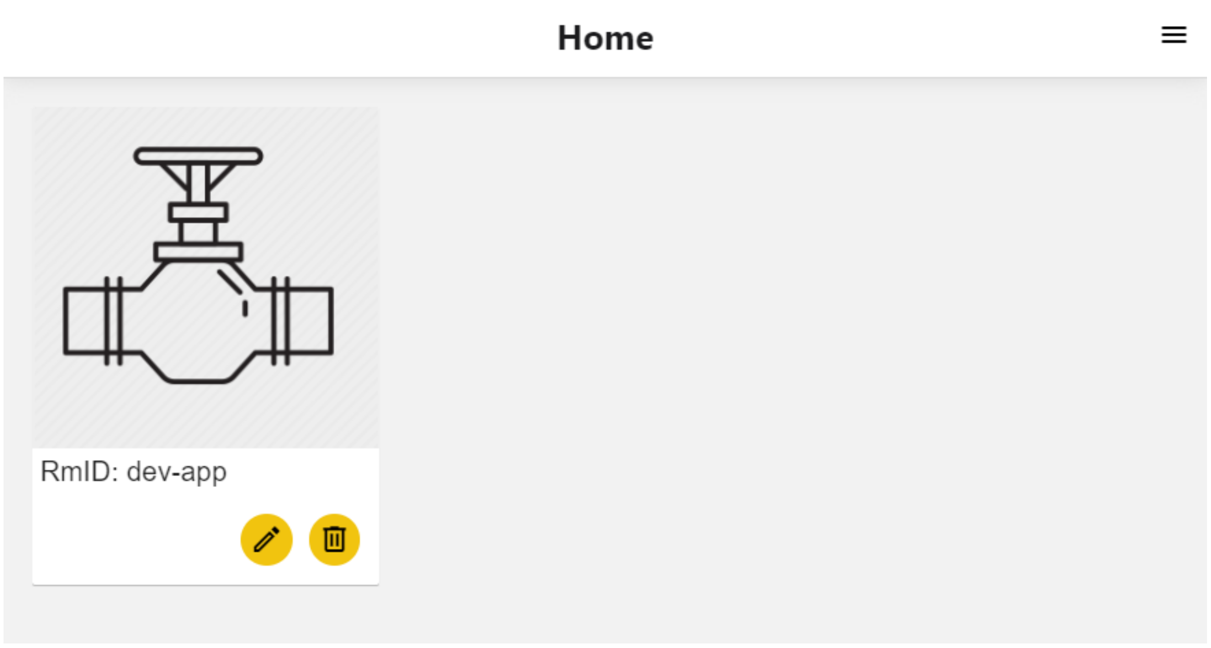


Figure 21. Home Page

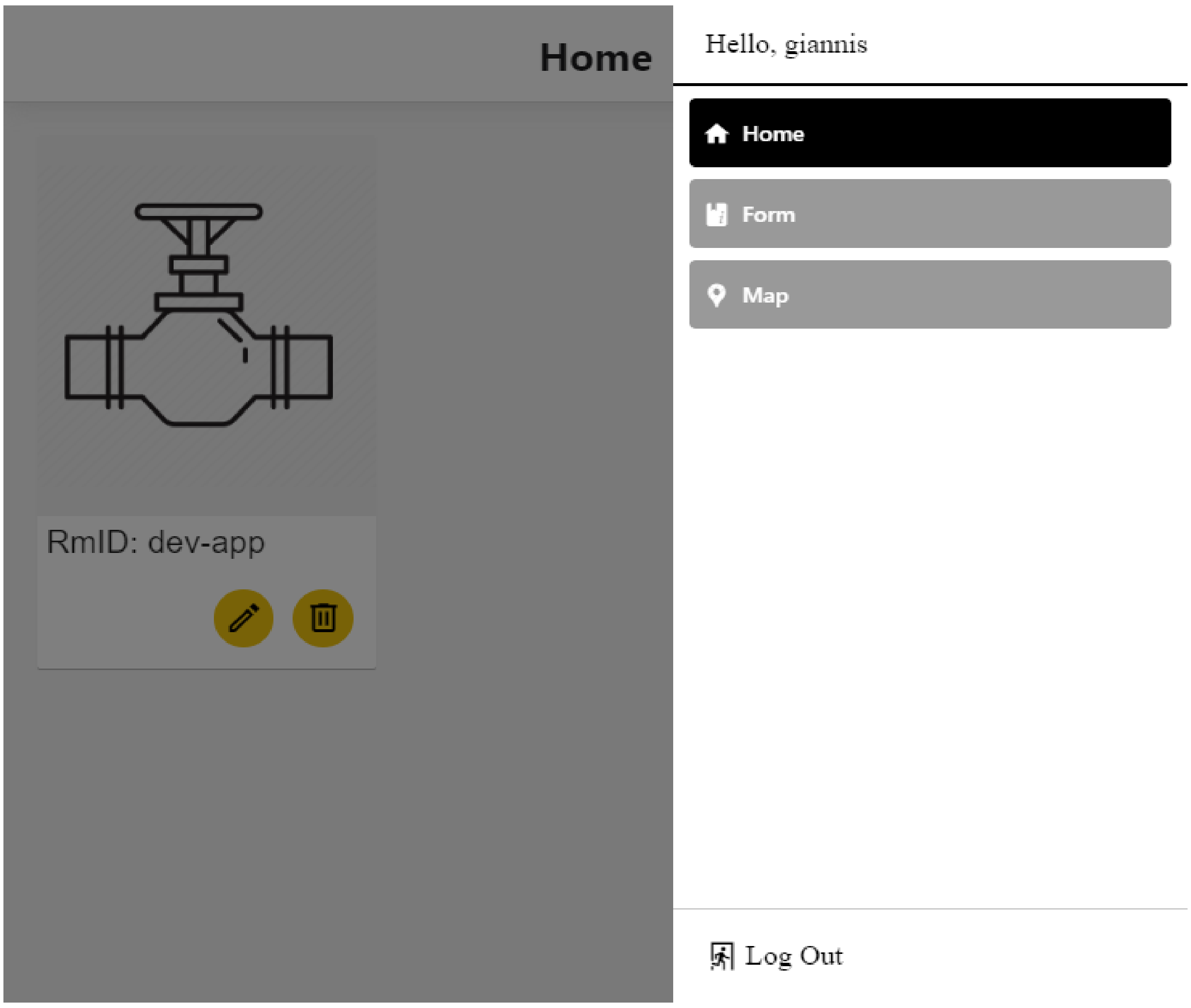
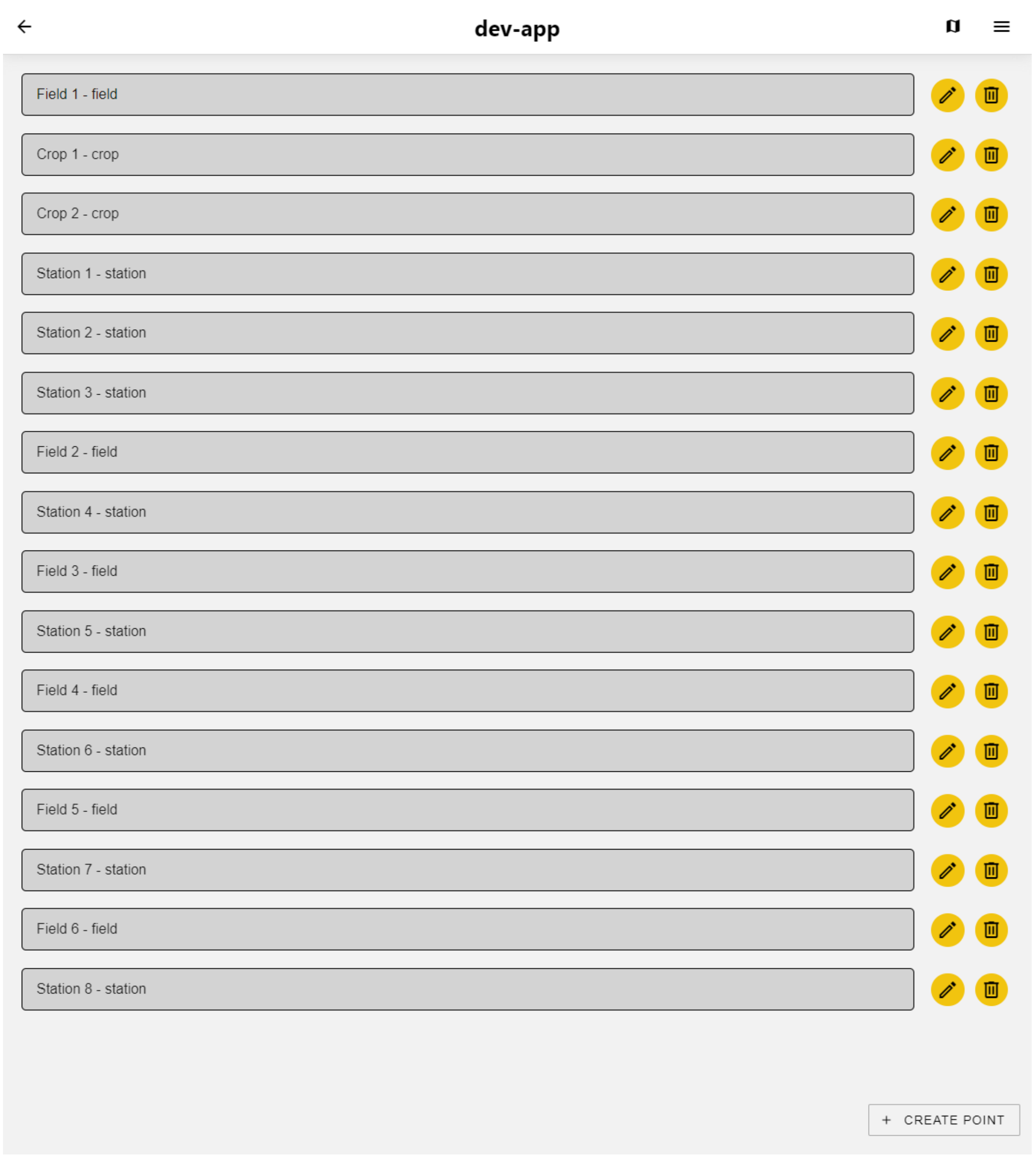


Figure 22. Drawer Menu

4.2.1 Home Page (Points)

After the user clicks the title of the presented resource manager the home page will change and display all its available points (see Figure 23). The user can access its fields, crops, and stations as well as editing or deleting them. He is also having the opportunity of creating new points by clicking the “create point” button located at the end of the page. Furthermore, a map icon is also displayed on the navigation bar, which provides a second method of accessing the map.


dev-app

Field 1 - field
Crop 1 - crop
Crop 2 - crop
Station 1 - station
Station 2 - station
Station 3 - station
Field 2 - field
Station 4 - station
Field 3 - field
Station 5 - station
Field 4 - field
Station 6 - station
Field 5 - field
Station 7 - station
Field 6 - field
Station 8 - station

Figure 23. Home Page (Points)

4.2.1.1 Home Page (Points-Stations)

This page (see Figure 24) is displayed when the user decides to navigate through one of his available stations. Presently, each station results in one microcontroller which administers multiple devices that contain sensors, and the conducted reports of these devices are obtained by the system to be visualized and analyzed. When the user clicks on that microcontroller, the system will generate chart components based on the number of the devices under that specific microcontroller (see Figures 25,26).

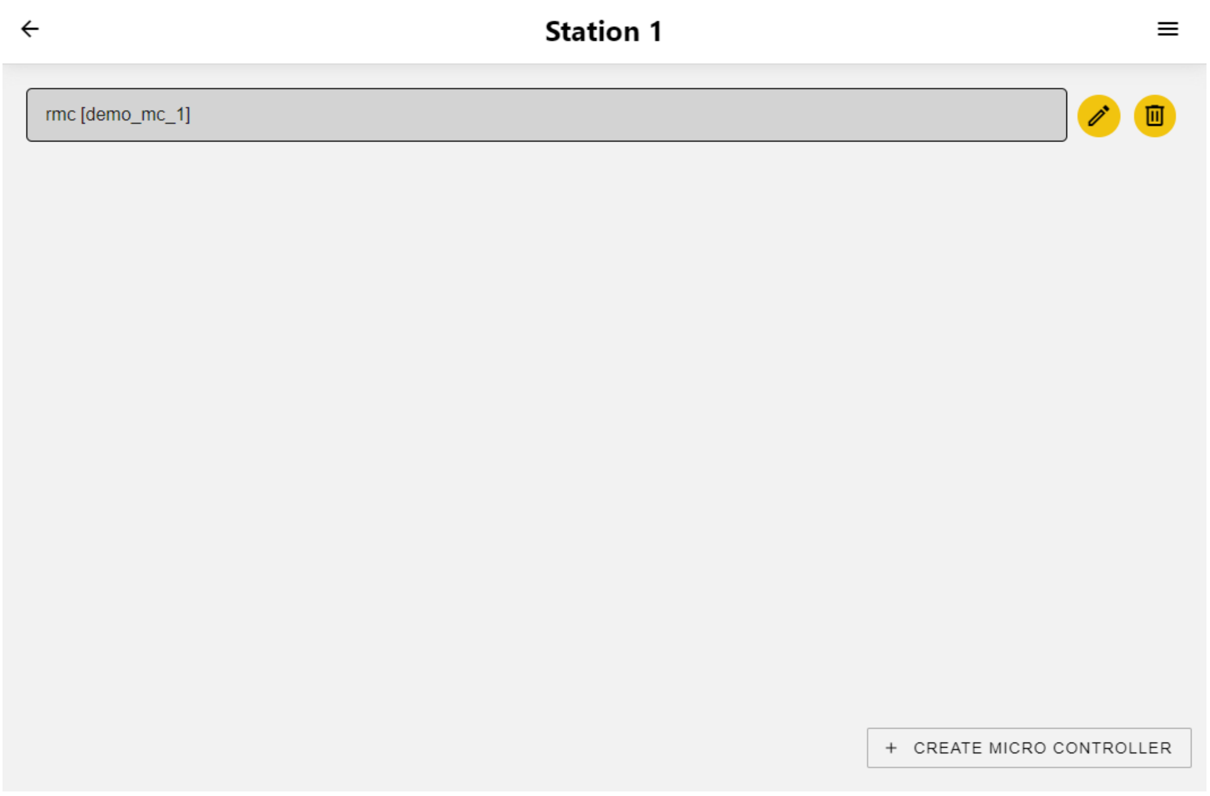


Figure 24. Home Page (Points/Station 1)

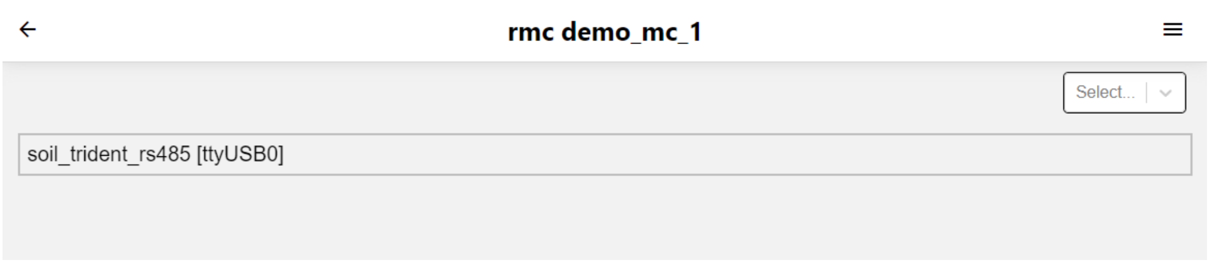


Figure 25. Example Screen of User 1

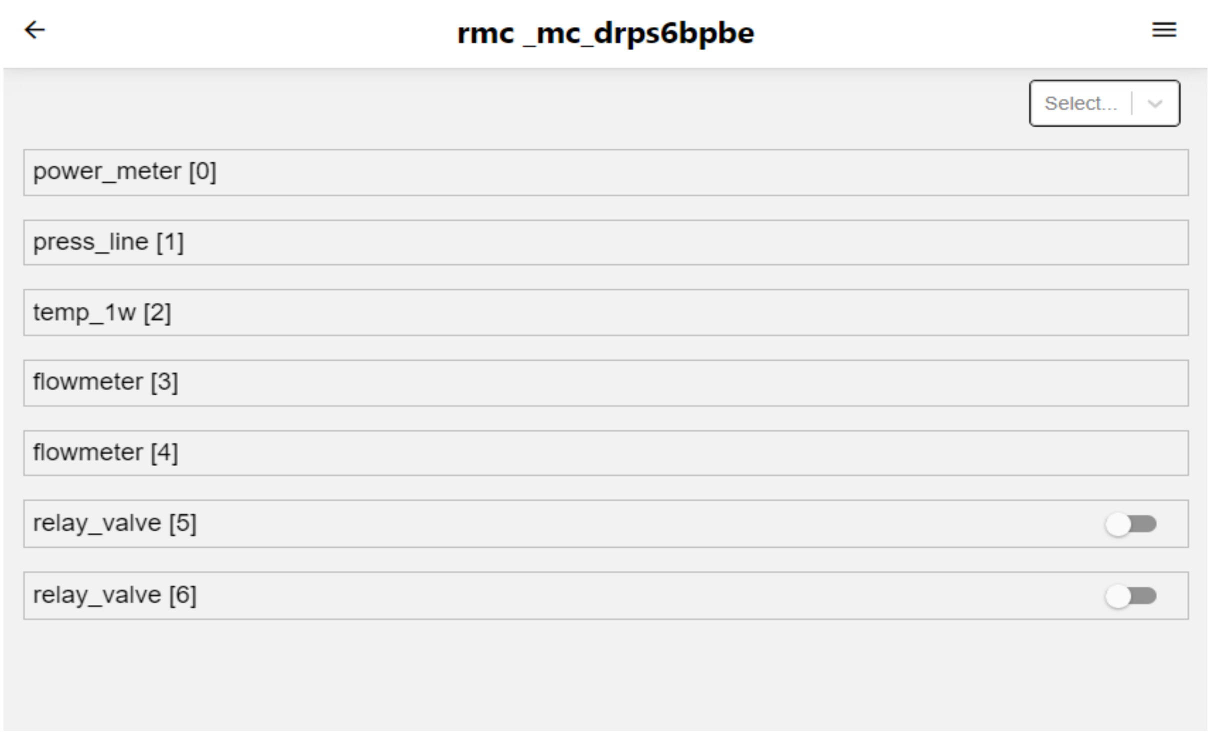
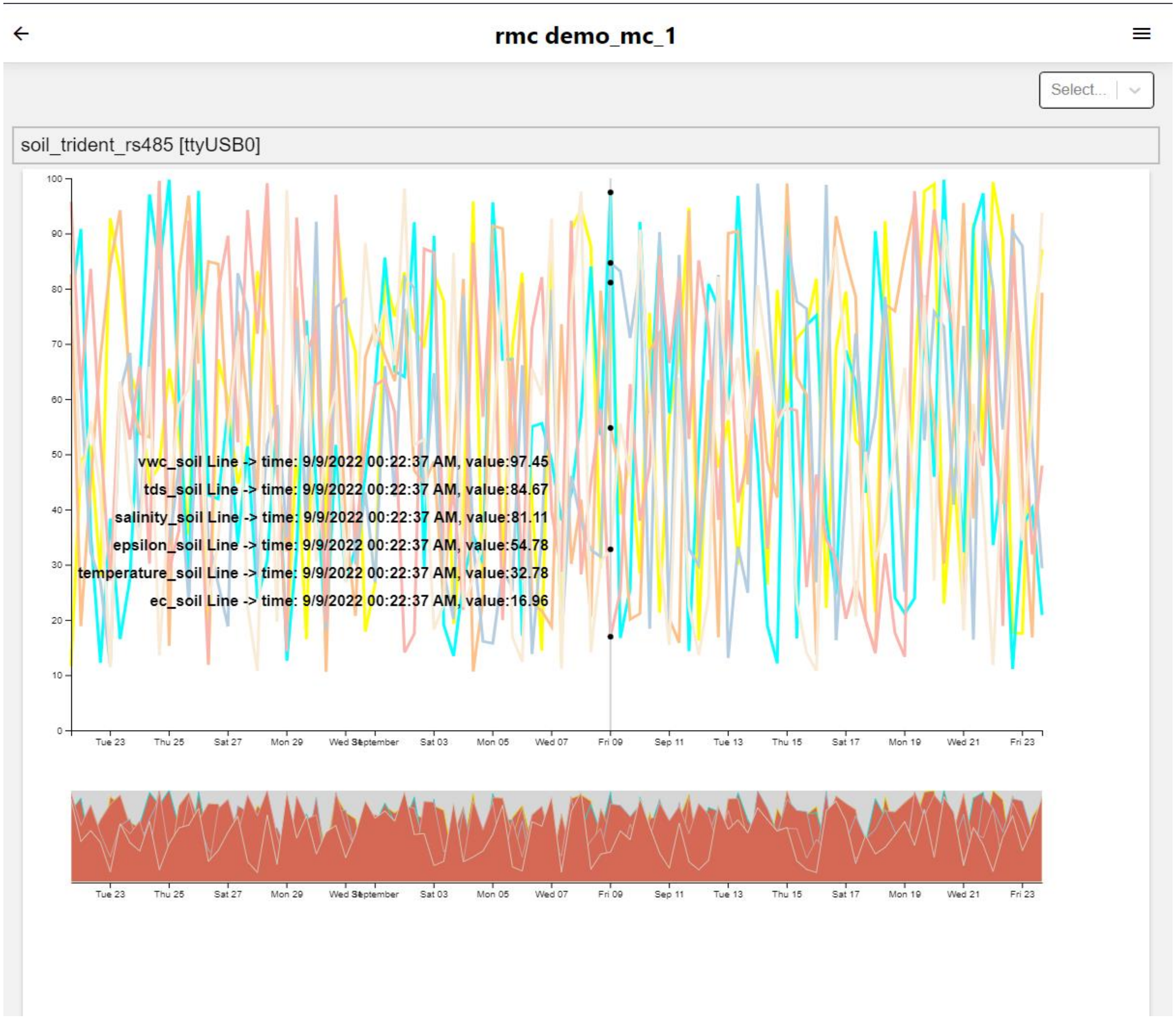


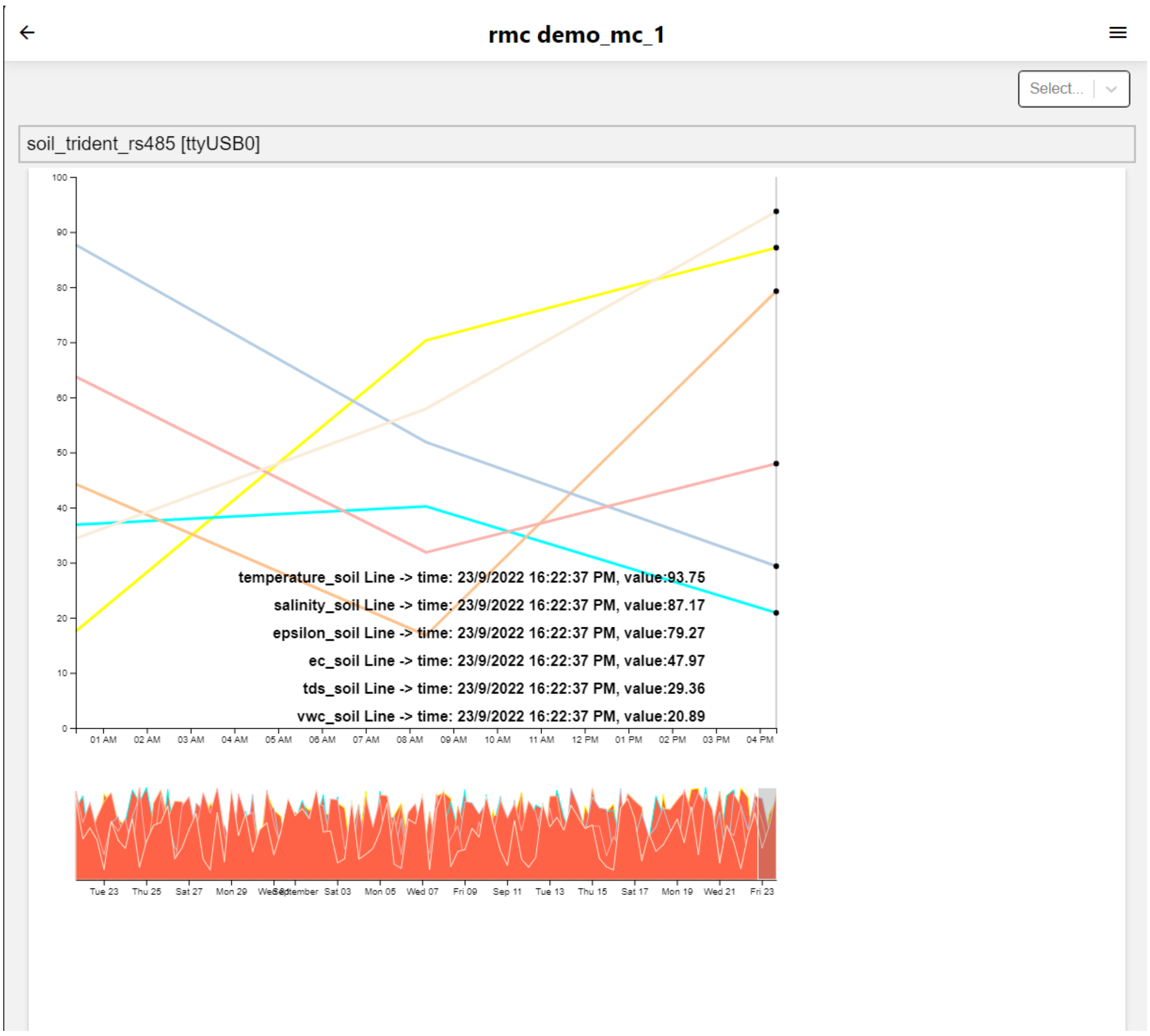
Figure 26. Example Screen of User 2

Each of these components represent a chart that visualizes the devices reports with precision and interactivity. Moreover, the user can click any of the available components to reveal or suppress them allowing him to establish his preferable chart combination. Every rendered chart has sophisticated interactive features to elevate his overall experience. The main chart tracks his mouse coordinates over its SVG area, and It delivers accurate information about that mouse coordinate in the form of a D3 text, including the name, value, and date depending on the nearest available data point of each line (see Figure 27). When the user interacts with the “Brush” which is the mini chart that is rendered with the main chart, it also initiates a series of events. To be more precise the user’s cursor is changed, prompting him to use his mouse to click and drag, thus selecting a specific location within the chart. As an output, this designated area can be used for a variety of purposes that affect the main chart including zooming in on a specific time range or highlighting a portion of data (see Figure 28). Finally, one last method to filter and specify a particular data range is through the date’s list container that is located in upper right side of the screen, directly beneath the navigation bar and the drawer menu. The user can choose between a predetermined variety of dates and filter both the brush and the time series charts according to his preferences (see Figure 29).



vwc_soil Line -> time: 9/9/2022 00:22:37 AM, value:97.45
tds_soil Line -> time: 9/9/2022 00:22:37 AM, value:84.67
salinity_soil Line -> time: 9/9/2022 00:22:37 AM, value:81.11
epsilon_soil Line -> time: 9/9/2022 00:22:37 AM, value:54.78
temperature_soil Line -> time: 9/9/2022 00:22:37 AM, value:32.78
ec_soil Line -> time: 9/9/2022 00:22:37 AM, value:16.96

Figure 27. Time Series Chart


rmc demo_mc_1

soil_trident_rs485 [ttyUSB0]

temperature_soil Line -> time: 23/9/2022 16:22:37 PM, value:93.75
salinity_soil Line -> time: 23/9/2022 16:22:37 PM, value:87.17
epsilon_soil Line -> time: 23/9/2022 16:22:37 PM, value:79.27
ec_soil Line -> time: 23/9/2022 16:22:37 PM, value:47.97
tds_soil Line -> time: 23/9/2022 16:22:37 PM, value:29.36
vwc_soil Line -> time: 23/9/2022 16:22:37 PM, value:20.89

Figure 28. Time Series Chart with Zoom

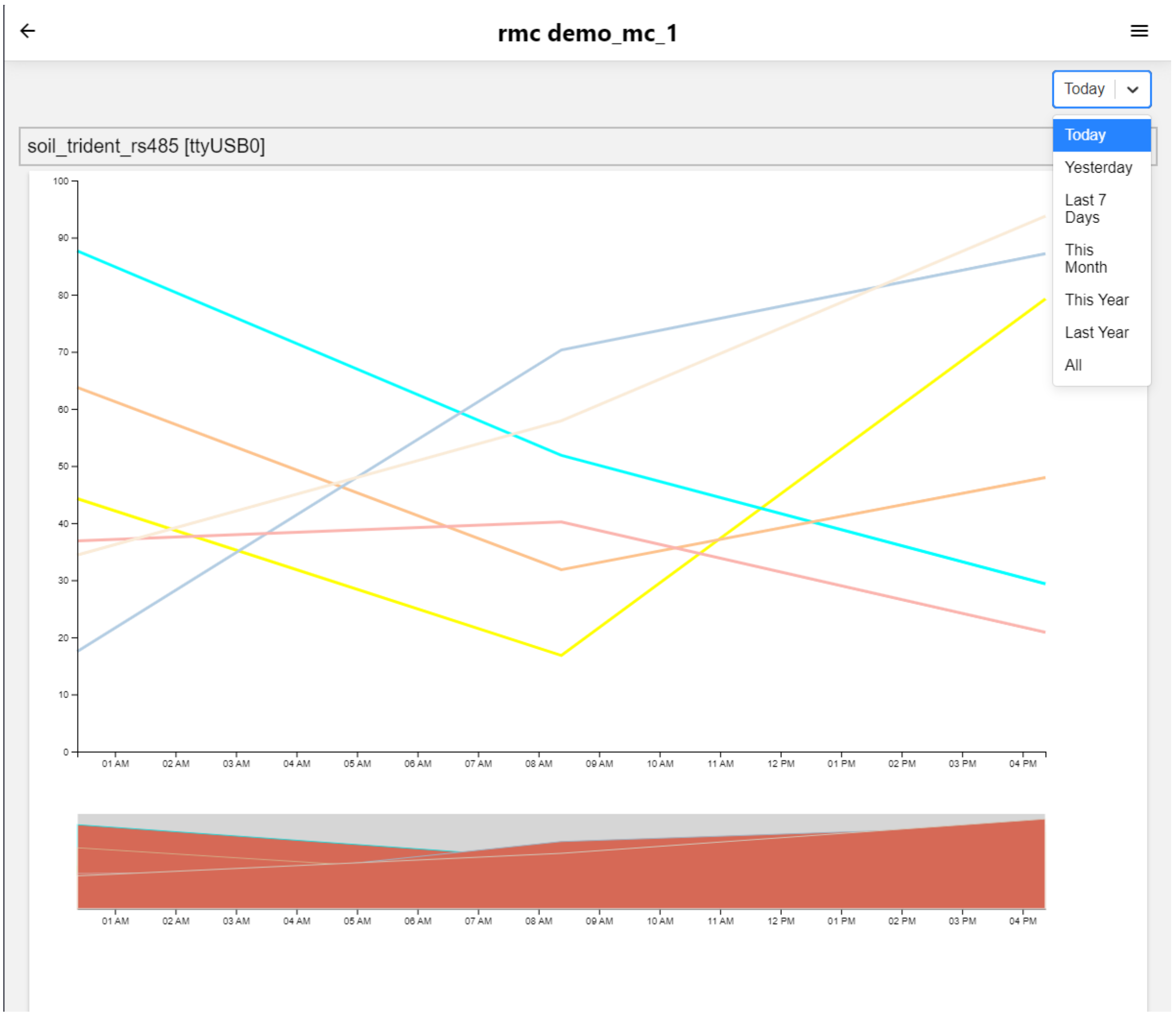
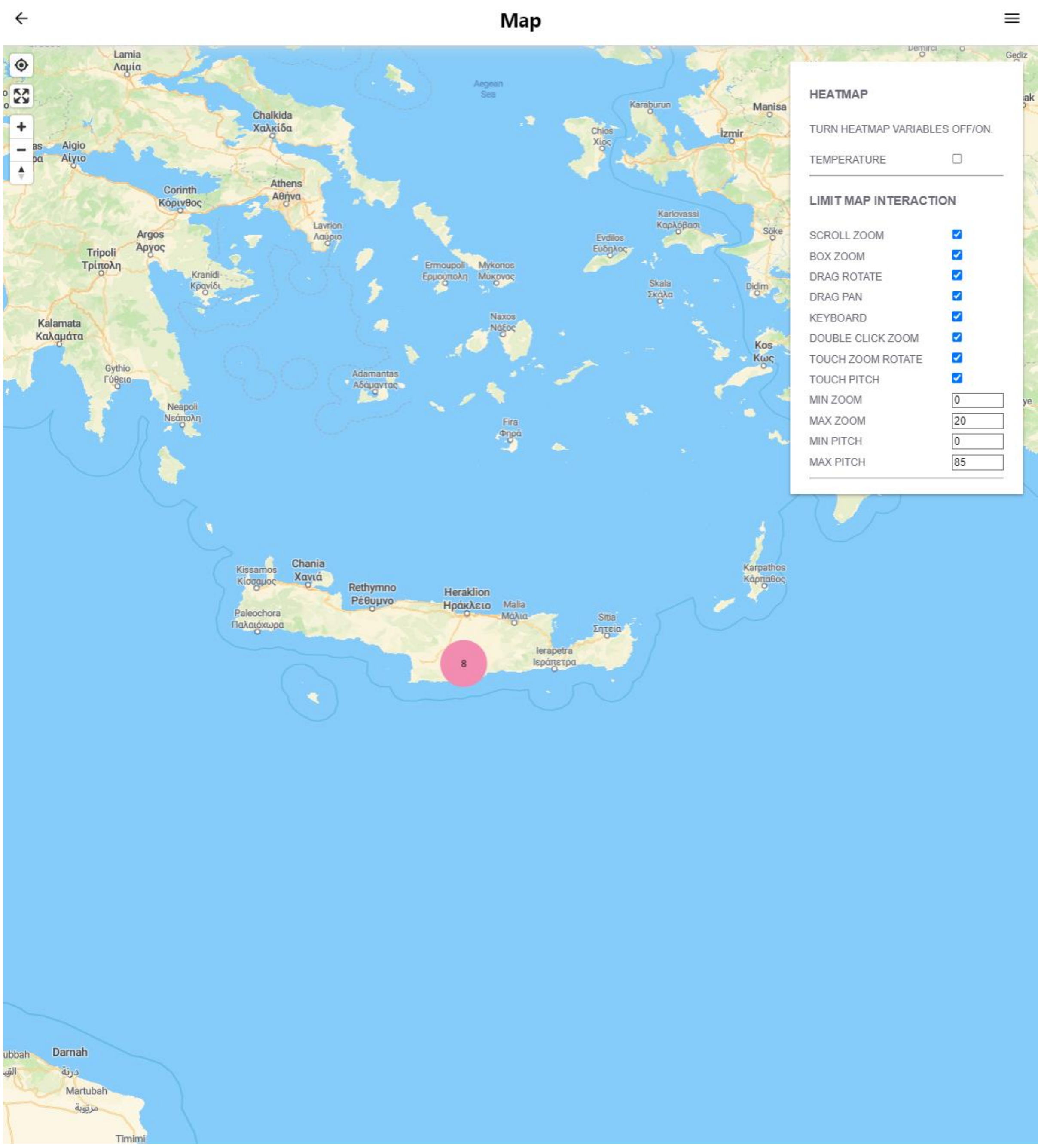


Figure 29. Time Series and Brush charts filter by today's data

4.3 Map

This is the map of the system (see Figure 30) and it stands as the second visualization method of this application. Right from the start the user can interact with the map with various mouse events that will enable him to drag, rotate, zoom in and out, and so on, providing an interactive experience. Alongside the map is positioned the control-panel. When the user interacts with it, the available variable that he enables, disables, increases, or decreases, are directly connected with the map variables, thus providing him a customizable opportunity. In consideration of performance and elegant issues all markers are able to perform a cluster effect. If the user clicks in those clusters or zooms in, then those clusters will begin to dissemble, revealing smaller clusters and/or the markers that it used to contain (see Figure 31). Nevertheless, if he decides to zoom out, those previous clusters will begin to assemble. As the user hovers his mouse over a visualized geospatial data, a pie chart will spawn near his mouse coordinates, indicating that this is part of his property. If the user clicks on that designated area, be it a polygon or marker, it will display a popup message, entailing information about that specific feature. Additionally, a mini time series chart will also be displayed in the "ControlPanel" along with an x button, should he want to remove it (see Figure 31).



HEATMAP

TURN HEATMAP VARIABLES OFF/ON.

TEMPERATURE	☐

LIMIT MAP INTERACTION

SCROLL ZOOM	✓
BOX ZOOM	✓
DRAG ROTATE	✓
DRAG PAN	✓
KEYBOARD	✓
DOUBLE CLICK ZOOM	✓
TOUCH ZOOM ROTATE	✓
TOUCH PITCH	✓
MIN ZOOM	0
MAX ZOOM	20
MIN PITCH	0
MAX PITCH	85

Figure 30. The Map

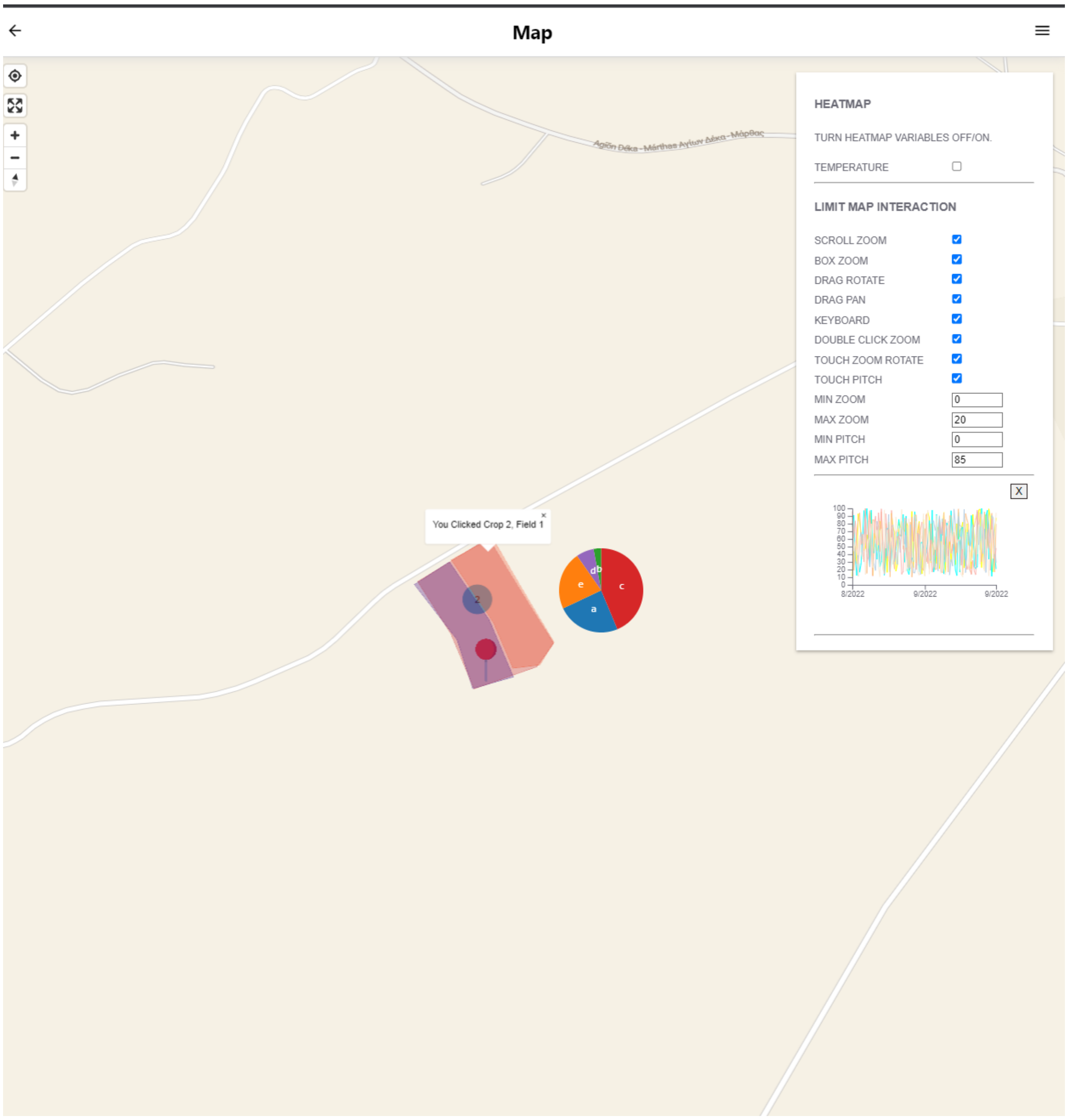
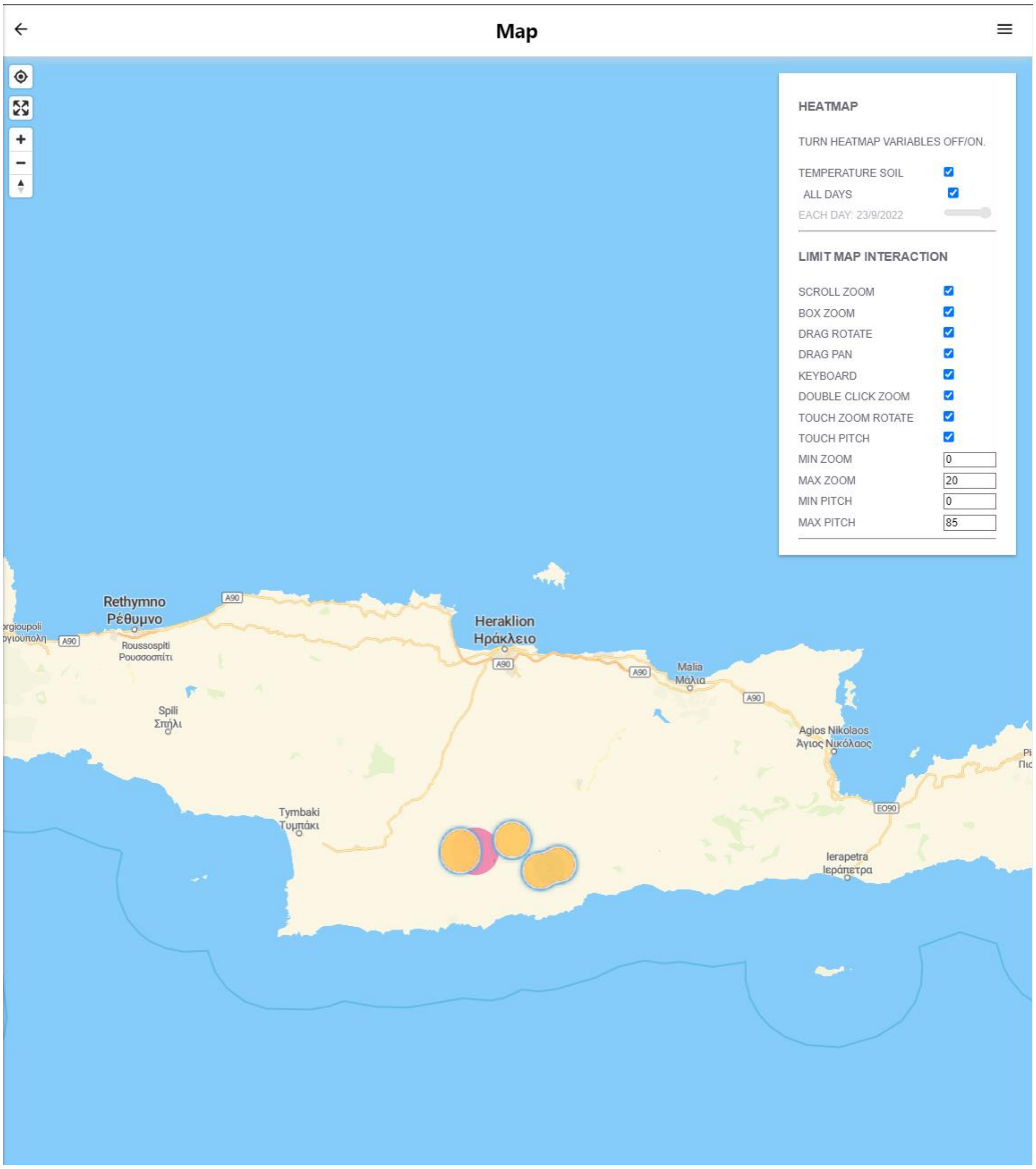


Figure 31. Map's events

Finally, when the user clicks on the heat map option, it will initiate the generation and the rendering of all the available heat map variables. Then the user can either choose to let all the available days to be displayed (see Figure 32) or pick a specific day of his preference using the range option (see Figure 33).



HEATMAP

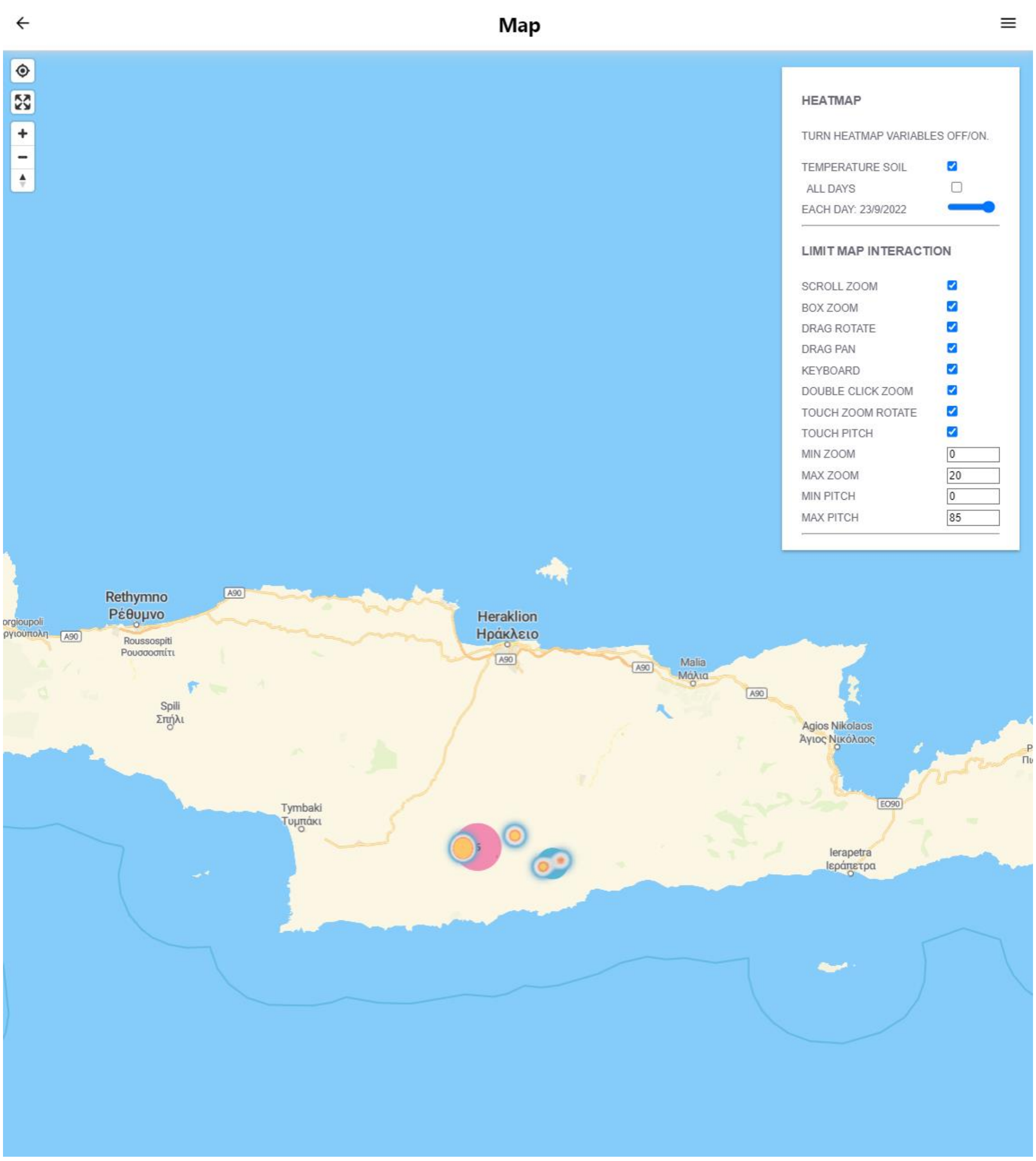
TURN HEATMAP VARIABLES OFF/ON.

TEMPERATURE SOIL	✓
ALL DAYS	✓
EACH DAY: 23/9/2022	

LIMIT MAP INTERACTION

SCROLL ZOOM	✓
BOX ZOOM	✓
DRAG ROTATE	✓
DRAG PAN	✓
KEYBOARD	✓
DOUBLE CLICK ZOOM	✓
TOUCH ZOOM ROTATE	✓
TOUCH PITCH	✓
MIN ZOOM	0
MAX ZOOM	20
MIN PITCH	0
MAX PITCH	85

Figure 32. Heat Map


Map

HEATMAP

TURN HEATMAP VARIABLES OFF/ON.

TEMPERATURE SOIL	✓
ALL DAYS	☐
EACH DAY: 23/9/2022	

LIMIT MAP INTERACTION

SCROLL ZOOM	✓
BOX ZOOM	✓
DRAG ROTATE	✓
DRAG PAN	✓
KEYBOARD	✓
DOUBLE CLICK ZOOM	✓
TOUCH ZOOM ROTATE	✓
TOUCH PITCH	✓
MIN ZOOM	0
MAX ZOOM	20
MIN PITCH	0
MAX PITCH	85

Figure 33. Heat Map (Latest Day)

5 Conclusions

This thesis illustrates the imperatives and intricacies entailed in the visualization of big data, delivering a thorough response by developing an application that will foster, process, and project big data, tailored to the agriculture sector. This application fulfills its aims by implementing the D3.JS and MapLibre GL JS as its visualization tools, operating harmoniously within the robust framework of React. This research demonstrates the capacity of these cutting-edge technologies to harness and visualize massive quantities of geospatial data, primarily generated by sensor networks.

The seamless integration of interactive maps, and sophisticated charts within a user-friendly interface, has not only enhanced data presentation but also elevated the overall user experience. This innovative approach, provides farmers, agricultural experts, or general users with a valuable tool for gaining insights regarding environmental and electromechanical conditions, allowing them to make accurate decisions, optimize resource use, and improve overall crop management.

In the future, enhancements such as the integration of machine learning for predictive analysis, serving multiple users simultaneously, as well as the long-term collection and visualization of their data, could be investigated. Big data visualization has enormous potential in agriculture, and this thesis lays a solid platform for future breakthroughs in this pivotal industry, while also igniting inspiration to other major fields that necessitate the harness of big data.

Bibliography

[1] Sagiroglu Seref and Sinanc Duygu, “Big Data: A Review.” https://ieeexplore.ieee.org/abstract/document/6567202.

[2] Al-Mekhlal Monerah and Ali Khwaja Amir, “A Synthesis of Big Data Definition and Characteristics.” https://ieeexplore.ieee.org/abstract/document/8919591/citations#citations.

[3] Naik Kirtida and Joshi Abhijit, “Role of Big Data in various sectors.” https://ieeexplore.ieee.org/abstract/document/8058321.

[4] P. Ribarics, “Big Data and its impact on agriculture,” *Ecocycles*, vol. 2, no. 1, pp. 33–34, 2016, doi: 10.19040/ecocycles.v2i1.54.

[5] Kumari Riya, “10 Companies that Uses Big Data.” https://www.analyticssteps.com/author/riya-kumari (accessed Jul. 03, 2023).

[6] Arora Vikas, “How Google Applies Big Data to Know You.” https://tweakyourbiz.com/posts/big-data.

[7] “How BI and Big Data are Essential to McDonald’s Growth Strategy.” https://yourshortlist.com/how-bi-and-big-data-is-essential-to-mcdonald-s-growth-strategy/.

[8] S. Neethirajan, “The role of sensors, big data and machine learning in modern animal farming,” *Sensing and Bio-Sensing Research*, vol. 29. Elsevier B.V., Aug. 01, 2020. doi: 10.1016/j.sbsr.2020.100367.

[9] B. Dorsemaine, J. P. Gaulier, J. P. Wary, N. Kheir, and P. Urien, “Internet of Things: A Definition and Taxonomy,” in *Proceedings - NGMAST 2015: The 9th International Conference on Next Generation Mobile Applications, Services and Technologies*, Institute of Electrical and Electronics Engineers Inc., Jan. 2016, pp. 72–77. doi: 10.1109/NGMAST.2015.71.

[10] V. Kumawat and B. Umamaheswari, “International Journal of Trend in Scientific Research and Development (IJTSRD) Internet of Things (IoT) Based Smart Environment Integrating Various Business Applications and Recent Research Directions the Creative Commons Attribution License (CC BY 4.0),” 2019. Accessed: Jul. 07, 2023. [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/59919631/92_Internet_of_Things__IoT__based_Smart_Environment_Integrating_various_Business_Applications_And_Recent_Rese20190703-16248-1t98xqi-libre.pdf?1562144044=&response-content-disposition=inline%3B+filename%3DInternet_of_Things_IoT_Based_Smart_Envir.pdf&Expires=1688762645&Signature=Lcvw4CQcoSDE6MSRWmaQKAshjW7nwP7ZO4NxUOb5z7WAG7LRMJtWdTybDS06n8qhqo5kd9xnImS~J-NtGH3awLu-ZSmP7Ab8yyrD~AENjvosWLJboFd6En5Z1PleZ1J6osjcK1bcXisnM745mvUnsYFQiZhboOfKwchTGS3y7xil4nigxinXMdE76ieieFWubunVOeyT7xmpiJwxpy8kPAU24YiWUHkKMiR2KGkFctgwJeKOXcTK1EjVHxYyfPKfnKcLgsbJbyOzi4x~Hcz6fKXxwpY6kzpVFFCO1Ahw~y3lQQ5oCfNpUSQtnY8WicnAaasbNxy4nmagh6XbAhlMGg__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

[11] S. Khare and M. Totaro, “Big Data in IoT.”

[12] SARAH JOHN, “Advantages of Big Data visualization and why you need it.” https://www.lightsondata.com/advantages-big-data-visualization/#t-1653520455893.

[13] E. Ó. Conchuir, P. J. Ågerfalk, H. H. Olsson, and B. Fitzgerald, “Global software development: Where are the benefits?,” *Commun ACM*, vol. 52, no. 8, pp. 127–131, Aug. 2009, doi: 10.1145/1536616.1536648.

[14] J. Đurković, V. Vuković, and L. Raković, “Open Source Approach in Software Development-Advantages and Disadvantages,” 2008. Accessed: Jul. 20, 2023. [Online]. Available: https://www.ef.uns.ac.rs/mis/archive-pdf/2008%20-%20No2/MIS2008_2_5.pdf

[15] S. Saeed, N. Z. Jhanjhi, M. Naqvi, and M. Humayun, “Analysis of software development methodologies,” *International Journal of Computing and Digital Systems*, vol. 8, no. 5, pp. 445–460, 2019, doi: 10.12785/ijcds/080502.

[16] S. Aghaei, “Evolution of the World Wide Web : From Web 1.0 to Web 4.0,” *International journal of Web & Semantic Technology*, vol. 3, no. 1, pp. 1–10, Jan. 2012, doi: 10.5121/ijwest.2012.3101.

[17] Ranjan Alok, Sinha Abhilasha, and Battewad Ranjit, *JavaScript for Modern Web Development: Building a Web Application Using HTML, CSS, and Javascript*. Accessed: Jul. 20, 2023. [Online]. Available: https://books.google.gr/books?hl=en&lr=&id=b2bdDwAAQBAJ&oi=fnd&pg=PT24&dq=javascript+and+web+history&ots=6fjI6AwqEZ&sig=-8c6y7n7PDFbgfagKY4HkFf5GaY&redir_esc=y#v=onepage&q=javascript%20and%20web%20history&f=false

[18] A. Wirfs-Brock and B. Eich, “JavaScript: The first 20 years,” *Proceedings of the ACM on Programming Languages*, vol. 4, no. HOPL, Jun. 2020, doi: 10.1145/3386327.

[19] “Inheritance and the prototype chain.” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain.

[20] Ahlawat bhishek, “Javascript Features.” https://www.studytonight.com/javascript/javascript-features (accessed Jul. 24, 2023).

[21] S. bin Uzayr, N. Cloud, and T. Ambler, *JavaScript Frameworks for Modern Web Development*. Apress, 2019. doi: 10.1007/978-1-4842-4995-6.

[22] C. Gackenheimer, *Introduction to React*. Accessed: Jul. 25, 2023. [Online]. Available: https://pepa.holla.cz/wp-content/uploads/2016/12/Introduction-to-React.pdf

[23] Anal Roy, “React JS for Web Developer.” Accessed: Jul. 25, 2023. [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/511874/Roy_Anal.pdf?sequence=5

[24] “React Fiber Architecture.” https://github.com/acdlite/react-fiber-architecture

[25] Cincović Jelica and Punt Marija, “Comparison: Angular vs. React vs. Vue. Which framework is the best choice?” http://www.eventiotic.com/eventiotic/files/Papers/URL/50173409-699e-4b17-8edb-9764ecc53160.pdf

[26] Levlin Mattias, “DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte.” https://www.doria.fi/bitstream/handle/10024/177433/levlin_mattias.pdf?sequence=2

[27] “Google Trends.” https://trends.google.com/trends/explore?date=today%205-y&q=React,Angular,Vue&hl=en

[28] “Stack Overflow Trends.” https://insights.stackoverflow.com/trends?tags=angular%2Creactjs%2Cvue.js

[29] Wohlgethan Eric, “Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue.js.” Accessed: Jul. 30, 2023. [Online]. Available: https://reposit.haw-hamburg.de/bitstream/20.500.12738/8417/1/BA_Wohlgethan_2176410.pdf

[30] Y. K. Xing, J. P. Huang, and Y. Y. Lai, “Research and analysis of the front-end frameworks and libraries in e-business development,” in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Feb. 2019, pp. 68–72. doi: 10.1145/3313991.3314021.

[31] “How To Build A Hybrid App Using React Native?” https://www.workfall.com/learning/blog/how-to-build-a-hybrid-app-using-react-native/

[32] Lu Yiren, “How to Make Your React Native Apps Work on the Web.” https://retool.com/blog/how-to-make-your-react-native-apps-work-on-the-

web/#:~:text=React%20Native%20for%20Web%20is%20a%20good%20option%20for%20those,app%20does%2C%20and%20no%20more.
[33] Anderson Cary, “Types of Maps.” https://www.e-education.psu.edu/geog486/node/641.
[34] “9+ Different Types of Maps: Popular Map Types Explained In Detail.” https://www.spatialpost.com/map/.
[35] “What is GIS?” https://www.esri.com/en-us/what-is-gis/overview (accessed Aug. 04, 2023).
[36] “MapLibre GL JS.” https://maplibre.org.
[37] Eriksson Oskar and Rydkvist Emil, “An in-depth analysis of dynamically rendered vector-based maps with WebGL using Mapbox GL JS.” Accessed: Aug. 07, 2023. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:851452/FULLTEXT02.pdf
[38] “React-Map-GL.” https://visgl.github.io/react-map-gl/docs.
[39] “Leaflet.” https://leafletjs.com/.
[40] “Google Maps.” https://en.wikipedia.org/wiki/Google_Maps.
[41] Palchuk Mykhailo, “Mapbox Vs Google Maps: What Maps API Is Best For Your App?” https://www.uptech.team/blog/mapbox-vs-google-maps-vs-openstreetmap.
[42] Ali Hassan Sial, Syed Yahya Shah Rashdi, and Dr. Abdul Hafeez Khan, “Comparative Analysis of Data Visualization Libraries Matplotlib and Seaborn in Python .” https://d1wqtxts1xzle7.cloudfront.net/65736020/ijatcse391012021-libre.pdf?1613836209=&response-content-disposition=inline%3B+filename%3DComparative_Analysis_of_Data_Visualizati.pdf&Expires=1691581848&Signature=bxw0FEQRRw~dHROglQTYLe6sV3MtsBQ60STeYyO7PcSWHfDfgYglEFgh1g8TDOXUM4dLz-fgppQEkAhLVBhaqsEQZYbUoRvkNad6H1uZaGSMkOYv8WRFyLAFpV0DUvQvsUBGR1xpbJkYB6xld7yMgPfmu1DW4vFIsTRwkNkpsjwHFG8ZdMhZ-rrX7voDV6~Xw3KAJX3oeOo5UqcgKM3EVby0dSvE5Y9dEAuW2~kjRHitNn7CQIxE5sFXuNHEBIghoFMlnh~6IF9mqvLpYXs7BTYc-kNmFEFmaHL6tZX-NCY2kakdIm7DSjf5Dlm5D5WDL8ND8R46OoXTg39sV63XKw__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA.
[43] “What is D3?” https://d3js.org/what-is-d3.
[44] T. Tiberghien, “React + D3.js: Balancing Performance & Developer Experience.” https://medium.com/@tibotiber/react-d3-js-balancing-performance-developer-experience-4da35f912484.
[45] S. Benbba, “COMPARISON OF D3.JS AND CHART.JS AS VISUALISATION TOOLS.” Accessed: Aug. 10, 2023. [Online]. Available: https://trepo.tuni.fi/bitstream/handle/10024/131287/BenbbaSafwane.pdf?sequence=2
[46] “Chart.js.” https://www.chartjs.org/docs/latest/.
[47] “D3 or Chart.js for Data Visualisation?” https://www.createwithdata.com/d3js-or-chartjs/.
[48] L. R. Nair, S. D. Shetty, and S. D. Shetty, “INTERACTIVE VISUAL ANALYTICS ON BIG DATA: TABLEAU VS D3.JS.” Accessed: Aug. 11, 2023. [Online]. Available: https://www.learntechlib.org/p/173675/
[49] S. A. South African Institute for Computer Scientists and Information Technologists. Research Conference (2012 : Centurion *et al.*, “What is software architecture?” https://dl.acm.org/doi/abs/10.1145/2389836.2389879.
[50] D. Garlan, “Software Architecture.” Accessed: Jul. 13, 2023. [Online]. Available: https://kilthub.cmu.edu/articles/journal_contribution/Software_Architecture/6609593/files/12101711.pdf
[51] “Class diagrams.” https://www.ibm.com/docs/en/rsm/7.5.0?topic=structure-class-diagrams.

[52] Rapid7, “React for Back-End Devs: Parallels between React and Object Oriented Programming.” https://www.rapid7.com/blog/post/2016/09/08/react-for-back-end-devs-parallels-between-react-and-object-oriented-programming/.
[53] “UML – Class Diagram – Association.” https://ourownjava.com/uml/uml-class-diagram-association/.
[54] “UML Class Diagrams Reference.” https://www.uml-diagrams.org/class-reference.html.
[55] “What’s the differences between helpers and utils? .” https://github.com/erikras/react-redux-universal-hot-example/issues/808.
[56] “GeoJSON.” https://docs.mapbox.com/help/glossary/geojson/.
[57] M. Martin, “What is a Functional Requirement in Software Engineering?” https://www.guru99.com/functional-requirement-specification-example.html.
[58] Martin Matthew, “What is Non-Functional Requirement in Software Engineering?” https://www.guru99.com/non-functional-requirement-type-example.html