HAROKOPIO UNIVERSITY

SCHOOL OF DIGITAL TECHNOLOGY
DEPARTMENT OF INFORMATICS AND TELEMATICS
POSTGRADUATE PROGRAMME OF APPLIED INFORMATICS

Aggregation of air traffic information using satellite and remote sensor data with interactive planning capabilities using crowdsourcing.

Master Thesis

Marinos Kouvaras

Athens, 2024

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

Απεικόνιση και ενημέρωση της εναέριας κυκλοφορίας με χρήση δορυφορικών δεδομένων και δεδομένων απομακρυσμένων αισθητήρων.

Μεταπτυχιακή εργασία

Μαρίνος Κουβαράς

Αθήνα, 2024

HAROKOPIO UNIVERSITY

SCHOOL OF DIGITAL TECHNOLOGY
DEPARTMENT OF INFORMATICS AND TELEMATICS
POSTGRADUATE PROGRAMME OF APPLIED INFORMATICS

Examining Committee

Ethics and Copyright Statement

DEDICATION PAGE

I would like to express my deepest gratitude to my supervisor, Mr. Christos Sardianos, for his continuous support, insightful feedback, and invaluable guidance throughout the development and writing of this thesis. His availability, constructive suggestions, and expert advice were instrumental in overcoming challenges and shaping the outcome of this project.

Additionally, I would like to extend my sincere thanks to the members of my thesis committee, Mr. Anargiros Tsadimas and Mr. Antonis Makris, for their insightful comments, support, and encouragement. Their expertise and suggestions enriched the quality of this work.

I am also thankful to the faculty and staff of Harokopio University, particularly the professors of the MSc in Applied Informatics program. The knowledge and skills they imparted during my studies have been fundamental to both my personal and professional growth. The environment fostered at the university has been a major factor in my success.

A special thank you goes to Elina, who have stood by my side with unwavering support and encouragement throughout this journey. Her belief in me, especially during moments of difficulty, has been a constant source of strength.

QUOTE PAGE

“ἃ γὰρ δεῖ μαθόντας ποιεῖν, ταῦτα ποιοῦντες μανθάνομεν”

Αριστοτέλης, Ἠθικὰ Νικομάχεια, Βιβλίο β’ 1103α

“Αυτά λοιπόν που οφείλουμε να πράττουμε αφού τα μάθουμε, αυτά τα μαθαίνουμε πράττοντες.”

“So what we must do after we learn them, we learn these things by doing them.”

TABLE OF CONTENTS

ABSTRACT IN GREEK

Σκοπός της παρούσας εργασίας είναι η περιγραφή, η ανάλυση των απαιτήσεων και η δημιουργία διαδικτυακής εφαρμογής μέσα στα πλαίσια εκπόνησης διπλωματικής εργασίας για την απόκτηση τίτλου μεταπτυχιακών σπουδών στην Εφαρμοσμένη Πληροφορική.

Το αντικείμενο της εργασίας, πιο συγκεκριμένα, αφορά την ανάπτυξη μιας διαδικτυακής εφαρμογής τύπου web map server όπου οι χρήστες θα έχουν την δυνατότητα δημιουργίας και διάδρασης με γεωχωρικά δεδομένα ενώ ταυτόχρονα θα πραγματοποιείται απεικόνιση δορυφορικών δεδομένων και δεδομένων απομακρυσμένων αισθητήρων.

Οι κυριότερες λειτουργίες περιλαμβάνουν τη δημιουργία χαρτών και δημοσίευσης αυτών σε πραγματικό χρόνο δομημένη στη φιλοσοφία λειτουργίας του πληθοπορισμού. Την αποστολή οδηγιών σε μορφή εντολών στους χρήστες για την εκτέλεση ενεργειών που αφορούν διαδικασίες πολιτικής προστασίας. Ενώ τέλος παρέχεται η λειτουργία άμεσης επικοινωνίας των χρηστών σε μορφή μηνυμάτων.

Η μεθοδολογία που ακολουθήθηκε είναι η αρχιτεκτονική της διαδικτυακής εφαρμογής μέσω εξυπηρετητή. Η ανάπτυξη της εφαρμογής πραγματοποιήθηκε σε επίπεδο full stack με τη χρήση της γλώσσας Javascript και του περιβάλλοντος εκτέλεσης node.js, ενώ παράλληλα έγινε χρήση δεδομένων από ανοιχτές πηγές μέσω της λειτουργίας API για την κάλυψη των επιθυμητών αναγκών.

Τα κυριότερα αποτελέσματα αφορούν μία εφαρμογή η οποία είναι πλήρως λειτουργική και καλύπτει τις ανάγκες όπως αυτές ορίστηκαν. Στα πλαίσια αξιολόγησης των δυνατοτήτων αποτελεί ένα αξιόπιστο και ασφαλές εργαλείο για τις ανάγκες χρήσης του.

Λέξεις κλειδιά: Διαδικτυακή εφαρμογή, εφαρμογή χαρτών, γεωχωρικά δεδομένα, συγκέντρωση δεδομένων

ABSTRACT IN ENGLISH

The purpose of this work is to describe, analyze the requirements, and create a web application within the framework of a master's thesis in Applied Informatics.

More specifically, the subject of the thesis involves the development of a web map server-type application where users can create and interact with geospatial data, while simultaneously displaying satellite data and remote sensor data.

The main functions include the creation of maps and their real-time publication, structured around the concept of crowdsourcing. Instructions in the form of commands will be sent to users to execute actions related to civil protection procedures. Additionally, the application offers a feature for users to communicate instantly via messages.

The methodology followed is based on the architecture of a web application via a server. The development of the application was carried out at a full-stack level using JavaScript and the node.js runtime environment, while open-source data was utilized via API functionality to meet the desired requirements.

The main outcomes include a fully functional application that meets the specified needs. In terms of its evaluation, it is a reliable and secure tool for the intended use cases.

Keywords: web application, map server, node.js, data aggregation

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ISR	Intelligence, Surveillance, and Reconnaissance
NRT	near real-time
RT	near real-time
IP	Internet Protocol
CRUD	Create Read Update Delete
HTML	Hypertext Markup Language
URL	Uniform Resource Locator
CSS	Cascading Style Sheets
HTTP	Hyper Text Transfer Protocol
FTP	File Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
CLI	Command Line Interface
OS	Operating System
NVM	Node Version Manager
CPU	Central Process Unit
DOM	Document Object Model
NPM	Node Package Manager
NPX	Node Package eXecute
API	Application Programming Interface
AST	Abstract Syntax Tree
JSON	Javascript Object Notation
GIS	Geographic Information System
TCP/IP	Transmission Control Protocol/Internet Protocol
ORM	Object-Relational Mapping
ADS-B	Automatic Dependent Surveillance–Broadcast
FAA	Federal Aviation Administration
ICAO	International Civil Aviation Organization
SESAR	Single European Sky ATM Research
ATM	Air Traffic Management
RF	Radio Frequency
UAT	Universal Access Transceiver
IFF	Identification Friend or Foe
WAM	Wide Area Multilateration
LOS	Line Of Sight
TCAS	Traffic Collision Avoidance System
SAA	Sense and Avoid
WGS-84	World Geodetic System 1984
ASTERIX	All Purpose Structured Eurocontrol Surveillance Information Exchange
MLAT	Multilateration
FLARM	Flight Alarm
FIRMS	Fire Information for Resource Management System
NASA	National Aeronautics and Space Administration
URT	Ultra Real Time

SHP	Shapefile
KML	Keyhole Markup Language
CSV	Comma Separated Values
MODIS	Moderate Resolution Imaging Spectroradiometer
VIIRS	Visible Infrared Imaging Radiometer Suite
FIR	Flight Information Region

Chapter.1: Introduction

1.1. Motivation

With the development of technology, the volume of data produced has also increased. Data is mainly created and collected by sensors, which, due to their low cost of manufacturing and use, are now found in a multitude of devices and can easily be acquired by anyone. A significant portion of this data pertains to information related to geospatial measurements and observations, such as weather data or traffic information.

For many years, the use and processing of geospatial data were done on personal computers. Users had the ability to visualize and process their data on personal computers, while specialized software was required for this purpose. With the creation of the Internet in 1989, the opportunity for data sharing among users through the use of special servers arose, with Xerox creating the first map server in 1993 that allowed data visualization through a browser. This was the first web mapping application[1].

Since then, the development of similar technologies has rapidly evolved, and with the dynamic entry of smartphones into the market, anyone can now access and visualize geospatial data through their smartphone, while they can also record and create data themselves. This capability has now become an integral part of daily life, with users being able to choose from a variety of functions, such as weather visualization, location sharing, navigation, and real-time updates on ongoing events (1).

Beyond data creation and collection, a necessary aspect is their visualization and, in general, their presentation in a way that humans can interpret and utilize. Specifically, the visualization of geospatial data is a major field in several sciences, as much of everyday life revolves around the Earth (2).

The functionality of web maps represents an interactive way of visualizing geospatial data in the form of a web page rather than through desktop apps, as was done in the past. Through the internet, data can be shared, displayed, and processed in a browser. The biggest advantage of web maps is that they are accessible like a website and can be accessed by anyone from any device, as

[1] https://en.wikipedia.org/wiki/Xerox_PARC_Map_Viewer

long as they have an internet connection. Web maps allow interaction with map data, such as selecting the type of map, zooming into points of interest, editing data, or creating new maps. Various uses of web maps include data visualization, real-time data display, executing queries, performing calculations with this data, searching to retrieve desired data, creating reference points, and collaboratively creating geospatial data through crowdsourced mapping[2] (3).

One model for data collection, data management, visualization, and performing calculations with simultaneous communication and crowdsourcing capabilities is the C4ISR model (4). The term C4ISR describes the process of Command, Control, Communications, Computers (C4) Intelligence, Surveillance, and Reconnaissance (ISR). It allows us to enhance situational awareness in a given environment, minimizing the time between acquiring information and taking action.

1.2. Scope

The creation of the Panorama App will be a tool for collecting a multitude of open geospatial data using the ISR technique, facilitating the user and helping them visualize it so they can quickly and easily extract all the necessary information and make appropriate decisions. At the same time, crowdsourcing will be enabled, and real-time communication through text chat or command assignment will be allowed (5). This will provide the C4 capability.

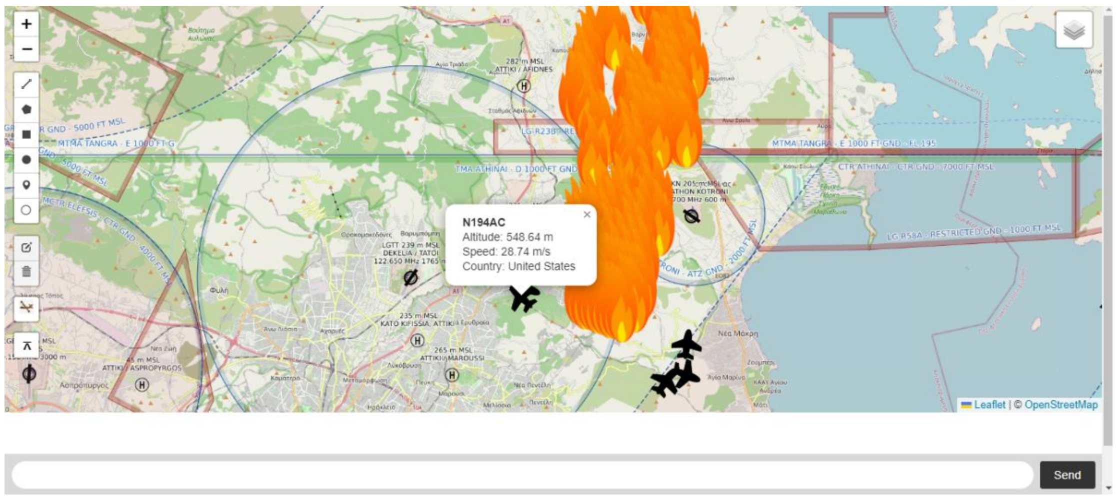

N194AC
Altitude: 548.64 m
Speed: 28.74 m/s
Country: United States

Figure 1: Panorama Application Preview

Specifically, the Panorama App will be a web map application and will include satellite maps, topographic maps, and special interest maps as the visualization base. On these, it will be possible to design and create new maps using design tools such as creating geospatial shapes. Additionally,

[2] https://geobgu.xyz/web-mapping-2021/index.html

the maps will display data collected from satellites and sensors, some of it in near real-time (NRT) and some in real-time (RT). This data will include fires, weather, earthquakes, air traffic, and maritime traffic, which will be visualized on a 2D and 3D map. The application will be aimed at control centers where operators monitor data and make decisions. This way, they will be able to understand the visualized data and make better and more effective decisions. Users will have the ability to send specific instructions for actions and engage in live chat.

For example, a fleet management company for firefighting aircraft could be notified in real-time of a fire within its area of responsibility and be called to act. Managers and operators of firefighting resources will be able to:

- Retrieve location data for the fire incident.
- Identify the geomorphology and geographical features of the area to study their approach and possible limitations.
- Plan their transition with geospatial data to help coordinate operations.
- In real-time, operational managers and administrators can see the movement of aerial resources and receive updates.
- Managers can coordinate and give instructions to the operators of the resources.

The application's usage can extend to various scenarios like the one described above or to different scenarios or combinations thereof.

1.3. Application Requirements

The application will run on a server and will be responsible for collecting and distributing data. Users will be able to access and use it through a web browser on a computer, smartphone, or tablet. The initial design envisions the application being used in decision centers with an internal network, so there will be no per-user authentication but only an initial login. Additionally, user identification has been planned, which includes the IP address of each computer, corresponding to the user who was at the station at the specific time. The application is estimated to be able to handle up to 100 simultaneous users, with the limitation mainly concerning the data sources. The data usage will involve open sources, and the capabilities of the application will be determined based on this.

Specifically, the desired features of the application are as follows:

- The application will run in a node.js environment and will be accessible from a web browser:
    - Desktop: Chrome, Firefox, Safari 5+, Opera 12+, IE 9-11, Edge
    - Mobile: Safari for iOS 7+, Chrome for mobile, Firefox for mobile, IE10+ for Win8 devices
- The base maps will be the most recent and updated by the open-source community, with the update time frame ranging from one (1) to three (3) years, depending on location and geographic area.
- The application will provide the option to use 2D and 3D map environments.
- Design data will be updated every one (1) minute and will be available to all users with CRUD functionality.
- The fire data from a satellite image will concern the area of Greece and will be available within three (3) hours (Near Real Time) and will be updated every five (5) minutes.
- Air traffic data will be available in real time and will be received every one (1) minute.
- Commands and messages will be sent in real-time.
- The application will be able to serve up to 100 users simultaneously.
- The application will perform basic level logging of database activities.

1.4. Synopsis

In the following chapters, we will detail all the elements that will help us implement the application. Specifically, in chapter 2, we will present all the tools that will assist us in creating the application, describe many of its features, and explain why we chose them.

In chapter 3, we will discuss how to combine all these tools to work together and help us develop the application.

In chapter 4, we will try to test and evaluate our application. Through the execution of a hypothetical usage scenario, we will determine the functionality of the application and whether it meets the conditions we set.

Finally, in chapter 5, we will review everything we observed and completed, mention the problems and limitations we encountered, and explore the possibilities for further development and improvement of the application.

Chapter 2: Analysis of the current situation

2.1. Web Map Server Technologies

Geospatial data is a particularly interesting field as it captures a large part of human and Earth activity, helping to draw conclusions and reveal new ways of action. Many applications are based on the utilization of such data to create a product that meets various needs. Currently, there are several available products that operate within the philosophy of web map applications, and below we will mention the most prominent ones that serve as guiding applications or provide the necessary data for the development of new applications (6):

- **GIS:** Geographic Information System is one of the most widely used platforms for map creation and data analysis, chosen by many users.
- **Mapbox:** Mapbox is a very popular solution for various needs, offering a wide range of mapping capabilities with a particular emphasis on visualization elements.
- **CARTO:** CARTO is a web map tool primarily focused on data management, utilization, and analysis.
- **OpenStreetMap:** It includes full web map application features, combining the advantages of community-driven open-source contributions.

However, there are also several web map applications that focus on more specialized data in specific sectors. Examples include:

- **Flightradar24:** This web map application mainly focuses on the visualization of aviation data.
- **Windy:** This web map application primarily focuses on the visualization of weather data with the ability to forecast weather conditions.
- **Marine Traffic:** This web map application mainly focuses on the visualization of maritime traffic data.

2.2. Tools and Technologies for Application Development

Below, we will mention all the tools and systems required for the creation and implementation of the application. It is important, through identifying the needs as analyzed and derived for the application's implementation, as well as from existing applications, to select all the available tools that will not only provide the necessary functionality but will also ensure the seamless operation of the application in terms of interaction, performance, and security.

2.2.1. Javascript

One of the most crucial decisions that will significantly influence the development approach of the application is the choice of programming language. For this specific application, JavaScript has been selected as the programming language. The rationale behind this choice is that, as stated on its official website, JavaScript is a “lightweight interpreted (or just-in-time compiled) programming language with first-class functions. While it is most well-known as the scripting language for web pages, JavaScript is a prototype-based, multi-paradigm, single-threaded, dynamic language, supporting object-oriented, imperative, and declarative (e.g., functional programming) styles”[3] Therefore, JavaScript is a highly suitable choice for the functionality required by this application (7).

It is one of the best options for web application development as it is a powerful language supported by all major web browsers. Additionally, JavaScript is very popular, with a large support community, an extensive array of libraries, and continuous evolution. Finally, it is an ideal language for developers with limited experience due to its ease of learning and user-friendly nature. As evidenced by a 2023 survey conducted by the renowned platform Stack Overflow, JavaScript was voted the most popular programming language.

[3] https://developer.mozilla.org/en-US/docs/Web/JavaScript

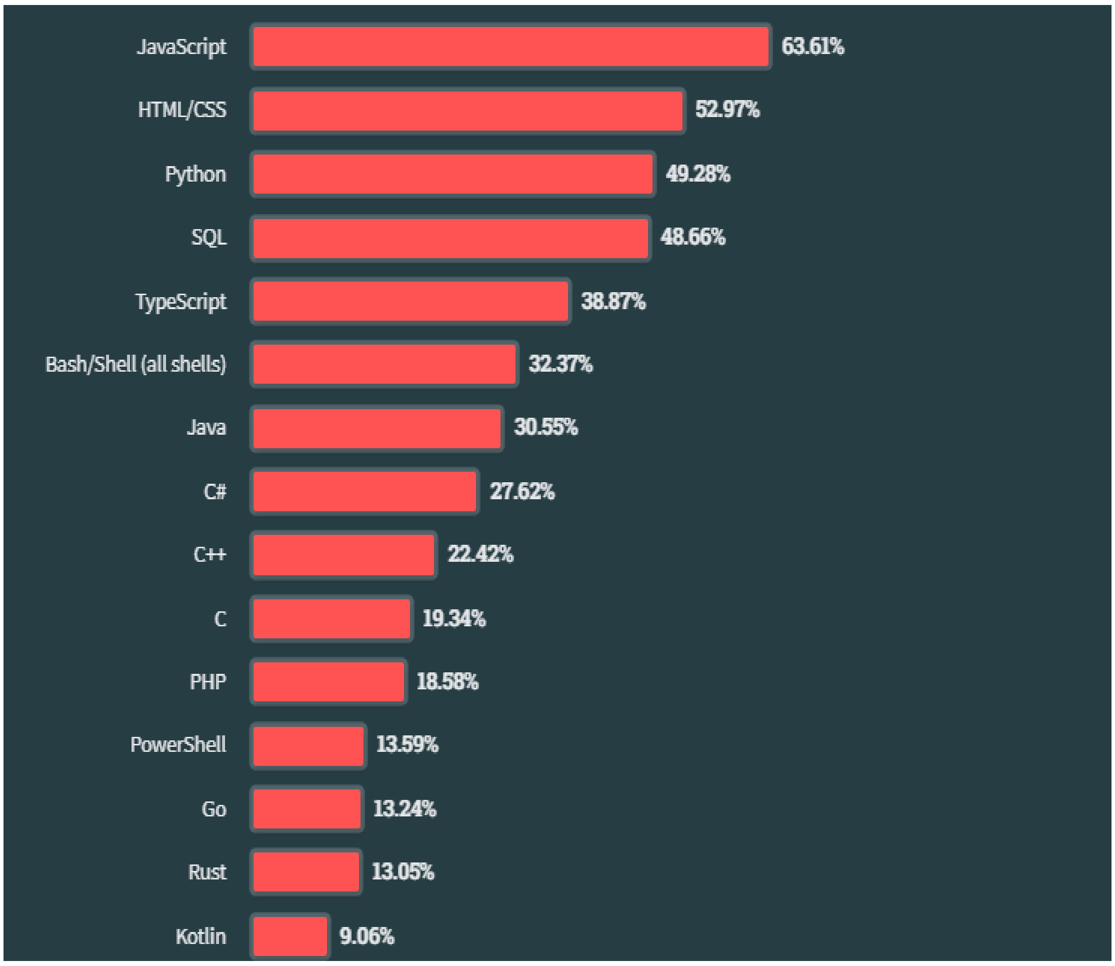


Figure 2: Programming language popularity, StackOverflow survey for 2023

2.2.2. Web Servers

The application will essentially be a web server, specifically a map server that distributes necessary geospatial data to users. The following section will describe the process of sharing and serving files and data through web servers to provide a clearer understanding of how the application will operate.

To begin with, it is essential to define that the most fundamental element of a website is the HTML text. This file is stored on a node on the internet, which is referred to as a server because it serves content on the internet, allowing users to retrieve it. For instance, when a user enters a URL in a browser, the browser locates the corresponding file, interprets it, and displays its content on the screen. During this interpretation process, the browser also applies CSS styling rules and executes any associated JavaScript code linked to the HTML file.

Therefore, to host a website, two key components are required:

- The necessary text and code files.
- The appropriate hardware to store these files and the suitable environment to serve them.

Once the text and code files are created, the next question is how to serve them so that they are accessible to everyone. This is addressed by the creation of web servers, which can refer to either hardware or software. From a hardware perspective, a web server is a computer that stores these files, is connected to the internet, and can serve the files via a specific URL.

From the software perspective, a web server encompasses all the tools required to manage how users can access these files. The minimum component needed for this function is an HTTP server, which is software that recognizes URLs and the Hypertext Transfer Protocol (HTTP)—essentially the communication protocol between the browser and the server.

In the above basic setup, we have a static server. In contrast, a dynamic server includes additional components and offers more advanced functionalities. A simple example of web server operation in its most basic form is when a web browser requests a file that resides on a server. In this scenario, the browser sends a request for the file via the HTTP protocol. Once the request reaches the appropriate server (hardware), the HTTP server (software) responds by sending the file back via HTTP. When the browser receives the file, it reads it and displays its content on the user's screen.

Previously, the operation of servers from both hardware and software perspectives was discussed. An additional noteworthy aspect to describe is the distinction between different types of web servers from the software perspective. As mentioned earlier, there are two main types of web servers in terms of software:

- **Static Servers:** These servers handle requests for static files, such as HTML, CSS, and JavaScript files, which do not change in response to user interactions. The server simply delivers the requested file as it is. Static web servers consist of a computer with an installed HTTP server (software). The term "static" originates from the fact that the server returns files exactly as they are, without any preprocessing. This essentially means that the server can only respond to GET requests for files that already exist and return them as the response to the request.

- **Dynamic Servers:** These servers are capable of generating content on-the-fly based on user requests. They involve additional components and can execute server-side scripts to produce dynamic web pages that can change based on user input or other factors. More specifically, dynamic web servers not only include an HTTP server (software) but also additional software such as an application server and a database. The term "dynamic" arises from the server's ability to dynamically generate HTML content or any other content in response to a request and then return it to the user who made the request. This process involves building the content on-the-fly based on the specific query and user interaction.

In the application map server will receive requests from clients and will respond with the appropriate content which is the HTML map with all the source and database data. This process involves handling data, ensuring fast response times and maintaining data accuracy and security.

By understanding the mechanism of web server operation, it becomes more and more clear how the application will function as an intermediary between the data providers or stored data and the end-users enabling them to access and interact with spatial information seamlessly.

2.2.3. Web Protocols

Another important aspect to describe is web protocols. Essentially, communication protocols define the rules for interaction between clients and servers, specifying how communication should take place and outlining the various operations that can be performed. One well-known protocol mentioned earlier is HTTP (Hypertext Transfer Protocol), which, as its name suggests, is primarily used for transferring hypertext.

There are other types of protocols as well, such as FTP (File Transfer Protocol) and WebSockets, which will also be relevant to our application. The majority of internet communication occurs via HTTP, including its secure version, HTTPS (Hypertext Transfer Protocol Secure). HTTPS is similar to HTTP but adds an additional layer of security to the communication.

A protocol refers to a defined and standardized method of communication between a client and a server. Specifically, the HTTP (Hypertext Transfer Protocol) protocol is a textual and stateless protocol.

- **Textual:** This means that all data is transmitted in a human-readable text format, making it easily understandable.

- **Stateless:** This means that neither the server nor the client retains information from previous communications. For example, in a simple HTTP interaction, a server does not remember if a client has logged in or the current state of a transaction. The server only responds to the current request without any memory of past interactions.

It should be noted that according to the HTTP communication protocol, clients can only make requests to the server, while servers can only respond to client requests. When clients send a request, they must specify the URL where the desired file is located. In turn, servers are required to respond to every request, even if it is with an error or failure message.

To facilitate communication over HTTP, specific methods are used to indicate the desired action and type of request. These methods include:

- **GET:** Retrieves the requested file or resource.
- **HEAD:** Similar to GET, but the response does not include the response body.
- **POST:** Submits data to be processed to a specified resource.
- **PUT:** Replaces the existing content of a resource with the content provided in the request.
- **DELETE:** Deletes the file or resource at the specified location.
- **CONNECT:** Establishes a tunnel to the server identified by the target resource.
- **OPTIONS:** Describes the communication options available for the target resource.
- **TRACE:** Performs a message loop-back test along the path to the target resource.
- **PATCH:** Applies partial modifications to a resource.

2.2.4. Git Bash

The application development will be conducted on Microsoft Windows. However, since many of the tools required are available via CLI and are best executed in a Bash environment—something not widely supported by Windows OS—Git Bash will be installed for this purpose.

Git Bash is an application for the Windows environment that provides a simulation layer for executing Git commands[4]. Git is a version control system that allows us to manage and track changes to our code. It enables visibility of any modifications made to the code and facilitates the creation of different versions of the same code. This capability allows for the testing of new features and the separation of code sections.

In addition to providing Git command functionality, Git Bash offers several features of Bash, which is a shell environment that allows interaction with the operating system through written commands. This enables the execution of many basic commands commonly used in Unix systems, facilitating a more versatile development environment on Windows.

2.2.5. NVM

After selecting the programming language for application development and determining the operating system to be used, the next step is to choose the execution environment for the application.

The chosen execution environment will be Node.js, which, as will be discussed later, offers several advantages and is one of the best choices for web application development. Before describing the Node.js environment, however, it is essential to mention a necessary component: the installation and use of NVM (Node Version Manager).

NVM is a tool for managing different versions of Node.js. It allows users to select a specific version for compatibility and testing purposes. Although the plan does not include having multiple versions of Node.js or changing versions frequently, using NVM is beneficial for maintaining compatibility and ensuring that the environment is set up according to best practices. Additionally,

[4] https://www.atlassian.com/git/tutorials/git-bash#:~:text=Git%20Bash%20is%20an%20application,operating%20system%20through%20written%20commands.

installing NVM provides the capability to work through CLI, which, as previously mentioned, is advantageous and preferred during the development phase[5].

2.2.6. Node.JS

Having selected the programming language, the next crucial decision is the development environment for the application. Among the various options, Node.js has been identified as the most suitable. According to official documentation, Node.js is “a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools, and scripts”[6].

As observed, Node.js is both open-source and free, meaning its use is provided at no cost, and its source code is accessible for modification as needed. Node.js serves as a runtime environment for JavaScript code and facilitates the creation of scalable network applications, a feature well-suited to the needs of the application (8).

Since Node.js will play a significant role in the development of the application, it is useful to discuss it further. Node.js is a widely used choice for developing various types of applications. A key feature of Node.js is that it runs the V8 JavaScript engine, which is also used by the Google Chrome browser, but operates outside of the browser. This allows Node.js to deliver high performance similar to what is achieved within the browser environment.

Additionally, all operations in Node.js are executed within a single process without creating new threads and are managed through asynchronous methods. This approach allows Node.js to perform I/O operations, such as network interactions, database functions, and file system operations, by starting these tasks and waiting for their completion. This design prevents unnecessary CPU resource consumption. Consequently, Node.js can handle thousands of simultaneous connections with a single server, avoiding the complexities associated with thread concurrency.

A significant advantage of Node.js is its use of JavaScript, meaning that someone proficient in JavaScript for frontend development can also write server-side code. This allows frontend

[5] https://github.com/nvm-sh/nvm
[6] https://nodejs.org/en

developers to create backend code without needing to learn a new programming language, which can be challenging and time-consuming. Although the same programming language is used for both frontend and backend development, there are important differences in the development of each component. However, these differences will not be discussed further as they are beyond the scope of this analysis.

Node.js also offers a wide range of tools suitable for networking, leveraging the fact that both browsers and Node.js use JavaScript. However, there are significant differences between developing applications for the browser and creating Node.js applications. For instance, browser-based development often involves interacting with the DOM and other Web Platform APIs, such as Cookies. These features are not available in the Node.js environment, nor is there interaction with the window object.

On the other hand, developing in Node.js provides access to libraries and tools not available in the browser environment, such as local file system interaction. This allows for more extensive capabilities in handling server-side operations and managing local resources.

Finally, a crucial aspect to consider is that, while browsers always run the latest version of software, Node.js allows users to choose and manage the version that best suits their needs. This flexibility enables developers to control the environment parameters required for each specific application. Thus, Node.js is recognized as one of the best options for web application development, as it provides all the tools and choices necessary for the effective functioning of our application.

2.2.7. V8 JavaScript Engine

As described earlier, Node.js utilizes the V8 JavaScript engine at its core. V8 is the JavaScript engine that powers Google Chrome, executing JavaScript code as users navigate through the browser. Notably, the V8 engine operates independently of the browser in which it is hosted. This independence made V8 an ideal choice for Node.js development and for various server-side applications, as well as desktop apps via tools like Electron.

For comparison, other JavaScript engines include:

- Firefox: SpiderMonkey
- Safari: JavaScriptCore (also known as Nitro)
- Edge: Initially based on Chakra, but more recently rebuilt using Chromium and the V8 engine.

The V8 engine is written in C++ and is continuously improved to enhance speed and performance. It can run on all major operating systems. Modern versions of V8 use a technique called "compile on time" and do not rely on an interpreter. This transition was crucial for handling large-scale JavaScript applications, such as Google Maps, which require the execution of extensive codebases efficiently. As a result, V8 enables high-performance, continuous execution of code in the browser.

2.2.8. NPM

NPM is the standard package manager for Node.js, serving as the official tool for retrieving and installing libraries and tools as needed. As one of the most widely used package managers, it had 2.1 million packages available as of September 2022[7]. Initially created to manage and utilize packages within the Node.js environment, NPM is now also used for frontend projects.

Alternative package managers include Yarn and pnpm, but they will not be discussed here, as NPM remains the largest repository for libraries where we can find and utilize the necessary resources for our application. NPM consists of three main components:

- Website: Allows users to search for packages and manage access to both private and public packages.
- Command Line Interface (CLI): Operates in the terminal and is the primary method for developers to interact with NPM.
- Registry: A large database of JavaScript software and associated metadata.

In summary, NPM will play a significant role during development, aiding in connecting existing packages to our project, finding tools for various tasks, and sharing code.

[7] https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager

2.2.9. NPX

NPX is a command-line utility that allows for the execution of npm packages without the need for prior installation. It installs the package temporarily in memory, which means users can execute it directly. NPX also allows users to specify the version of the package they wish to run.

This command is particularly useful for running commands that create structures within an application, often for initialization purposes, and which do not require ongoing use by the developer. NPX will be essential for the initial creation and setup of the application's structure.

2.2.10. Express

As described in the first chapter, the application will be a type of web map. Therefore, it is necessary to choose a web framework that will handle a significant portion of the web functionality. For this purpose, the Express web framework has been selected.

According to the official documentation, Express is “a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications”[8]. This framework operates with Node.js and allows for relatively easy customization of web functions. Specifically, it enables the creation of API applications using HTTP, leverages the advantages of Node.js rather than hindering them, and facilitates the use of middlewares with efficient routing capabilities. Overall, Express is user-friendly, offers numerous options for application development, and ensures both security and high performance.

2.2.11. Browserify

As previously discussed, the development environment for the backend has been chosen as Node.js. However, we have yet to address the frontend development environment. Although there are numerous options available for this application, and given that extensive dynamic content display is not required on the frontend, the Browserify library will be used.

Browserify creates a bundle that includes all the dependencies and code from the backend. It essentially parses and fragments the Abstract Syntax Tree (AST) to link each function with its

[8] https://expressjs.com/

required dependencies. The final bundled code is then presented on the frontend and can be utilized for any functionality in the same manner as with a frontend library.

2.2.12. Leaflet

As mentioned in the first chapter, the primary focus and goal of this project is to create a web map application. For this purpose, the Leaflet library has been chosen. As stated in the literature, "Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps. Weighing just about 42 KB of JS, it has all the mapping features most developers ever need"[9]. Its main features include ease of use, high performance, and compatibility across multiple platforms, including both desktop and mobile, with strong community support. These characteristics make it ideal for the needs of the application (9).

To better understand how to use the library, it's important to discuss the architecture and functioning of web map systems. A web map does not just refer to a map on the internet but rather a map that is powered by the internet. Typically, these maps are interactive and do not contain all the data themselves but instead pull data from various sources. The philosophy of web maps is that they consist of multi-layered data. Usually, background data are static and consist of base maps created for general use.

To better understand how to use the library, it's important to discuss the architecture and functioning of web map systems. A web map does not just refer to a map on the internet but rather a map that is powered by the internet. Typically, these maps are interactive and do not contain all the data themselves but instead pull data from various sources. The philosophy of web maps is that they consist of multi-layered data. Usually, background data are static and consist of base maps created for general use.

Raster tiles represent the older and simpler method of using tiles in web mapping. They consist of images with a size of 256x256 pixels. For each zoom level, a separate list of tiles is required, and as zoom increases, the number of images increases as well. Generally, a rule followed is that for every increase in the zoom level, each tile is subdivided into four smaller tiles. For example, at zoom level 0, only one tile is needed. At zoom level 1, we need 2 x 2 = 4 separate tiles. The general rule is 2^n X 2^n = 4^n, where n is the zoom level. The maximum zoom level is 18, and in this

[9] https://leafletjs.com/

case, it would require 68,719,476,736 tiles to cover the Earth's surface. In each interaction with the map, only the necessary tiles are loaded, which minimizes the data that needs to be transferred while also enhancing performance. All images are served by a tile server, which is a static server where all tiles are stored in a specified directory. The method to retrieve a map tile is via a GET request in the format http://.../Z/X/Y.png, where Z represents the zoom level, X is the column, and Y is the row of the desired tile. In Figure 2[10], the operation of tile layers and the method for retrieving each tile using the GET method are illustrated.

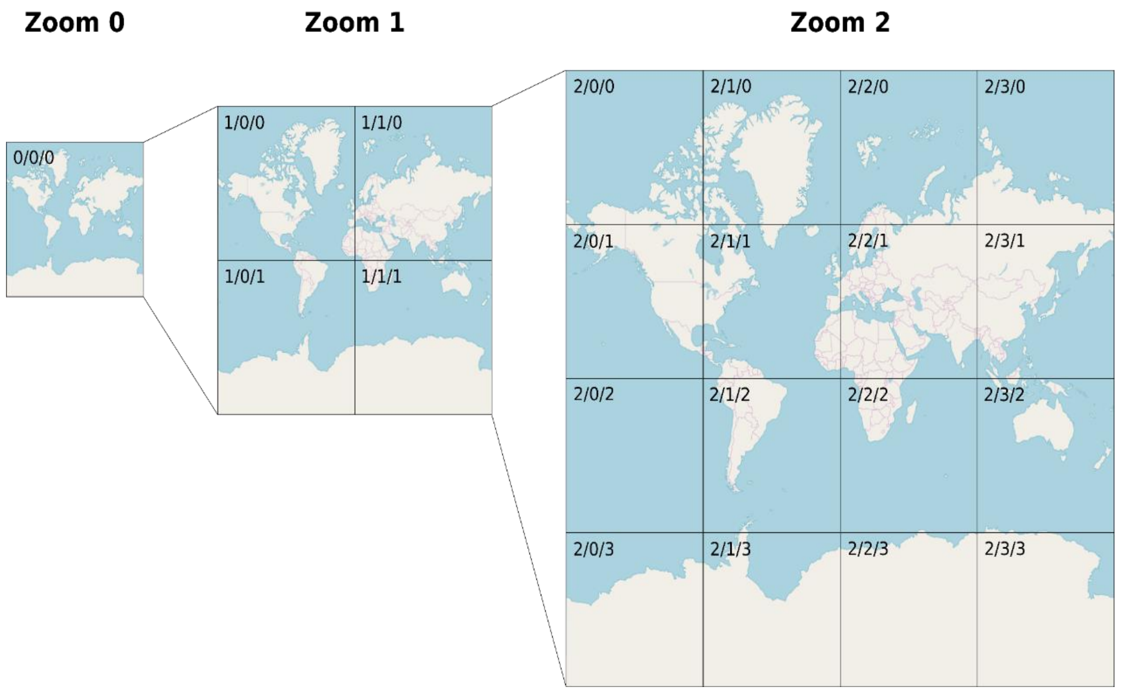


Figure 3: Zoom level and tile sizes

Another method for map layers is the use of vector tiles. Vector tiles offer the advantage of preserving the orientation of map features even when the map is rotated, and they do not require reloading tiles at different zoom levels, which allows for smoother functionality. Unlike raster tiles, vector tiles provide more flexibility and efficiency in rendering and interaction. However, Leaflet does not use vector tiles by default, and thus this technique will be briefly mentioned as it falls outside the scope of this application.

After the initial loading of the base map, all additional layers are placed on top of it and are referred to as foreground layers. These are usually vector layers, including points, lines, polygons,

[10] https://geobgu.xyz/web-mapping-2021/images/tile_zoom_levels.png

or other forms of data. These foreground layers are essential for adding interactive and dynamic elements to the map, and their details will be discussed further below.

2.2.13. GeoJSON

As mentioned, the process of creating maps involves constructing layers that are stacked on top of each other to produce the final result. This can be accomplished using various methods and tools, but the data needs to be described and managed effectively by the application. The chosen method is to use GeoJSON format. GeoJSON is a JSON-based format that ensures full compatibility with web technologies such as JavaScript and web maps. It is also capable of describing complex vector layers with attributes that need to be visualized on the map.

GeoJSON is based on the JSON format and is a plain-text format used to describe vector geometries. It has become a widely adopted format for exchanging geospatial data, with most GIS applications now exclusively using it. Essentially, it serves as a means of transferring data between the client and server in web applications. GeoJSON supports the following geometric elements: Point, LineString, Polygon, MultiPoint, MultiLineString, and MultiPolygon[11]. When a geometric shape includes additional attributes beyond its type and coordinates, it is referred to as a feature object. A collection of features is grouped into a broader category known as Feature Collections. Below is an example of a GeoJSON description[12]:

[11] https://geojson.org/
[12] https://datatracker.ietf.org/doc/html/rfc7946#section-3.1.10

```
{
        "type": "FeatureCollection",
        "features": [{
            "type": "Feature",
            "geometry": {
                "type": "Point",
                "coordinates": [101.0, 0.7]
            },
            "properties": {
                "property0": "val0"
            }
        }, {
            "type": "Feature",
            "geometry": {
                "type": "LineString",
                "coordinates": [
                    [101.0, 0.0],
                    [102.0, 1.0],
                    [103.0, 0.0],
                    [104.0, 1.0]
                ]
            },
            "properties": {
                "property0": "val1",
                "property1": 0.0
            }
        }, {
            "type": "Feature",
            "geometry": {
                "type": "Polygon",
                "coordinates": [
                    [
                        [100.0, 0.0],
                        [101.0, 0.0],
                        [101.0, 1.0],
                        [100.0, 1.0],
                        [100.0, 0.0]
                    ]
                ]
            },
            "properties": {
                "property0": "value0",
                "property1": {
                    "this": "that"
                }
            }
        }]
    }
```

Table 1: Typical GeoJSON format

As observed, the structure of GeoJSON is simple and human-readable, allowing for the description of shapes with various attributes. However, a disadvantage of GeoJSON is that in the case of complex shapes, the size of the GeoJSON file can become quite large compared to other formats, such as shapefiles. It is important to note that the example provided is a generic example; specifically, the Leaflet library uses coordinates in the WGS84 format (EPSG:4326), where the format is [lon, lat]. Therefore, during the usage and development of the application, this coordinate format should be maintained.

2.2.14. Postgres

After discussing the creation and usage of the base map, as well as the method for creating and defining data, the next step is to address data storage. The data need to be stored in a database capable of managing them efficiently. Although there are several options available, PostgreSQL was chosen as the database system. According to official literature, "PostgreSQL is a powerful, open-source object-relational database system with over 35 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance"[13]. The high-performance levels, widespread use by numerous users, and the open-source nature of PostgreSQL make it an ideal choice for the application's development. Furthermore, PostgreSQL offers tools for managing spatial data through the PostGIS extension, which adds a significant reason for selecting this database system.

The database is a crucial component of the application, and understanding its operation is essential for effective development. PostgreSQL operates on a client/server model. When a new session is initiated, a server process named “Postgres” starts. This server manages the database files, handles connection requests from clients, and performs the necessary actions on their behalf.

Clients can vary in form, they may be a desktop application, a web application, or another server. In a simple client/server model, the client and server may be on different hosts, communicating over a TCP/IP connection. PostgreSQL supports simultaneous connections from multiple clients. It achieves this by creating a new process for each connection, a technique known as “forking.” From this point onward, the client and the new server process communicate independently, without involving the initial “Postgres” process.

[13] https://www.postgresql.org/

For the application, this model means that the server's operations will be handled by the Node.js server, which will then distribute the required data to other users. To manage interactions between the Node.js server and PostgreSQL, an additional tool will be used: the ORM (Object-Relational Mapping) library Sequelize, which will be described later.

2.2.15. PostGIS

As previously mentioned, PostGIS extends the capabilities of PostgreSQL by enabling the storage, processing, and calculation of spatial data. This extension is essential for the development of the application and its installation is necessary.

Here are the capabilities provided by PostGIS[14]:

- Spatial Data Storage: Enables the storage of spatial data, such as lines and polygons, in both 2D and 3D formats.
- Indexing: Allows for efficient searching and retrieval of data based on their location.
- Geometric Functions: Provides functionalities for filtering and analyzing data, measuring distances, areas, etc.
- Geometric Processing: Offers tools for processing spatial data, including transformations, simplifications, and more.
- Raster Data Support: Includes support for raster data, such as elevation and weather data.
- Geocoding: Supports geocoding and reverse geocoding functionalities.
- Integration: Facilitates integration with other tools, such as QGIS, MapServer, and others.

These features make PostGIS a powerful tool for handling spatial data, and its integration with PostgreSQL enhances the ability to manage complex geographic information effectively.

2.2.16. Sequelize

To create the database, the Code First approach will be employed. This means that all database entities and attributes will be created through the ORM (Object Relational Mapper) without having an initial database schema. Each new element or entity will be created in the database via the ORM rather than the other way around. For this purpose, Sequelize has been chosen. As noted in the literature, "Sequelize is a modern TypeScript and Node.js ORM for Oracle, Postgres, MySQL,

[14] https://postgis.net/

MariaDB, SQLite, and SQL Server, featuring solid transaction support, relations, eager and lazy loading, read replication, and more”[15].

In general, an Object Relational Mapper is a piece of software that handles the translation between the data stored or created in a database and the data used or created in an object-oriented programming environment such as Node.js. From a programming perspective, the Sequelize ORM allows us to define and manage our database schema using JavaScript, facilitating the interaction between the PostgreSQL database and the Node.js application[16].

2.2.17. Turf

A significant aspect of our application involves managing geospatial data. While these tasks could be efficiently handled using PostGIS, which excels in performing such operations swiftly and effectively, we will opt for an alternative library for simpler queries: Turf.js which is a geospatial analysis library that provides a range of functions from basic to complex for processing GeoJSON data[17].

The choice to use turf.js stems from its ability to perform geospatial operations efficiently and effectively, and its compatibility with Node.js makes it an ideal choice. Moreover, turf.js allows for operations on data without the need to first store it in a database, which is particularly useful for handling dynamic data provided through APIs in our application.

2.2.18. Socket

The term "socket" refers to the bidirectional communication between a client and a server. Typically, a server runs on a computer and has a socket for each port, allowing it to communicate with clients. The server listens on these ports, ready to respond to any client attempting to connect. From the client’s side, it knows the hostname and port number of the server it wants to communicate with and attempts to connect. Once a successful connection is established, a socket connection is formed, enabling direct communication between the server and client[18].

[15]https://sequelize.org/
[16]https://www.prisma.io/dataguide/types/relational/what-is-an-orm
[17]https://turfjs.org/docs/intro
[18]https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html#:~:text=Definition%3A,programs%20running%20on%20the%20network.

Sockets are crucial for sending messages and commands within the application. Real-time data and message exchange will be facilitated through this communication method. To achieve this, the library Socket.IO will be used.

Socket.IO supports WebSocket communication, which is tailored for web applications. It provides the reliability needed to handle scenarios where communication might be interrupted. Additionally, it offers scalability features essential for gradually developing application capabilities without restricting growth. Most importantly, Socket.IO enables bidirectional communication with low latency and is based on an event-driven model for interaction between server and client[19].

2.2.19. Winston

An essential feature for our application is the implementation of logging. Generally, logging involves recording events that occur during the execution of processes at various levels, such as issues, errors, or simple informational messages. These messages are captured and stored in separate files as a history. This allows us to categorize messages, store them appropriately, and refer to them as needed to troubleshoot problems or better understand the operation of a process. For this purpose, the chosen tool is the Winston library. Winston is one of the most well-known logging libraries for Node.js, offering excellent features for the application. It is easy to use, providing the ability to set logging levels and methods. Additionally, it allows customization of log message formats, logging levels, and the choice of storage medium for each message and category[20].

2.2.20. ADS-B

One of the main aspects that the application will manage is the visualization of airborne traffic data, which will be received from ADS-B (Automatic Dependent Surveillance–Broadcast) transponders. The ADS-B system is essentially a satellite-based geolocation and data exchange network relying on radio frequencies. This system provides automated and reliable information on the position, altitude, and intentions of aircraft (10).

[19] https://socket.io/
[20] https://github.com/winstonjs/winston

The FAA and ICAO have established operational rules for the ADS-B system, mandating its use by all aircraft operating in controlled airspace from 2020 onwards. However, due to the SARS-CoV-2 pandemic, this deadline was postponed to 2022. Through the NextGEN and SESAR initiatives, there has been an agreement on the need for continuous data broadcasting from aircraft using ADS-B transponders. Specifically, a common policy has been adopted for modernizing the ATM system that does not rely on expensive and outdated radar systems.

The ADS-B system comprises two (2) subsystems with in (receiving) and out (transmitting) functionalities. According to the design, both control stations and aircraft use both subsystems. Communication occurs via satellites, with RF frequencies creating separate communication networks. These networks transmit data that can provide visualization and correlation of elements in three dimensions. The data integration offered by the system provides real-time information such as NOTAMs, significant weather, traffic, and more. However, for the application, only data related to airborne traffic from the ADS-B system will be selected. Aircraft data is first collected by the aircraft's navigation systems and then transmitted. Some of the emitted information includes the aircraft's identifier (ID), route, current position, signal reliability, etc.

Two of the interface protocols are UAT at 978 MHz and 1090ES at 1090 MHz. The use of the UAT protocol requires the installation of specialized equipment, which is why it is primarily used in airspace controlled by the FAA in the United States and Eurocontrol in Europe. The 1090ES protocol leverages the capability of data dissemination via Mode S. It is worth noting that Mode S is a function of the IFF (Identification Friend or Foe) system, and since all aircraft operating in controlled airspace are required to be equipped with it, the ADS-B reference will imply the use of the 1090ES protocol.

Data is transmitted automatically every second, compared to radar data where the response time ranges from four seconds in aircraft approach areas to twelve seconds for aircraft in flight. As expected, significant emphasis has been placed on the standardization and definition of ADS-B out signals, as these are the ones transmitted to ground stations and the ADS-B device. The signals are transmitted freely without significant encryption, with only the last 24 bits ensuring data integrity.

Ground transponders are central to the system and ensure the connectivity between aircraft, satellites, and ground control stations. The system connectivity methods via satellites are summarized as follows:

- **Direct Downlink**
- **Inter Satellite Link**
- **SatCom**

Data is received and forwarded to control stations, where appropriate algorithms process and display the data through interface applications, and simultaneously, it is distributed to the aircraft cockpits. For the system's usage, a WAM (Wide Area Multilateration) network is developed to ensure reliable data transfer. WAM networks can cover much larger distances than radar and allow for the connection of multiple ground stations. In remote and inaccessible areas, satellites are used to ensure Line of Sight (LOS). It is worth noting that even ships communicating via satellites could participate in developing this network.

Key operational modes of the system that could be mentioned are:

Airport Operations:

- Aircraft servicing
- Taxiway management
- Departure management
- Arrival management

Air Traffic Management:

- Wide coverage of tracks in remote areas
- Timely warning for avoiding track conflicts and upgrading the TCAS (Traffic Collision Avoidance System) for commercial aircraft and SAA (Sense and Avoid) system for UAVs (Unmanned Aerial Vehicles).

For obtaining the necessary data, open-source data will be used, specifically the API provided by the OpenSky Network website[21]. Through this API, we can access live aviation data for research

[21] https://opensky-network.org/

purposes. The data are provided in the form of state vectors, including aircraft information sourced from ADS-B and Mode S transponders. The information includes the aircraft's position, speed, altitude, and identifier. For each aircraft, the API retrieves data from the moment it enters the range of a transponder. If data has not been updated within 15 seconds, the same data is used in each response from the API. If an aircraft has no data for any reason or if the data cannot be updated for more than 15 seconds, that track is omitted from the message and only included again when it regains one of the above properties.

In the table below, the JSON response data for each query are illustrated:

Property	**Type**	**Description**
time	integer	The time which the state vectors in this response are associated with. All vectors represent the state of a vehicle with the interval [time−1,time]
states	array	The state vectors.

Table 2: Open-Sky JSON response data

The property of states is a two-dimensional array, and each row represents a state vector and contains the following information:

Index	**Property**	**Type**	**Description**
0	icao24	string	Unique ICAO 24-bit address of the transponder in hex string representation.
1	callsign	string	Callsign of the vehicle (8 chars). Can be null if no callsign has been received.
2	origin_country	string	Country name inferred from the ICAO 24-bit address.

Index	Property	Type	Description
3	time_position	int	Unix timestamp (seconds) for the last position update. Can be null if no position report was received by OpenSky within the past 15s.
4	last_contact	int	Unix timestamp (seconds) for the last update in general. This field is updated for any new, valid message received from the transponder.
5	longitude	float	WGS-84 longitude in decimal degrees. Can be null.
6	latitude	float	WGS-84 latitude in decimal degrees. Can be null.
7	baro_altitude	float	Barometric altitude in meters. Can be null.
8	on_ground	boolean	Boolean value which indicates if the position was retrieved from a surface position report.
9	velocity	float	Velocity over ground in m/s. Can be null.
10	true_track	float	True track in decimal degrees clockwise from north (north=0°). Can be null.
11	vertical_rate	float	Vertical rate in m/s. A positive value indicates that the airplane is climbing, a negative value indicates that it descends. Can be null.
12	sensors	int[]	IDs of the receivers which contributed to this state vector. Is null if no filtering for sensor was used in the request.

Index	Property	Type	Description
13	geo_altitude	float	Geometric altitude in meters. Can be null.
14	squawk	string	The transponder code aka Squawk. Can be null.
15	spi	boolean	Whether flight status indicates special purpose indicator.
16	position_source	int	Origin of this state's position. ● 0 = ADS-B ● 1 = ASTERIX ● 2 = MLAT ● 3 = FLARM
17	category	int	Aircraft category. ● 0 = No information at all ● 1 = No ADS-B Emitter Category Information ● 2 = Light (< 15500 lbs) ● 3 = Small (15500 to 75000 lbs) ● 4 = Large (75000 to 300000 lbs) ● 5 = High Vortex Large (aircraft such as B-757) ● 6 = Heavy (> 300000 lbs) ● 7 = High Performance (> 5g acceleration and 400 kts) ● 8 = Rotorcraft ● 9 = Glider / sailplane ● 10 = Lighter-than-air ● 11 = Parachutist / Skydiver

Index	Property	Type	Description
			• 12 = Ultralight / hang-glider / paraglider • 13 = Reserved • 14 = Unmanned Aerial Vehicle • 15 = Space / Trans-atmospheric vehicle • 16 = Surface Vehicle – Emergency Vehicle • 17 = Surface Vehicle – Service Vehicle • 18 = Point Obstacle (includes tethered balloons) • 19 = Cluster Obstacle • 20 = Line Obstacle

Table 3: JSON property of states

2.2.21. FIRMS

A category of data to be visualized in the application is fire data. For this purpose, data from the FIRMS system will be used. The name FIRMS stands for Fire Information for Resource Management System, developed by NASA, which provides near-real-time data on active fires (11). It uses satellite monitoring to track fire hotspots and transmits the data via email, datasets, online maps, and web services. We will utilize the available web services[22].

Among the scientific community, there is a record of fire data for scientific and research purposes. FIRMS can produce data in ultra-real-time (URT), real-time (RT), and near-real-time (NRT) formats to cover fire detection needs. For URT and RT, data is received directly from satellites, while NRT data is first stored and processed before distribution. Specifically, FIRMS data is updated frequently: NRT data is available within three hours of observation, RT data is available within thirty minutes, and URT data is available within five minutes of detection.

[22] https://firms.modaps.eosdis.nasa.gov/

Interactive maps on the website are updated every five minutes, while files provided in formats such as SHP, KML, and CSV are updated every sixty minutes. Currently, URT and RT data are available only for the United States and Canada, while NRT data is sent for all other regions.

All information is provided via satellites from the MODIS and VIIRS sensors, which are capable of detecting active fires and thermal anomalies. Data from the Moderate Resolution Imaging Spectroradiometer (MODIS) is distributed through the Aqua and Terra satellites, while data from the Visible Infrared Imaging Radiometer Suite (VIIRS) is distributed through the S-NPP, NOAA-20, and NOAA-21 satellites.

Satellites capture snapshots during their overpasses of the Earth. Each hotspot or active fire detected represents the center of a pixel, indicating the presence of one or more fires or some thermal anomaly such as a volcanic crater, a hot chimney from a factory, or an ongoing forest fire. For MODIS sensors, the pixel size is one kilometer (1 km), while for VIIRS sensors, the pixel size is 375 meters (375 m). The location of detection is at the center of the pixel and not necessarily the coordinates of the fire's pixel placement. Typically, the fire size is smaller than the pixel size, and if we observe a series of pixels, it represents a fire front.

Fire detection is carried out using a contextual algorithm that identifies strong radiative emissions from fires or heat sources. As mentioned, each pixel's reflection is detected in degrees Kelvin, which means that the emission in a specific wavelength is converted into temperature through appropriate processing. This method can detect fires as small as 1000 m^2, and under ideal conditions, it can detect hotspots as small as 100 m^2. The MODIS sensor performs the algorithm on each pixel during its search and provides one of the following indications:

- Missing data
- Cloud
- Water
- Non-fire
- Fire
- Unknown

The VIIRS sensor is an evolution of MODIS and, in addition to performing the same detection and categorization logic, includes a hybrid threshold detection capability for radiation bands to reject false fire indications, thereby returning more accurate data. In Figure 4, the threshold for fire detection and categorization under various conditions is shown diagrammatically. Figure 5 and 6 illustrate how each element is detected and marked on the map provided on the FIRMS website[23]. Understanding this is essential to comprehend the structure of the data sent through the API.

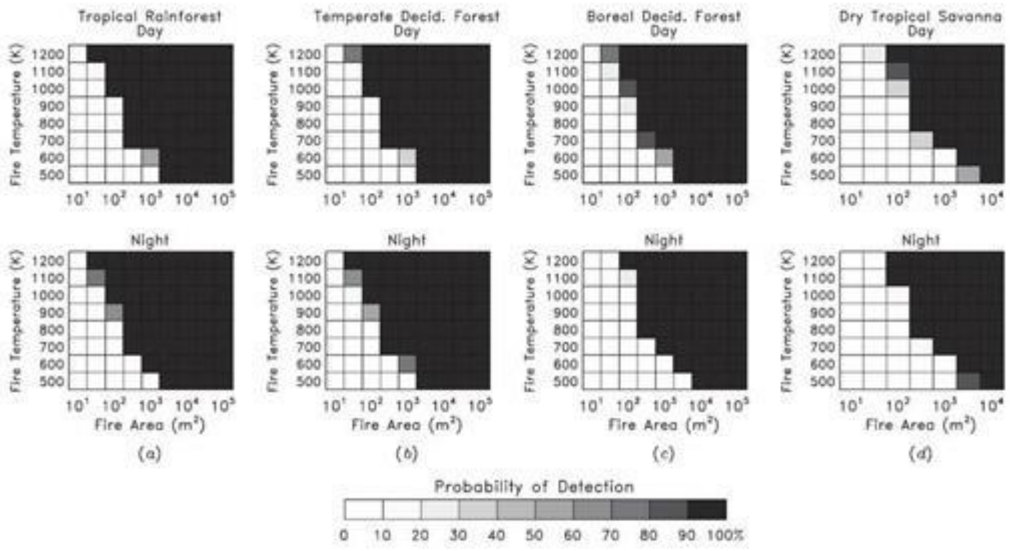


Figure 4: Fire detection threshold

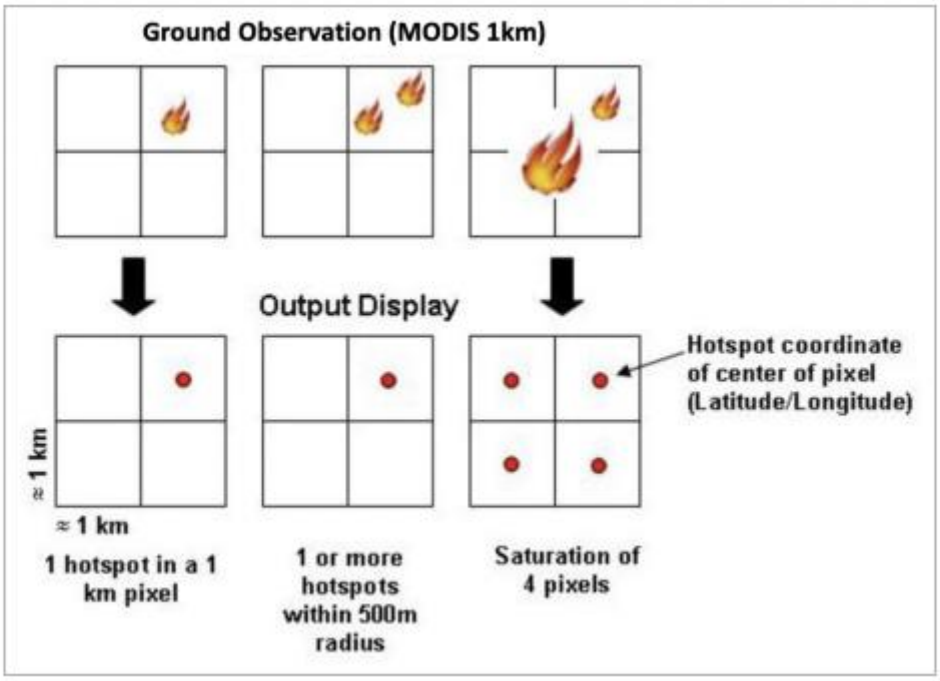

Ground Observation (MODIS 1km)
Output Display
Hotspot coordinate of center of pixel (Latitude/Longitude)
≈ 1 km
≈ 1 km
1 hotspot in a 1 km pixel
1 or more hotspots within 500m radius
Saturation of 4 pixels

Figure 5: MODIS detection process[24]

[23] https://firms.modaps.eosdis.nasa.gov/
[24] https://www.earthdata.nasa.gov/faq/firms-faq#ed-fire-on-ground

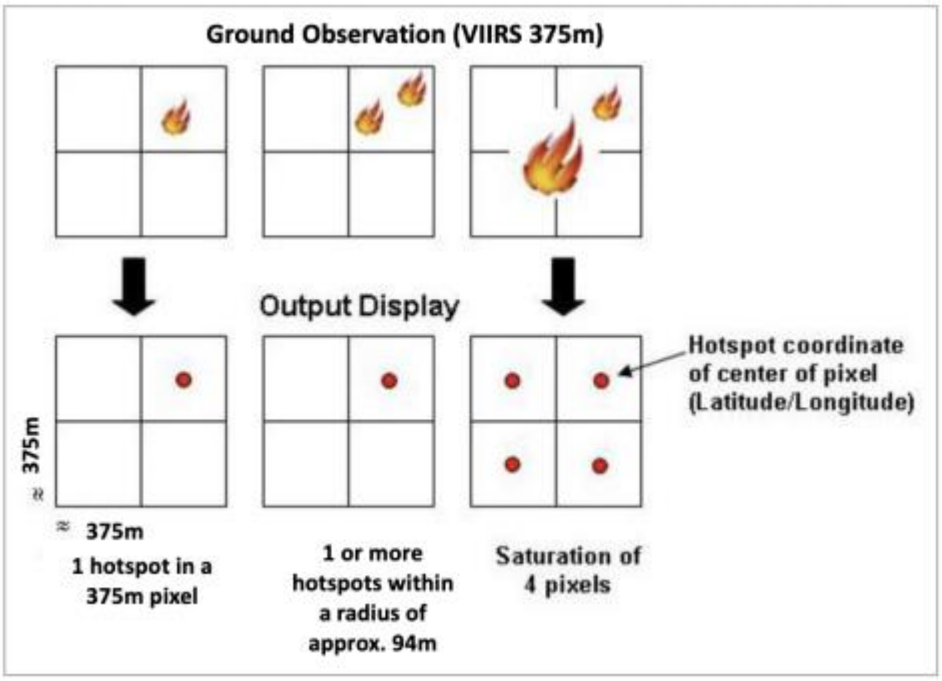

Ground Observation (VIIRS 375m)
Output Display
Hotspot coordinate of center of pixel (Latitude/Longitude)
≈ 375m
≈ 375m
1 hotspot in a 375m pixel
1 or more hotspots within a radius of approx. 94m
Saturation of 4 pixels

Figure 6: VIIRS detection process[25]

The MODIS sensors provide continuous coverage, ensuring full Earth coverage within one to two days with a recording range of 2,330 kilometers. Terra passes daily at 10:30 PM and 10:30 AM Mean Local Time (MLT), while Aqua passes at 1:30 AM and 1:30 PM, resulting in at least four passes and consequently coverage by MODIS for each area at the Equator. This number increases as we move towards the poles due to overlap.

The VIIRS sensors on Suomi NPP and NOAA-20/21 also record data continuously. Their 3,040-kilometer recording range allows for about 15% overlap between successive orbits, achieving full Earth coverage within twelve hours. Suomi NPP passes the Equator at 1:30 AM and 1:30 PM MLT, while NOAA-20/21 passes at 12:40 AM and 12:40 PM, with a lead time of approximately 50 minutes ahead of Suomi. The orbital path allows for coverage of mid-latitudes three to four times a day. Figure 7 shows a typical Earth surface coverage from a satellite, with batch size and footprint varying based on the type of orbit and the satellite's field of view.

[25] https://www.earthdata.nasa.gov/faq/firms-faq#ed-fire-on-ground

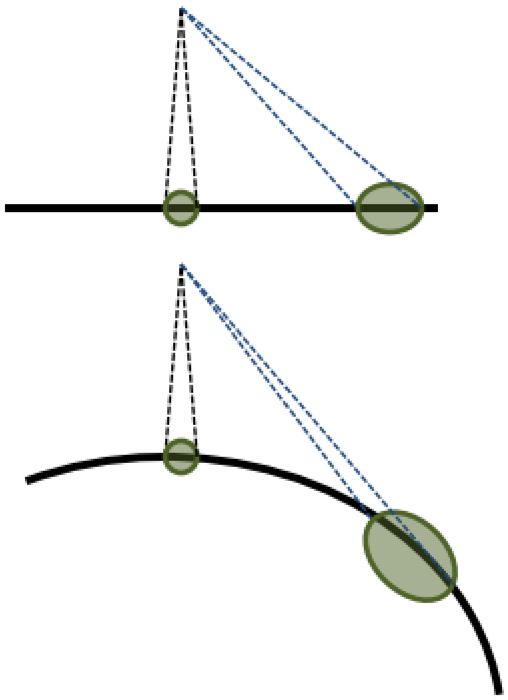


Figure 7: Satellite footprint and surface scanning[26]

In the context of this report, it is worth noting that the positional difference between MODIS and VIIRS NRT detections is usually less than 100 meters, but there have been cases where the difference reached the order of kilometers. The main causes of such cases are usually satellite maneuvers or the effects of space weather changes. Generally, data accuracy decreases with satellite movement, but for the MODIS/Terra, VIIRS/Suomi NPP, VIIRS/NOAA-20, and VIIRS/NOAA-21 systems, the period during which we can receive reduced performance data is less than two hours, unlike MODIS/Aqua, where the duration can be up to twelve hours. The standard data published are properly processed and provided after a considerable time, corrected for position and satellite movement errors, with the resulting data being of level 2.

The following tables illustrate the JSON data returned in each query response, as well as an explanation of each value.

Attribute	**Short Description**	**Long Description**
Latitude	Latitude	Center of nominal 375 m fire pixel

[26] chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://modis-fire.umd.edu/files/MODIS_C6_C6.1_Fire_User_Guide_1.0.pdf

Attribute	Short Description	Long Description
Longitude	Longitude	Center of nominal 375 m fire pixel
Bright_ti4	Brightness temperature I-4	VIIRS I-4 channel brightness temperature of the fire pixel measured in Kelvin
Scan	Along Scan pixel size	The algorithm produces approximately 375 m pixels at nadir. Scan and track reflect actual pixel size
Track	Along Track pixel size	The algorithm produces approximately 375 m pixels at nadir. Scan and track reflect actual pixel size
Acq_Date	Acquisition Date	Date of VIIRS acquisition
Acq_Time	Acquisition Time	Time of acquisition/overpass of the satellite (in UTC)
Satellite	Satellite	N= Suomi National Polar-orbiting Partnership (Suomi NPP), N20= NOAA-20 (formally JPSS-1), N21=NOAA-21
Confidence	Confidence	This value is based on a collection of intermediate algorithm quantities used in the detection process. It is intended to help users gauge the quality of individual hotspot/fire pixels. Confidence values are set to low, nominal, and high. Low confidence daytime fire pixels are typically associated with areas of Sun glint and lower relative temperature anomaly (<15 K) in the mid-infrared channel I4. Nominal confidence pixels are those free of potential Sun glint contamination during the day and marked by strong (>15 K) temperature anomaly in

Attribute	Short Description	Long Description
		either day or nighttime data. High confidence fire pixels are associated with day or nighttime saturated pixels. Please note: Low confidence nighttime pixels occur only over the geographic area extending from 11° E to 110° W and 7° N to 55° S. This area describes the region of influence of the South Atlantic Magnetic Anomaly which can cause spurious brightness temperatures in the mid-infrared channel I4 leading to potential false positive alarms. These have been removed from the NRT data distributed by FIRMS.
Version	Version (collection and source)	Version identifies the collection (e.g., VIIRS Collection 1 or VIIRS Collection 2), and source of data processing (Ultra Real-Time (URT suffix added to collection), Real-Time (RT suffix), Near Real-Time (NRT suffix) or Standard Processing (collection only). For example: "2.0URT" - Collection 2 Ultra Real-Time processing "2.0RT" - Collection 2 Real-Time processing "1.0NRT" - Collection 1 Near Real-Time processing "1.0" - Collection 1 Standard processing
Bright_ti5	Brightness temperature I-5	I-5 Channel brightness temperature of the fire pixel measured in Kelvin
FRP	Fire Radiative Power	FRP depicts the pixel-integrated fire radiative power in megawatts (MW).

Attribute	Short Description	Long Description
		Given the unique spatial and spectral resolution of the data, the VIIRS 375 m fire detection algorithm was customized and tuned to optimize its response over small fires while balancing the occurrence of false alarms. Frequent saturation of the mid-infrared I4 channel (3.55-3.93 µm) driving the detection of active fires requires additional tests and procedures to avoid pixel classification errors. As a result, sub-pixel fire characterization (e.g., fire radiative power [FRP] retrieval) is only viable across small and/or low-intensity fires. Systematic FRP retrievals are based on a hybrid approach combining 375 and 750 m data. In fact, starting in 2015 the algorithm incorporated additional VIIRS channel M13 (3.973-4.128 µm) 750 m data in both aggregated and unaggregated format.
Type*	Inferred hot spot type	0 = presumed vegetation fire 1 = active volcano 2 = other static land source 3 = offshore detection (includes all detections over water)

Table 4: Attribute table for VIIRS

Attribute	Short Description	Long Description
Latitude	Latitude	Center of 1 km fire pixel, but not necessarily the actual location of the fire as one or more fires can be detected within the 1 km pixel.
Longitude	Longitude	Center of 1 km fire pixel, but not necessarily the actual location of the fire as one or more fires can be detected within the 1 km pixel.
Brightness	Brightness temperature 21 (Kelvin)	Channel 21/22 brightness temperature of the fire pixel measured in Kelvin.
Scan	Along Scan pixel size	The algorithm produces 1 km fire pixels, but MODIS pixels get bigger toward the edge of scan. Scan and track reflect actual pixel size.
Track	Along Track pixel size	The algorithm produces 1 km fire pixels, but MODIS pixels get bigger toward the edge of scan. Scan and track reflect actual pixel size.
Acq_Date	Acquisition Date	Data of MODIS acquisition.
Acq_Time	Acquisition Time	Time of acquisition/overpass of the satellite (in UTC).
Satellite	Satellite	A = Aqua and T = Terra.
Confidence	Confidence (0-100%)	This value is based on a collection of intermediate algorithm quantities used in the detection process. It is intended to help users gauge the quality of individual hotspot/fire pixels. Confidence estimates range between 0 and 100% and are assigned one of the three fire classes (low-confidence fire, nominal-confidence fire, or high-confidence fire).

Version	Version (Collection and source)	Version identifies the collection (e.g., MODIS Collection 6.1) and source of data processing (Ultra Real-Time (URT suffix added to collection), Real-Time (RT suffix), Near Real-Time (NRT suffix) or Standard Processing (collection only). For example: "6.1URT" - Collection 6.1 Ultra Real-Time processing. "6.1RT" - Collection 6.1 Real-Time processing. "6.1NRT" - Collection 61 Near Real-Time processing. "6.1" - Collection 61 Standard processing. Find out more on collections and on the differences between FIRMS data sourced from LANCE FIRMS and the University of Maryland.
Bright_T31	Brightness temperature 31 (Kelvin)	Channel 31 brightness temperature of the fire pixel measured in Kelvin.
FRP	Fire Radiative Power (MW - megawatts)	Depicts the pixel-integrated fire radiative power in MW (megawatts).
Type*	Inferred hot spot type	0 = presumed vegetation fire 1 = active volcano 2 = other static land source 3 = offshore

Table 5: Attribute table for MODIS

It should be noted that the confidence levels have the following differences. For MODIS, the level is expressed as a percentage ranging from 0 to 100 and can be used to categorize fires into low-confidence, nominal-confidence, or high-confidence fires across all pixels. For VIIRS, the confidence levels are categorized as low, nominal, and high. In the low category, the data might include sun reflections or temperatures emitted in channel I4. In the nominal category, the detection is either day or night with emissions greater than 15K. In the high category, the data includes saturated pixels. Additionally, the value scan corresponds to the east-west direction, and the value track corresponds to the north-south direction. The nominal pixel size refers only to the vertical position of the satellite relative to the Earth.

Chapter 3: Application Design and Implementation

In this chapter, the design architecture and implementation of the application will be described, as well as the way in which all the necessary components are combined to provide the required functionality.

3.1. Scenario

The application is designed to operate on a central server and run continuously, with interaction required only for maintenance purposes. The server provides services to workstations that are connected to a private network. Each workstation is managed by authorized personnel responsible for handling resources designated for the execution of civil protection missions.

The application is a map on which data is displayed, while notifications and real-time information are also shown. Users can navigate the map and choose the amount of data displayed, adjusting the declutter level accordingly. From a toolbar, users can proceed with drawing and editing shapes, which are stored in the database and instantly shared with other users. They can also visualize real time tracks and send command-type messages to other users. Additionally, live chat functionality is available to facilitate user communication.

3.2. Use case diagram

Before proceeding with the architectural analysis, it is essential to provide a detailed description of the application's desired functionalities, as mentioned above, to clarify how users will interact with the application. The following figure provides a concise representation of the application's use, while this section will also present the most important use cases individually.

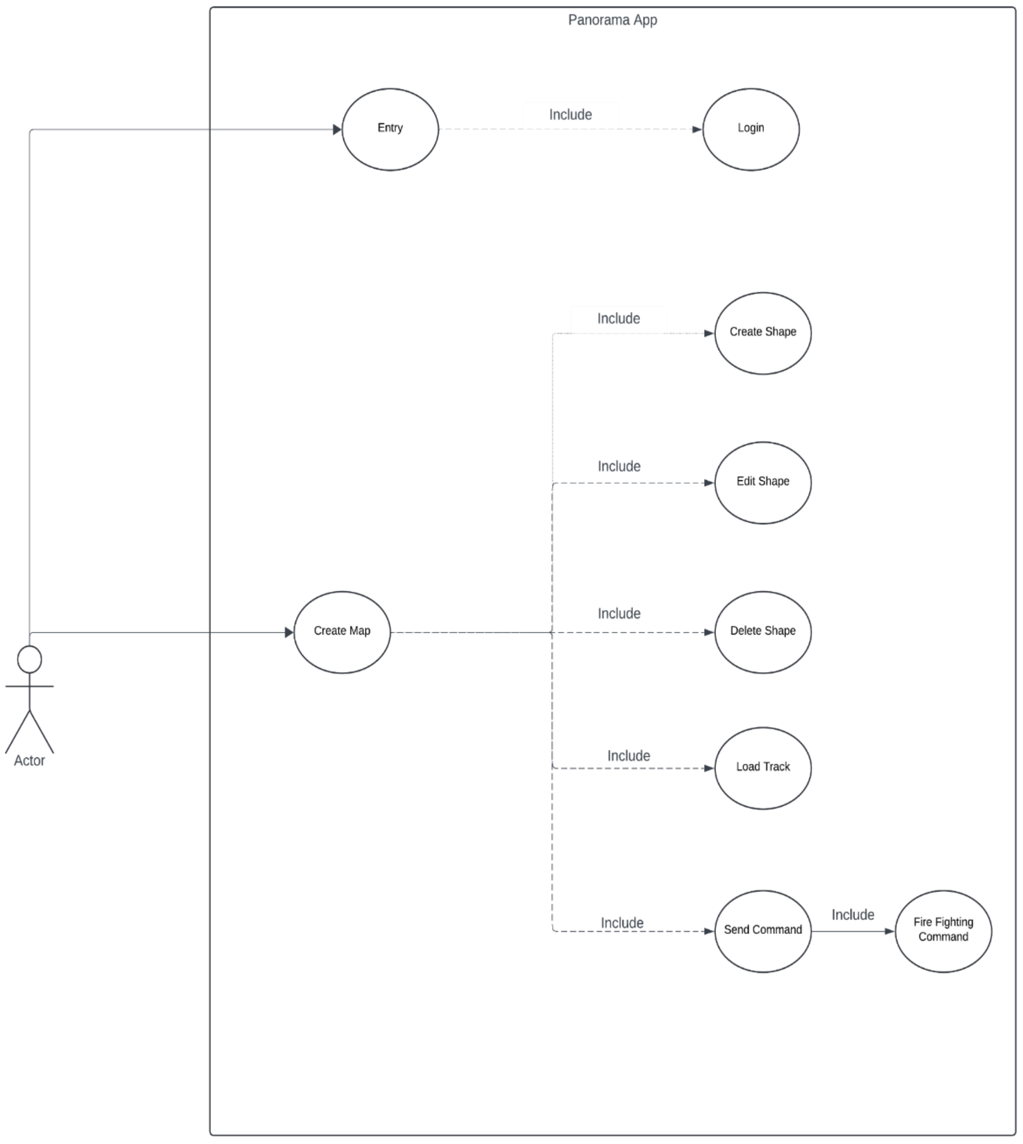


Figure 8: Panorama Application use case diagram

3.3. Use cases

Below are the most important use cases of the application:

Use Case - Title	User Login	
Brief description	The user is logging into the system.	
Actors	User	
Flow/Steps	**Steps/ Tasks**	**Data**
	The user visits the login web page	

	The user enters the username and password	Username, password
	User clicks "login"	
	The system checks the connection details and if they are correct the connection is achieved.	
Alternative Flow/Steps	User clicks "login"	
	The system checks the connection details and if they are not correct, it is informed to correct them.	

Table 6: Login use case

Use Case - Title	Edit shapes	
Brief description	The user creates and edits the desired shapes on the map.	
Actors	User	
Flow/Steps	**Steps/ Tasks**	**Data**
	The user selects the toolbar	
	The user selects the desired shape they wish to create.	shape
	The user creates the shape	
	The user fills in the name and description fields and presses save	shape
Alternative Flow/Steps	The shape is stored in the database and displayed on the map	
	The user selects the toolbar	
	The user selects the “edit” option	
	The user edits the desired shapes and when finished selects save	
	The processed shapes are stored in the database and displayed on the map.	
	The user selects the toolbar	

Alternative Flow/Steps	The user selects the “delete” option	
	The user selects the desired shapes and when finished selects save	
	The selected shapes are deleted from the database.	

Table 7: Create/edit use case

Use Case - Title	Sending an aerial firefighting command	
Brief description	The user sends aerial firefighting commands from a responsible unit.	
Actors	User	
Flow/Steps	**Steps/ Tasks**	**Data**
	The user selects the toolbar	
	The user selects the special button of the aerial firefighting command.	
	The user selects from the options menu the airport to operate, the type of mission and the location of the fire.	Airport, Fire
	The system sends a notification message to all users	
	The message waits until the command is accepted by a user.	
	The user accepts the aerial firefighting command	

Table 8: Firefighting command use case

3.4. Design Architecture

The application is a web map with the capability to display dynamic data as well as the ability to create real-time maps. Below, the image shows the general architecture of the application, which will be analyzed further.

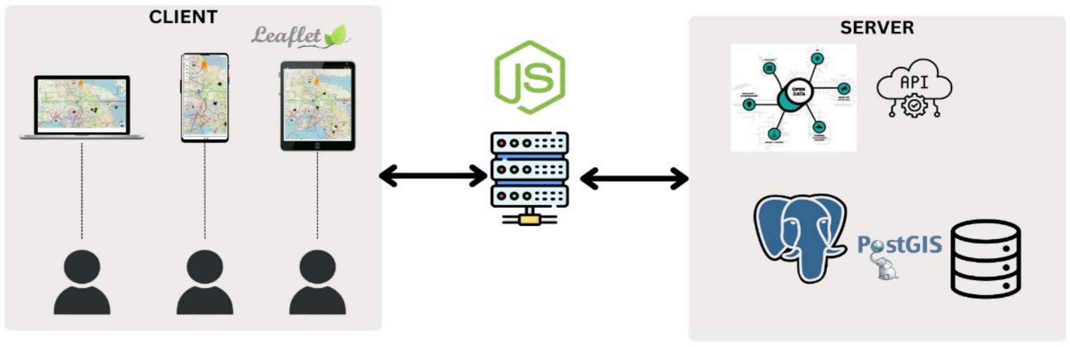


Figure 9: Panorama application architecture

Initially, in the broader sense of the big picture, the following structure can be observed on a case-by-case basis:

- **Client-side:** The functionality of map rendering in the frontend is handled by the Leaflet library, which can deliver the desired results on PCs, laptops, tablets, and smartphones.
- **Backend:** The backend functionality on the server-side is handled by Node.js, which manages the interoperability between the application and the data, as well as between the application and the users.
- **Database:** Map data is stored, managed, and processed by the PostgreSQL database, with the simultaneous use of the PostGIS extension.
- **External Services:** External data provision services are utilized via APIs to optimize the application's functionalities.

The application generally follows the CRUD operation philosophy, with the acronym representing the functions of Create, Read, Update, and Delete. In the application, users will have the ability to create data and access it through the UI, which in this specific application will be done via the map. They will also be able to execute updates on the data or perform edit operations, and finally, they will be able to delete any data they wish. Each of the above functions corresponds to a specific HTTP request method: for the Create function, we have POST; for the Read function, we have GET; for the Update function, we have PUT/PATCH; and finally, for the Delete function, we have DELETE.

The application, using Node.js and the Express web framework, will follow and support the MVC pattern, which is an architectural approach for developing web servers that ensures reliability in operation while facilitating application development. In this architecture, the web server separates the following functions:

- Controller: This component is responsible for managing client requests and providing appropriate responses to each query.
- Model: This component handles the interface with the database, such as creating the database schema, storing data, or retrieving it.
- View: This component is responsible for the rendering process and the visualization of the corresponding data.

An additional feature of the MVC pattern structure is the separation and assignment of each function to a distinct component. Below is the application's structure to provide a better understanding of its organization, while the operation and usage of each component will be discussed in detail.

```
assets
│   ├── Civilairportwithcontroltowerbig.png
│   ├── airplane.png
│   ├── cndr.png
│   ├── fire.png
│   └── landing_haf_low.png
├── bin
│   └── www
├── data
│   └── airports.js
├── database
│   ├── config
│   ├── logging
│   ├── migrations
│   ├── models
│   └── seeders
├── mapUtils
│   ├── fileUploader.js
│   ├── fireFightingCommands.js
│   ├── loadADSB.js
│   ├── loadFireData.js
│   ├── loadFlightData.js
│   ├── mapDataUtils.js
│   ├── mapDrawControllers.js
│   ├── mapOverlays.js
│   ├── notifications.js
│   └── timeStamp.js
├── public
│   ├── assets
│   ├── javascripts
│   ├── stylesheets
│   ├── index.html
│   └── login.html
├── routes
│   ├── drawings.js
│   ├── fireRouter.js
│   ├── flightRouter.js
│   ├── index.js
│   ├── openSky.js
│   ├── shapes.js
│   └── users.js
├── README.md
├── app.js
├── auth.js
├── bundle.js
├── config.dev.js
├── config.js
├── config.prod.js
├── imageUrls.js
├── package-lock.json
├── package.json
├── sequelize.log
└── webmap.js
```

Table 9: Panorama application structure

In summary, the functionality of the Model is integrated into the database directory, which includes the entities and database operations. The View functionality is handled by the Browserify library and is reflected in the public directory, while the Controller is represented by the webmap.js file in conjunction with the routes directory. In the context of developing the architecture using the MVC pattern and applying the Clean Code model, it should be noted that data management from sources is conducted through the router functions, while mapUtils contains tools for every necessary operation.

Below is the method of use and application of each tool in order to complete the development of the application.

3.5. VS code

For the development of the application, there is a plethora of platforms designed to assist users and facilitate processes, allowing them to easily focus on coding. In this specific application, VS Code[27] will be used, and the version in use is 1.92.1. It is a widely used source code editor that can run on various operating systems, supports JavaScript without requiring installation, and also supports Node.js. Additionally, it allows users to browse and find any desired tool through various extensions provided in its library, which can be installed automatically. Finally, it enables interaction and execution of commands in terminals within its interface, which is particularly useful for installing and using programs. The installation instructions followed are for the Windows operating system.

[27] https://code.visualstudio.com/

3.6. Git Bash

To install Git[28], a visit was made to the official Git website, and the download for the Windows operating system was selected. Once the download was completed, the .exe file was executed, and Git Bash is now installed on the computer. It should be noted that this will now be the primary means of interaction with the operating system, and all tools will be installed and managed through Git Bash. The version in use is git version 2.39.1.windows.1.

3.7. NVM

To install NVM[29], we follow the instructions available in the library's GitHub repository, using the *wget* option, which is executed in the Git Bash environment. More detailed instructions are provided in the next paragraph. The version used for the development of the application is v20.9.0.

3.8. Node.js

For the development of the application, the CommonJS module system will be used, which is suitable and recommended for server-side coding in a Node.js environment. Another option is to use the ECMAScript module, but it will not be discussed in the context of this work. Node.js can be installed in several ways; however, we will choose to do it via a package manager. A very common way to use a package manager is through NVM, which allows us to experiment with different versions of Node.js and manage them more effectively. Installation instructions can be found at the official link[30], where we can choose between two options. For the initial installation, the following code was executed.

```
# installs nvm (Node Version Manager)
wget -qO- https://raw.githubusercontent.com/nvm-
sh/nvm/v0.40.0/install.sh | bash

export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s
"${HOME}/.nvm" || printf %s "${XDG_CONFIG_HOME}/nvm")"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm


# verifies the right Node.js version is in the environment
node -v
```

[28] https://git-scm.com/download/win
[29] https://github.com/nvm-sh/nvm
[30] https://github.com/nvm-sh/nvm?tab=readme-ov-file

```
# verifies the right npm version is in the environment
npm -v
```

Table 10: Node.js installation

After the installation, it should be noted that we now have access to the Node executable program via the command line. Additionally, npm has been installed automatically. In this project, version 20.14.0 is used.

3.9. NPM

The use of npm[31] during the development of the application is extensive and frequent. It serves as the means for installing and utilizing all necessary functions, while also acting as a gateway for finding information and instructions related to each library. Below are some of the most important commands:

```
npm install
```

All packages listed in the package.json file are installed for use in the application, and the node_modules directory is created if it does not already exist. This command is useful in the case where the application needs to run on a new computer.

```
npm install <package-name>
```

One can install a specific package while also adding it to the package.json file. Of course, as with any tool or library, various options are available through the use of flags.

```
npm update
```

It performs a search for newer versions for all packages, and there is also the option to update only the specific package we want.

```
npm install <package-name>@<version>
```

We are also given the option to choose a specific version of a package.

```
npm run <task-name>
```

We have the ability to execute commands or combinations of them simply by using specific formats[32]. For this project, version 10.5.0 is used.

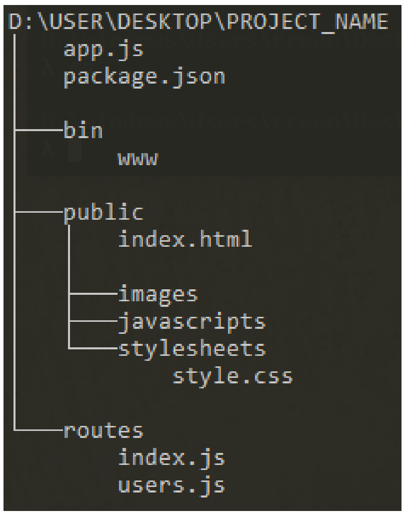
[31] https://docs.npmjs.com/cli/v10/configuring-npm/package-json
[32] https://www.naukri.com/code360/library/difference-between-npm-and-npx

3.10. Express

Once Node.js has been installed, the next step is to create a skeleton for the application on which the basic structure will be based. For this purpose, the Express framework will be used. The following describes this process.

First, we execute the command `npm init`, which will create the package.json file. During execution, corresponding prompts appear to create the desired structure. The main file is named app.js and serves as the core of the server's functionality. Next, the express-generator library is installed using the command `npm install -g express-generator`. This will help create the necessary folders with the appropriate settings for the project, essentially serving as a way to create a blueprint.

Once a directory is created where the application is desired to be developed, the following command is executed: `express <project_name> --no-view`, where <project_name> is the name of the project. Automatically, the following structure is created:



```
D:\USER\DESKTOP\PROJECT_NAME
│   app.js
│   package.json
│
├───bin
│       www
│
├───public
│   │   index.html
│   │
│   ├───images
│   ├───javascripts
│   └───stylesheets
│           style.css
│
└───routes
        index.js
        users.js
```

Figure 10: Express no view template

In more detail, and for better understanding, each function of the structure is referenced:

- app.js: This file contains the main code of the server.
- package.json: A JSON-formatted file that includes all the necessary dependencies and information about the application.
- bin: The folder contains executable code, specifically:

    - www: The startup file of the server.
- public: This directory contains all the static files and the necessary code for the frontend.
    - index.html: The main page of the application.
    - images: The folder containing the necessary images for the application.
    - javascripts: The folder contains the required JavaScript code for the frontend.
    - stylesheets: Contains the necessary CSS code files.
    - style.css: The file contains the CSS code for various cases.
- routes: The folder contains the appropriate files that respond to client requests.
- index.js: Configures how the application will behave in response to each client request and includes some configuration details.
- users.js: Defines the response to client requests at <address>/users, depending on how it is used in the app.js file.

It should be noted that the above structure was created by the express generator as a blueprint for the application and does not represent the final structure, which was mentioned in the architecture.

Next, we navigate to the project directory using `cd <project_name>` and install all the necessary libraries with the command `npm install`. The development of the application can now proceed smoothly. For this project, version 4.16.1 is used.

3.11. Browserify

The Browserify library will help serve the content to the frontend without requiring the use of a view engine, as none was installed. To use the library, we first install it with `npm install -g browserify`.

Since every change we make in the backend will also affect the frontend, we place the execution of Browserify in the package.json file as a build option. The JSON field will be as follows:

```
"scripts": {
    "start": "node ./bin/www",
    "build": "browserify webmap.js -o public/javascripts/webmap.js"
  },
```

Table 11: Browserify preset options

For the application, the backend file webmap.js will be bundled at the path ./public/javascripts/webmap.js. The version being used is 17.0.0.

3.12. Leaflet

In this paragraph, all the steps that will help create a basic web map will be described. First, the Leaflet library must be installed with the command `npm install --save leaflet`.

A folder is first created to contain the backend code, so the command `touch webmap.js` is executed. In the created webmap.js file, the following code is placed:

```
// Import the leaflet package
var L = require('leaflet');

// Creates a leaflet map binded to an html <div> with id "map"
// setView will set the initial map view to the location at
coordinates
// 13 represents the initial zoom level with higher values being more
zoomed in
var map = L.map('map').setView([43.659752, -79.378161], 20);

// Adds the basemap tiles to your web map
// Additional providers are available at: https://leaflet-
extras.github.io/leaflet-providers/preview/
L.tileLayer('https://{s}.basemaps.cartocdn.com/dark_all/{z}/{x}/{y}{r
}.png', {
      attribution: '&copy; <a
href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>
&copy; <a href="https://carto.com/attributions">CARTO</a>',
      subdomains: 'abcd',
      maxZoom: 21
}).addTo(map);

// Adds a popup marker to the webmap for GGL address
L.circleMarker([43.659752, -79.378161]).addTo(map)
      .bindPopup(
            'MON 304<br>' +
            'Monetary Times Building<br>' +
            '341 Victoria Street<br>' +
            'Toronto, Ontario, Canada<br>' +
            'M5B 2K3<br><br>' +
            'Tel: 416-9795000 Ext. 5192'
      )
      .openPopup();
```

Table 12: Leaflet basic structure and operation

In the file ./public/index.html, the following code is placed:

```
<html>

<head>
     <meta charset="utf-8"/>
     <title>GGL Leaflet Webmap Tutorial</title>
     <link rel="stylesheet" href="stylesheets/style.css">
</head>

<body>
     <div id="map"></div>
</body>


</html>
```

Table 13: Basic map rendering

An important point to note is the CSS files. Since the application is being developed locally, the CSS elements of each library must be moved to the frontend. For example, the CSS files of Leaflet can be copied as follows:

```
cp node_modules/leaflet/dist/leaflet.css
public/stylesheets/leaflet.css
```

Table 14: CSS styling method

Additionally, each CSS file should be included in the HTML file within the head tag.

```
<link rel="stylesheet" href="stylesheets/leaflet.css">
```

In more detail, in the backend file named webmap.js, the L object is initialized, which represents the map object. Everything that is added is an object with properties. In this way, the view, the tiles, and the circleMarker are added. In the HTML file, it is only necessary to add `<div id="map"></div>`, which will render the map. Lastly, it should be noted that the respective CSS file must be included for each library. The version of the Leaflet library used in this work is 1.9.4.

3.13. Postgres

To use the PostgreSQL[33] database, it must first be installed. The application is being developed on a Windows operating system, so the instructions for this specific OS are followed. The version chosen is 16.4, which is the most recent version at the time of writing this work.

3.13.1. Setting up pgAdmin

After installing PostgreSQL, we have also installed pgAdmin, a web-based interface for managing databases. We search for the pgAdmin program in the Start menu and open it. Then we follow the instructions.

3.14. Postgis

Next, PostGIS is installed, which enables processing and calculations with geospatial data. The instructions for the Windows operating system are followed. The most recent version of PostGIS at the time of writing this work is 3.4.2 and includes:

- PostGIS 3.4.2 with MVT support, with raster, GEOS 3.12.1, PROJ 8.2.1, SFCGAL support (1.5.1), address_standardizer, topology
- PostGIS Tiger geocoder extension - Tiger 2023
- pgRouting 3.6 pgRouting 3.6.2
- Commandline raster loader (raster2pgsql), shapefile import/export (shp2pgsql,pgsql2shp)
- Commandline osm2pgrouting 2.3.8 for loading data from .osm files into pgRouting routable format
- GUI: shp2pgsql-gui which has both import and export support for geometry/geography
- ogrfdw 1.1.4 - spatial foreign data wrapper for reading both spatial (spatial columns become postgis geometry) and non-spatial data. IMPORT FOREIGN SCHEMA support New features in 1.1 - character_encoding option and utility functions ogr_fdw_drivers(), ogr_fdw_version()
- GDAL 3.8.5 with OpenJPEG 2.5.20 (JPEG 2000), ODBC, Curl, SQLite3 (for GeoPackage and OSM support), excel (XLS) (via FreeXL 1.0.6), libreoffice, XLSX spreadsheet (via expat), Arrow 13.0 for Arrow, Parquet, and GeoParquet support (used by both PostGIS raster and ogrfdw)

[33] https://coding-boot-camp.github.io/full-stack/postgresql/postgresql-installation-guide

- pgpointcloud 1.2.4 for querying LIDAR point cloud and in/out functions to convert to PostGIS geometry
- h3-pg 4.1.3 for using Uber h3 api and converting h3 index representations to postgis geometry/geography/raster (raster support is new in this release).
- MobilityDB v1.1.1 for managing trajectories. Includes many temporal spatial types and functions for them. Refer to Enabling PostGIS Extensions: MobilityDb for guidance.

As can someone see it includes all the necessary components for the interoperability of the application.

3.15. Sequelize

For the development of the application and primarily for the structure of the data, the Code First technique was chosen. The reason is mainly that geospatial data can become very complex, so it is beneficial to create them more dynamically or during development. For this purpose, the Sequelize library and the sequelize-cli tool are first installed. The versions used are 6.37.3 and 6.6.2, respectively. Therefore, with the command `npm install --save sequelize`, the library is installed, and then the appropriate driver for the chosen database, which is Postgres, must be installed by executing the command `npm install pg pg-hstore`.

For executing all necessary processes, the use of the CLI is preferred, so all operations will be performed via commands. Referring to the structure of the database, the following can be observed:

```
├── database
│   ├── config
│   ├── logging
│   ├── migrations
│   ├── models
│   └── seeders
```

Table 15: Model construction

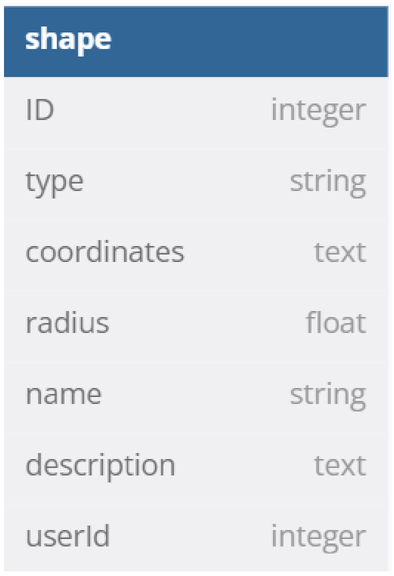
- config: This directory includes the config.js file, which contains basic settings for the database and its connection to the application. Its purpose is to avoid repeating code for the same processes and settings.
- logging: This directory contains the db_logger.js file, which is responsible for initializing and defining the database logging using the Winston library.

- migrations: This directory includes various migration files that are executed in the database as it changes based on the current requirements and data structure.
- models: This directory contains a main file called index.js, which includes all the elements and configurations necessary for Sequelize to communicate with the database, as well as folders corresponding to the entities being created.
- seeders: This directory contains files with data. It is mainly used during the development of the application to provide mock data for testing purposes.

In this section, it's important to note how the database was designed. The entity that is essential for the application's operation is generally a shape, which, aside from the classic forms it can take, can also become quite complex at times. To address this, the design philosophy is to create a very general entity that is managed accordingly by algorithms for the final rendering.

Specifically, in the shape entity, there may be characteristics such as radius, which might not apply to certain shapes like squares. Here, the application development approach manages such scenarios effectively to yield the desired result.

Below is the definition of the shape entity:



shape	
ID	integer
type	string
coordinates	text
radius	float
name	string
description	text
userId	integer

Figure 11: Shape entity

Here's a breakdown of the attributes in the shape entity:

- ID:
    - Managed automatically by Sequelize.
    - Maintains a sequential number for each created object, adjusting with every operation in the database.
- Type:
    - Specifies the type of the shape, which belongs to one of the categories supported by the GeoJSON structure.
    - Automatically generated based on the shape drawn by the user in the application, rather than specified by the user.
- Coordinates:
    - A list of X, Y pairs representing the vertices and edges of each shape.
    - Essential for accurately rendering the shape on the map.
- Radius:
    - Corresponds to circular shapes.
    - The application's business logic determines when this attribute is needed and when it should not be applied.
- Name:
    - A required field for the identification of each shape by the user, aiding in organization and retrieval.
- Description:
    - Allows users to enter comments and descriptions in free text for the created shape, providing context and additional information.
- UserId:
    - Enables the creation of relationships between users and shapes.
    - This field is included for proper entity creation but may not be utilized further in the application logic.

3.16. Data Layer

In this section, the management of data in the application is described, adhering to the principles of CRUD and MVC. Each CRUD operation corresponds to an HTTP method, making the use of Express routers a suitable approach for data management. Below is an analysis of the structure:

```
├── routes
│   ├── drawings.js
│   ├── fireRouter.js
│   ├── flightRouter.js
│   ├── index.js
│   ├── openSky.js
│   ├── shapes.js
│   └── users.js
```
Table 16: Controller construction

The application utilizes dedicated router files within the /routes directory to manage routes related to shapes and other entities.

Each router file defines the endpoints and connects them to the corresponding controller methods that execute the CRUD operations. Each route in the router is associated with a function in the controller, which contains the logic for interacting with the model and performing the required database operations.

For example, a controller function for creating a shape would validate the input, create a new shape record using Sequelize, and return the appropriate response to the client.

- drawings.js: This file manages the drawings entity, which is outside the scope of this particular work.
- fireRouter.js: Communicates with the API that provides fire data, retrieving information based on specified parameters such as area or satellite of interest. For Greece, the most suitable satellite for near-real-time data is the VIIRS NOAA20.
- flightRouter.js: Manages and retrieves ADS-B data related to military flights from open-source platforms. This router facilitates access to flight information relevant to military operations.
- index.js: Manages the View component of the MVC model. In this application, its use is necessary but is limited to essential functions.
- openSky.js: Handles and retrieves ADS-B data for all categories of flights except military-tagged flights. For this application, data retrieval is limited to the area within the Greek FIR (Flight Information Region).
- shapes.js: Manages the CRUD operations for the shapes data within the database. This router is critical for creating, reading, updating, and deleting shape records.

- users.js: Intended for future use, this file is not elaborated on further at this time.

3.17. Business Logic

In this section, the business logic that implements all the operational logic of the application will be described. Within the framework of separating functions and developing the application based on the principle of Clean Code, the business logic has been centralized in mapUtils as shown below:

```
├── mapUtils
│   ├── fileUploader.js
│   ├── fireFightingCommands.js
│   ├── loadADSB.js
│   ├── loadFireData.js
│   ├── loadFlightData.js
│   ├── mapDataUtils.js
│   ├── mapDrawControllers.js
│   ├── mapOverlays.js
│   ├── notifications.js
│   └── timeStamp.js
```

Table 17: Business logic - Map Custom Tools

More specifically:

- fileUploader.js: This file manages the function of visualizing files that log geospatial data from relevant devices. It supports file formats such as geojson, kml, json, and gpx.
- fireFightingCommands.js: This file takes on one of the application's most essential functions. It is responsible for the operation of commands and specifically manages the command sending menu, the visualization required for user updates, and the sending of appropriate messages.
- loadADSB.js: This file processes ADS-B data, separates the relevant properties, and performs the final visualization on the map.
- loadFireData.js: This file was created as a general tool for handling fire data processes and is used accordingly by various processes.
- loadFlightData.js: This file was created as a general tool for handling flight data processes and is used accordingly by various processes.
- mapDataUtils.js: This file is responsible for creating the data input form designed by the user. Additionally, it communicates with all necessary files to complete the CRUD actions performed by the user through the UI.

- mapDrawController.js: This file regulates the design parameters of the shapes.
- mapOverlays.js: This file manages the base maps used by the application.

notifications.js: This file manages the application's notification functionality.

timeStamp.js: This file manages the format of timestamps.

3.18. Execution of the Application

The application now has the desired structure and functions. With the files and directories organized as they are, the application can be executed for users to begin operating it. During development, specific commands were created for better management of the application, which are presented below:

```
"scripts": {
    "start": "node ./bin/www",
    "devstart": "nodemon ./bin/www",
    "dev": "nodemon app.js",
    "serverstart": "DEBUG=express-locallibrary-tutorial:* npm run
devstart",
    "build": "browserify webmap.js -o public/javascripts/webmap.js & cp
node_modules/leaflet/dist/leaflet.css public/stylesheets/leaflet.css & cp
node_modules/leaflet-draw/dist/leaflet.draw-src.css
public/stylesheets/leaflet-drw.css & cp node_modules/leaflet-
draw/dist/leaflet.draw.css public/stylesheets/leafletdrw-drw.css",
    "build:dev": "cp config.dev.js config.js && npm run build",
    "build:prod": "cp config.prod.js config.js && npm run build",
    "lint": "eslint app.js",
    "watch": "watch 'npm run build' .",
    "db:migrate": "npx sequelize-cli db:migrate",
    "db:reset": "npx sequelize-cli db:drop && npx sequelize-cli db:create
&& npx sequelize-cli db:migrate && npx sequelize-cli db:seed:all --seeders-
path ./database/seeders"
  }
```

Table 18: npm custom commands

For simple use, the command `npm build:prod` is executed first, followed by the command `npm start`. The application has been successfully started, and we can now begin working with it.

Chapter 4: System/Application Evaluation

The section that follows the analysis of requirements, the definition of architecture, and the development of the appropriate code is that of application evaluation. In this chapter, there will be a tour of the application's tools, and the application's functionalities will be presented as they were defined based on real events, as recorded during testing. Additionally, metrics for measuring the application's performance will be mentioned in order to evaluate the results.

4.1. Use of the Panorama Application

Login Page

Upon entering the application, the login page is displayed:

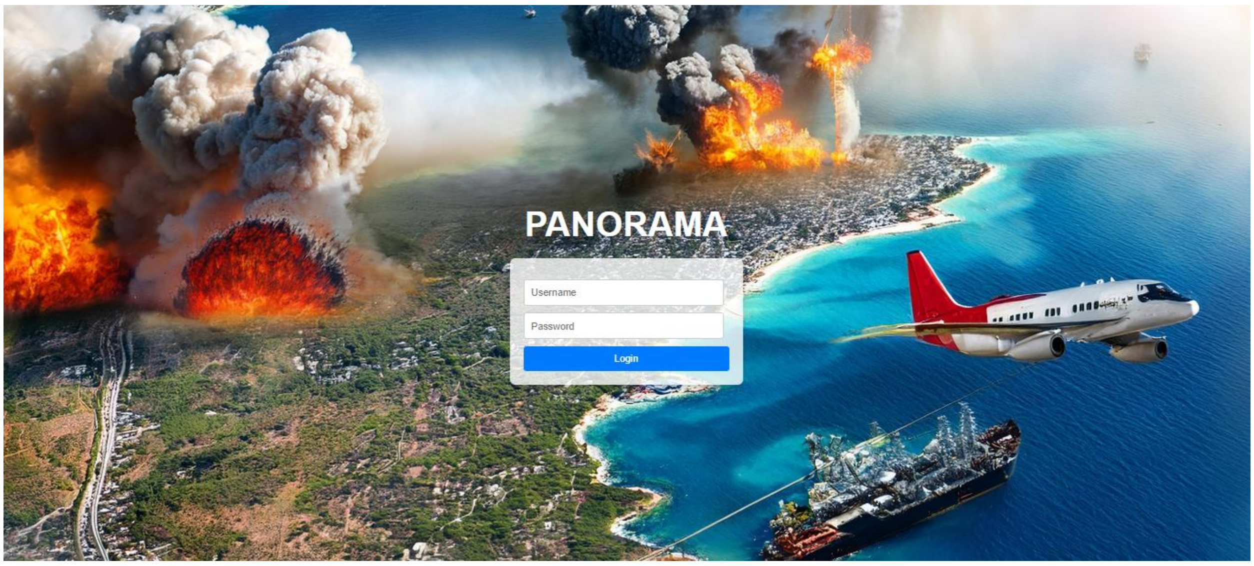


Figure 12: Panorama login page

By entering the appropriate credentials, access to the application is granted.

User Interactive Map

Upon logging into the application, the interactive map is displayed.

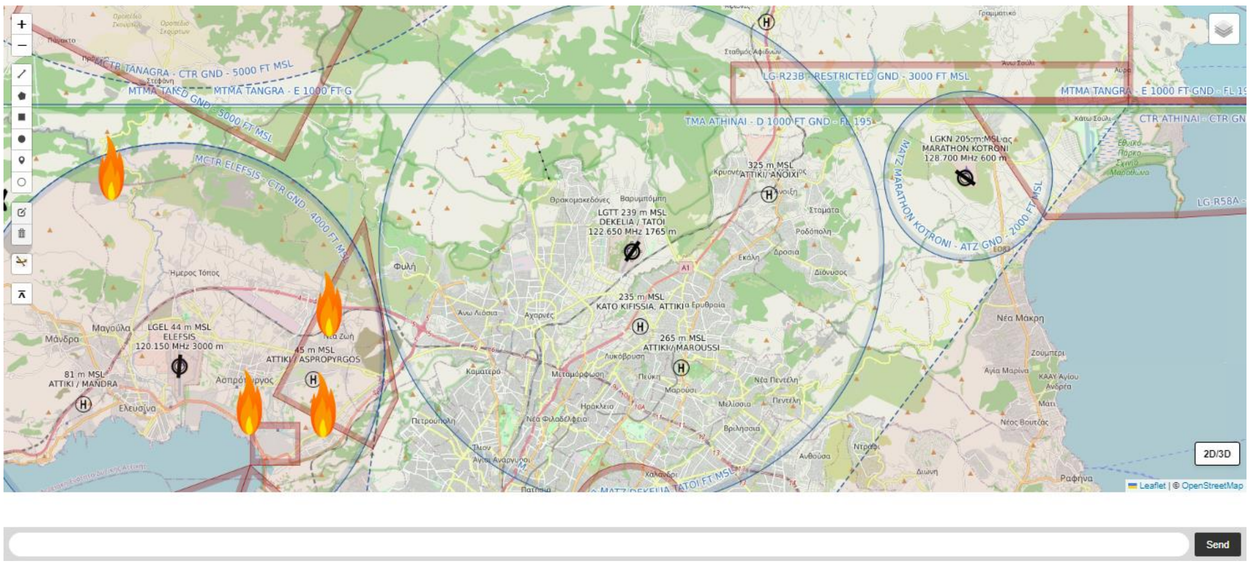
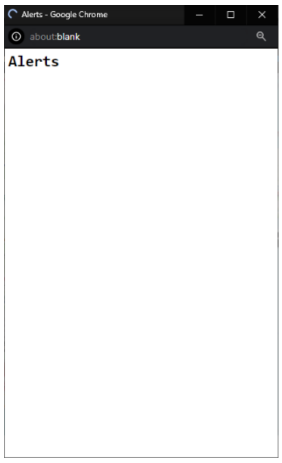


Figure 13: Panorama user interface

Additionally, a helpful window for notes and updates also appears.



Alerts

Figure 14: Alert message window

From the interactive map, we can distinguish the following functions:

Zoom Level

Map zoom capabilities from level 1 to level 18.

Tools Menu

From the tools menu, we have the following functions:

The polyline tool provides the ability to draw lines.

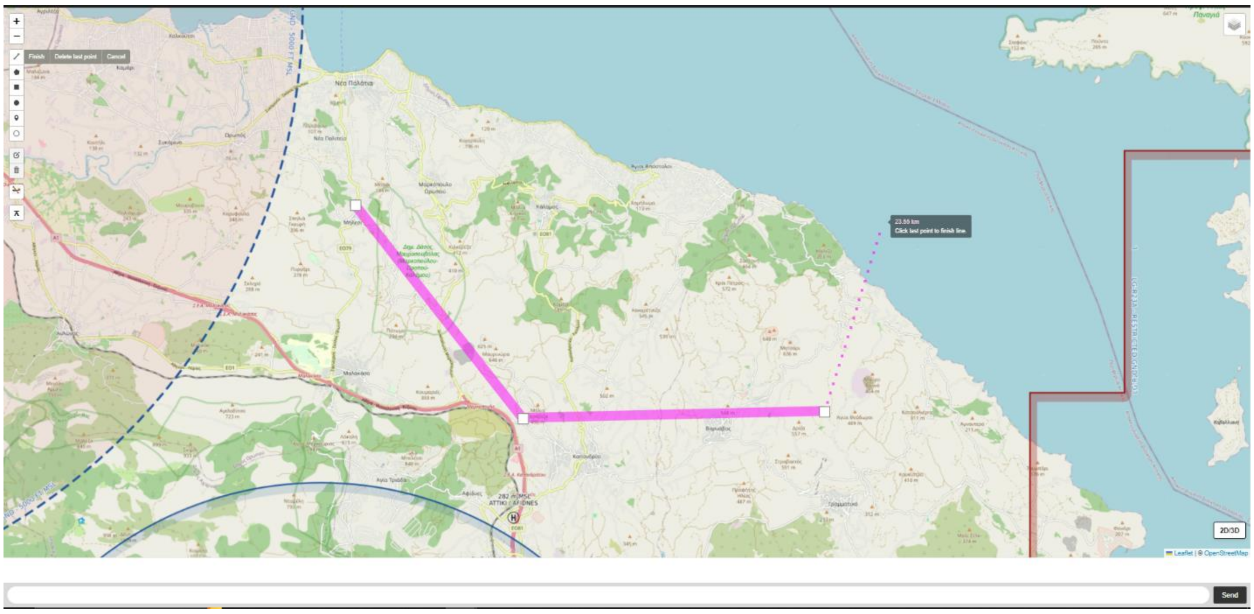


Figure 15: Polyline tool

The polygon tool allows for the drawing of polygons

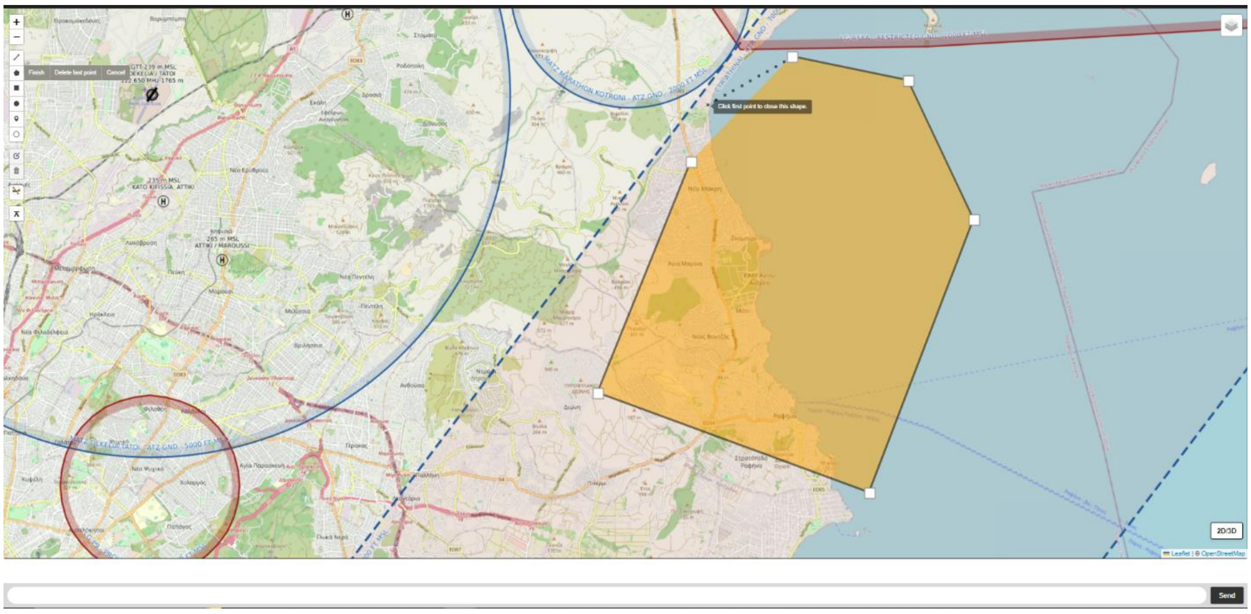


Figure 16: Polygon tool

■ The rectangle tool allows for the drawing of squares.

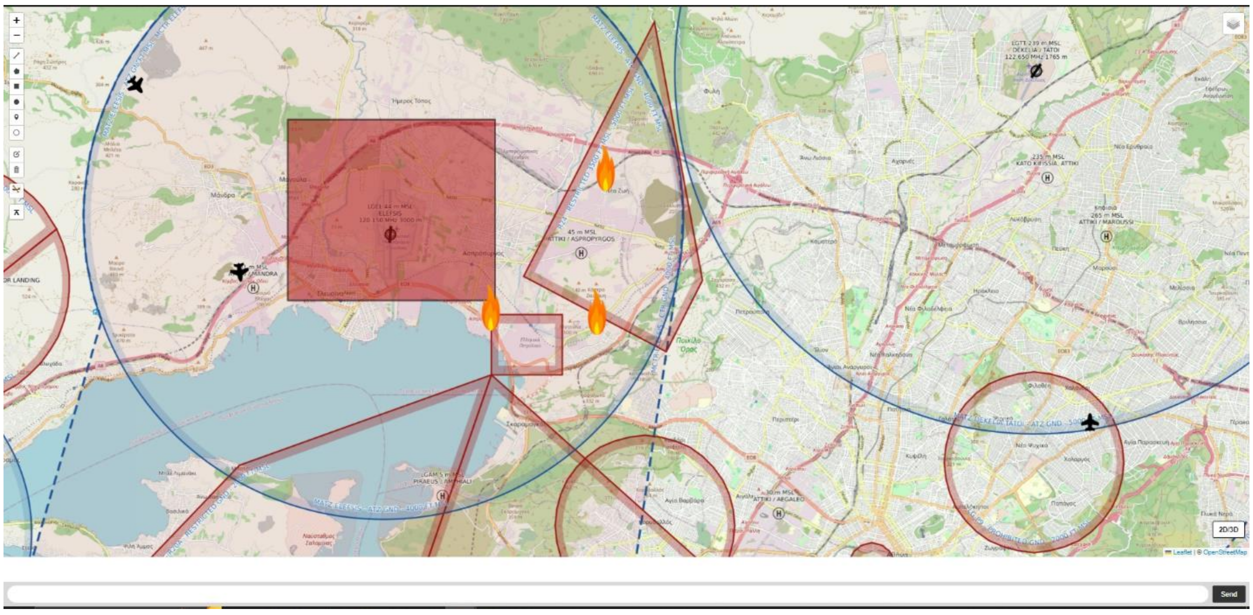


Figure 17: Rectangle tool

● The circle tool allows for the drawing of circles.

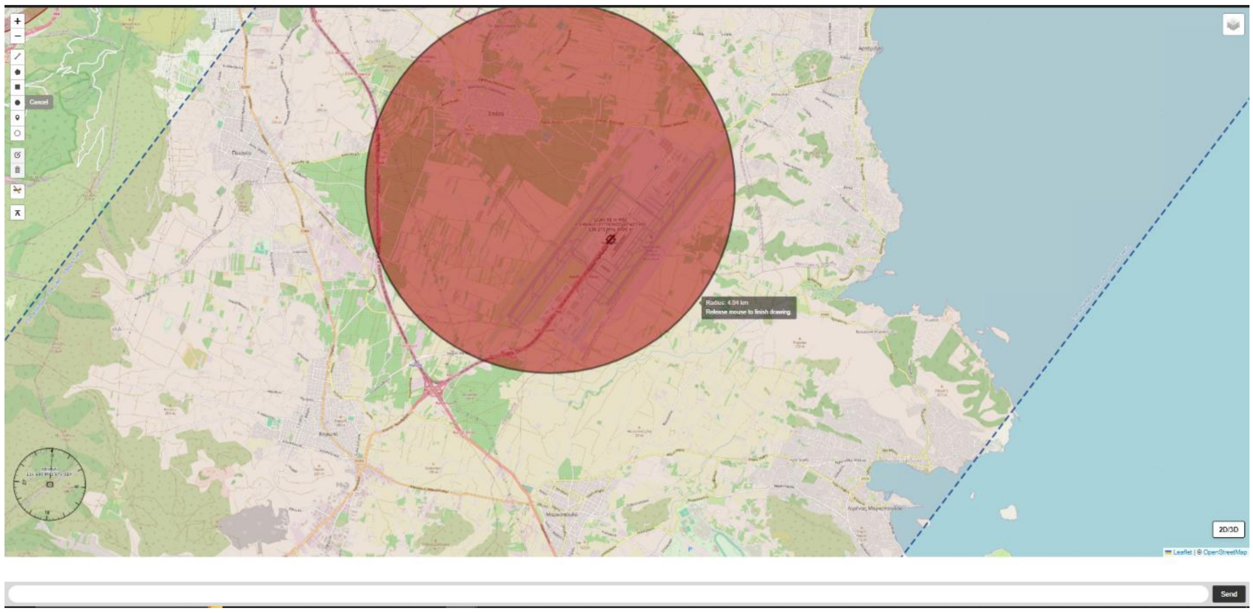


Figure 18: Circle tool

The marker tool allows for the drawing of point markers.

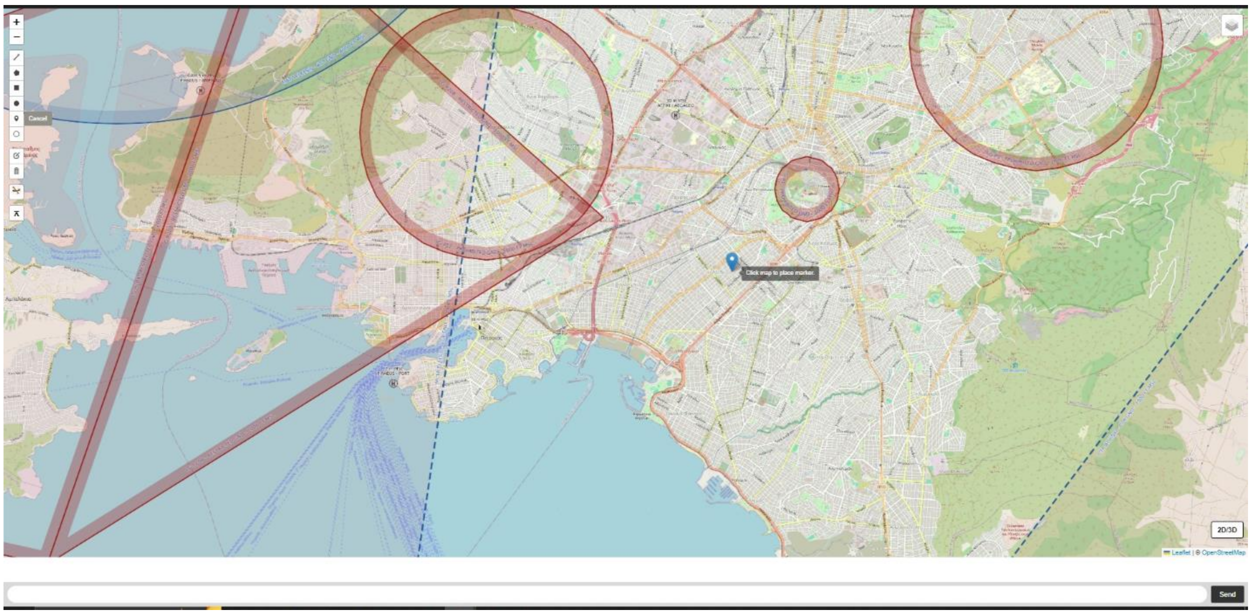


Figure 19: Marker tool

The circle marker tool allows for the drawing of circular markers.

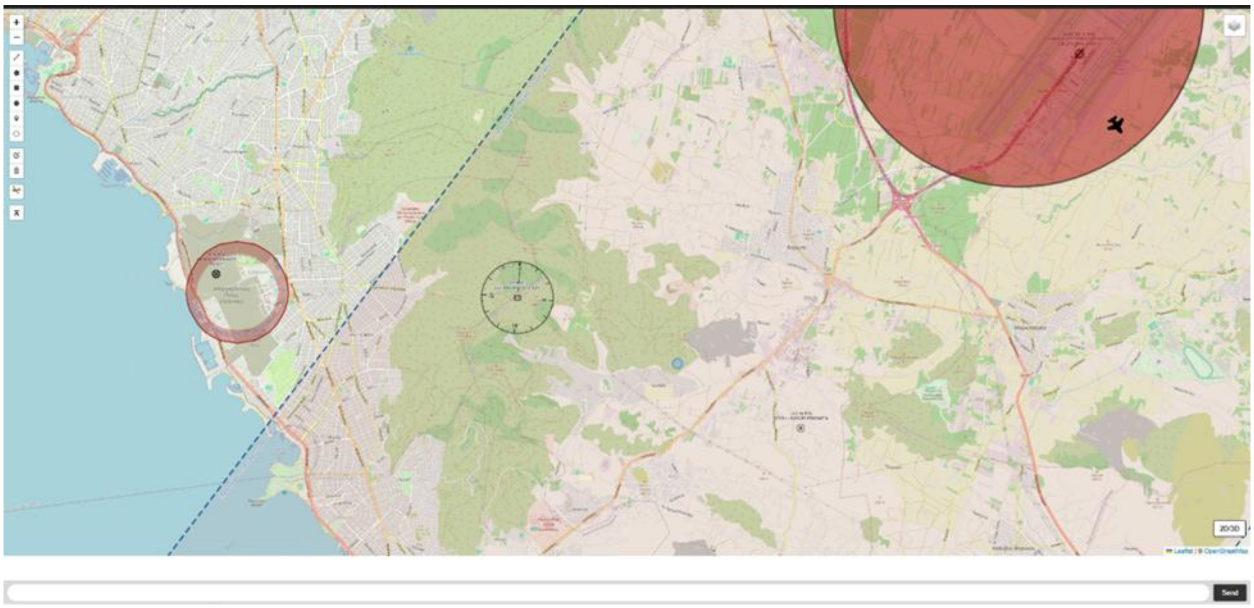


Figure 20: Circle marker tool

Menu Edit/Delete

Map shape edit and delete functions

Firefighting Command

Firefighting command control button.

File upload

Kml, gpx, geoJSON file support for track overview.

Layers

Layer control and decluttering information controller.

2D/3D

Controller switching between 2D and 3D maps.

Each element of the map is interactive, with information displayed on a left click.

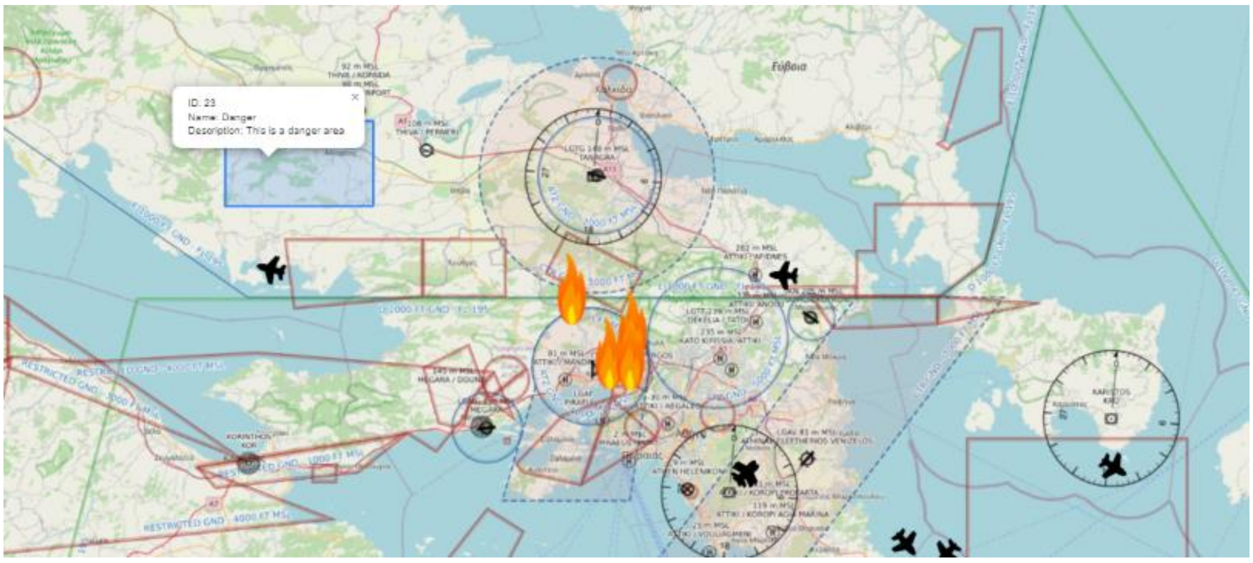


Figure 21: Shape information

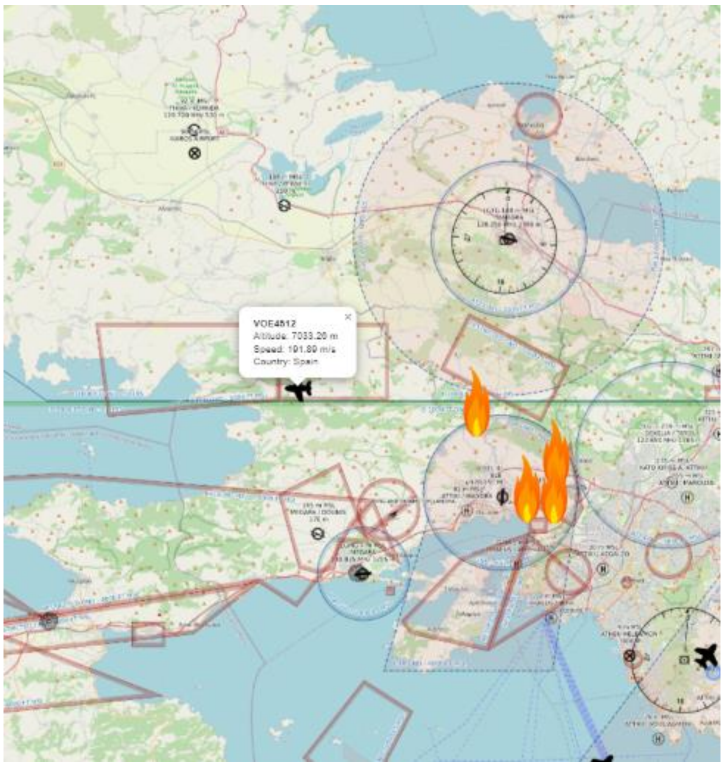

VOE4512
Altitude: 7033.20 m
Speed: 191.89 m/s
Country: Spain

Figure 22: Aircraft information

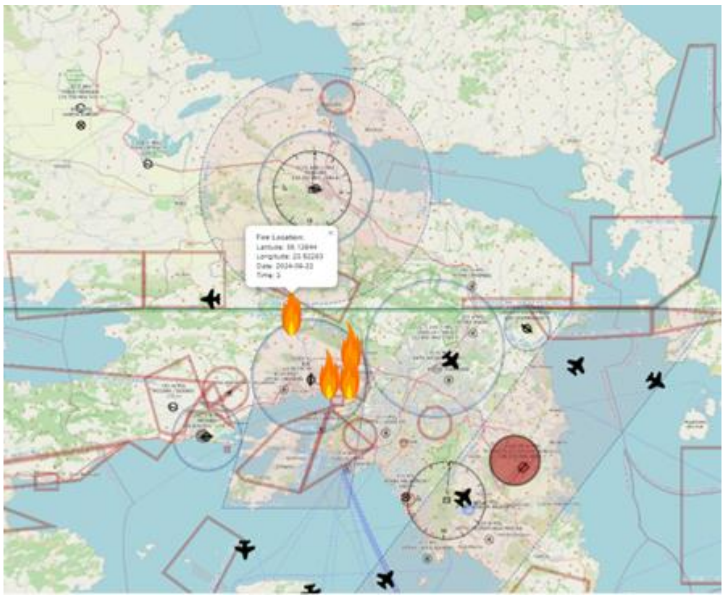

Fire Location:
Latitude: 38.12844
Longitude: 23.52243
Date: 2024-08-22
Time: 3

Figure 23: Fire information

Map Options

The application has the capability to display:

Satellite map.

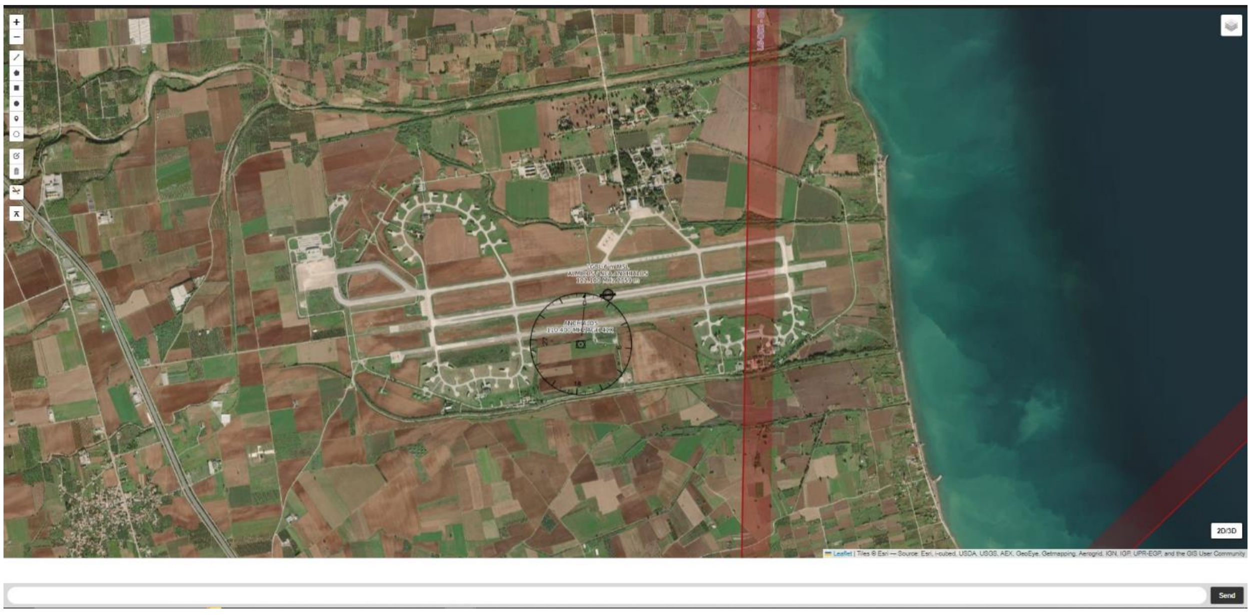


Figure 24: Satellite map

Tactical high contrast map

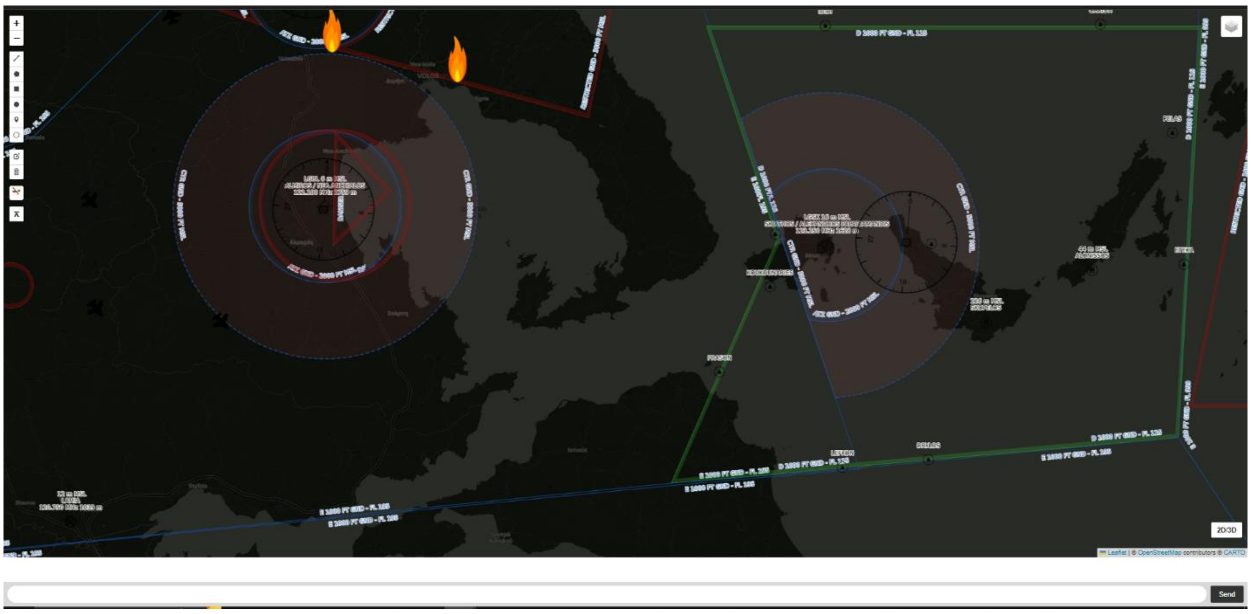


Figure 25: Tactical map

Topographic map

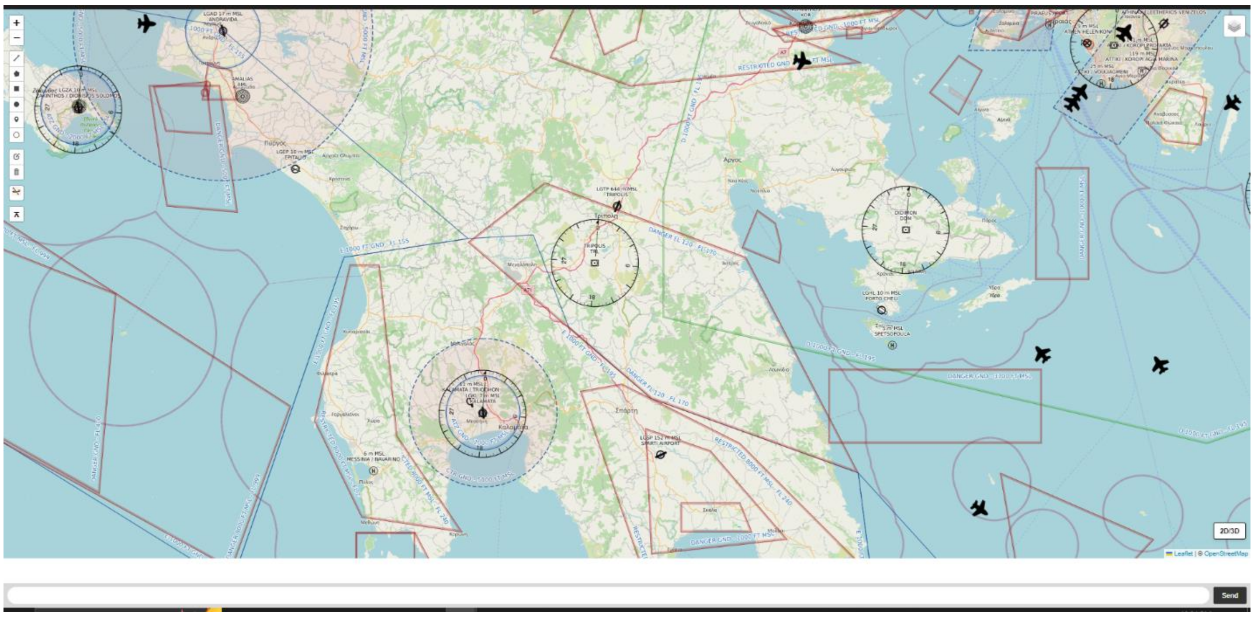


Figure 26: Topographic map

3D Map

The option to use a three-dimensional map is provided.

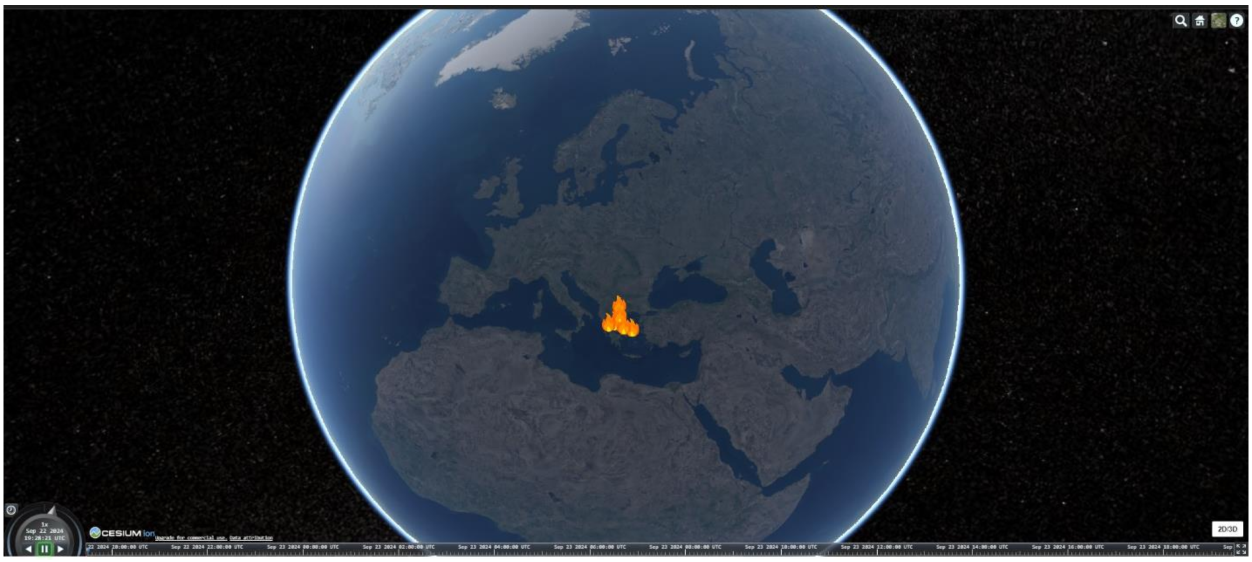


Figure 27: 3D map top view

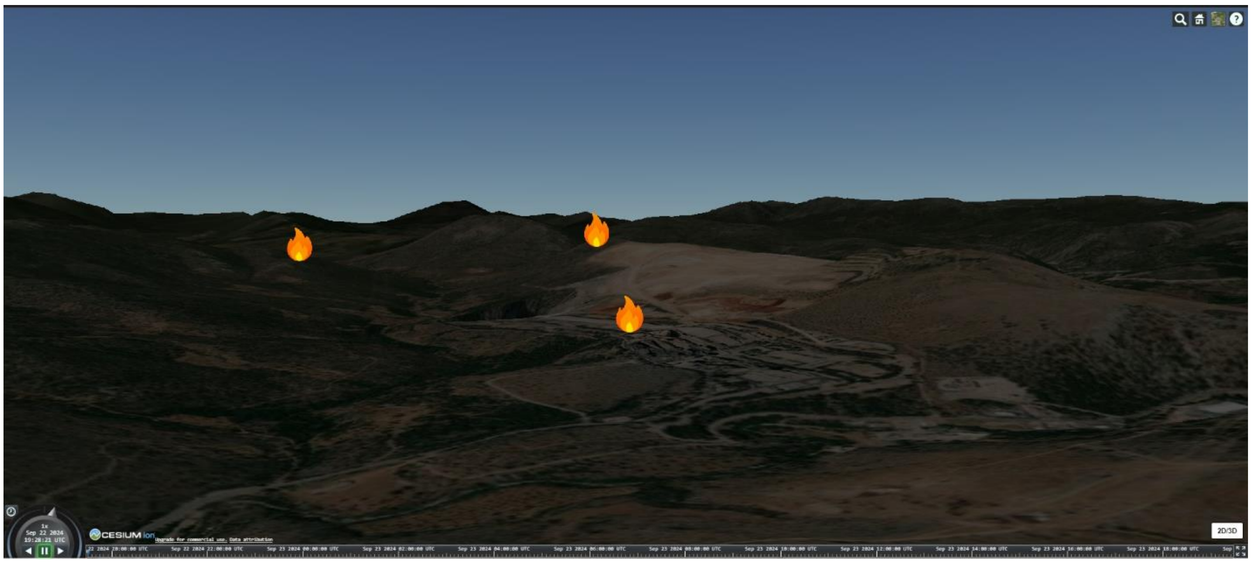


Figure 28: 3D map side-terrain view

4.2. Virtual Use Case Scenario

For the presentation of the application's use case scenario, data from the major fires that struck the Attica region on August 11, 2024, will be used. Below is an outline of the actions and functions of the application for emergency situations like these.

This specific capture was taken on August 12, 2024, at 07:38:02 UTC, when the FIRMS system had already recorded the event, and the vast extent of the fire was threatening inhabited areas.

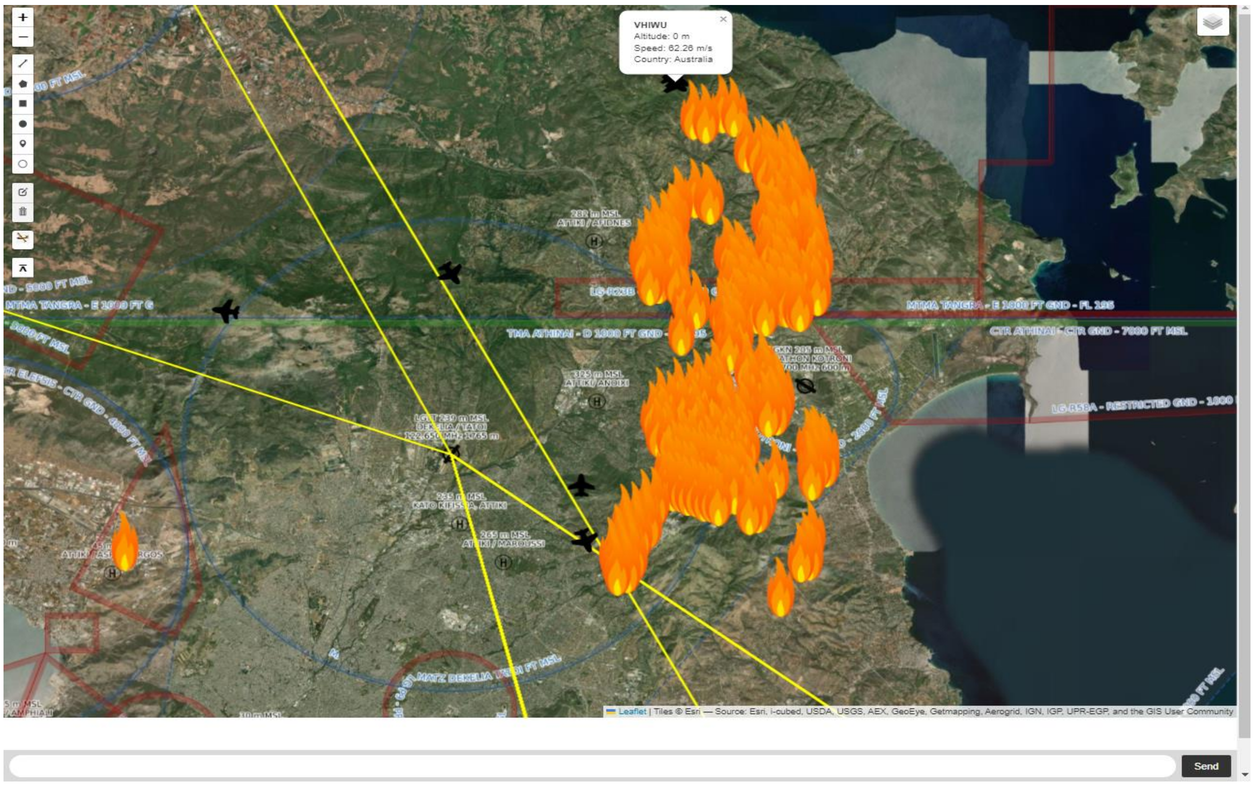
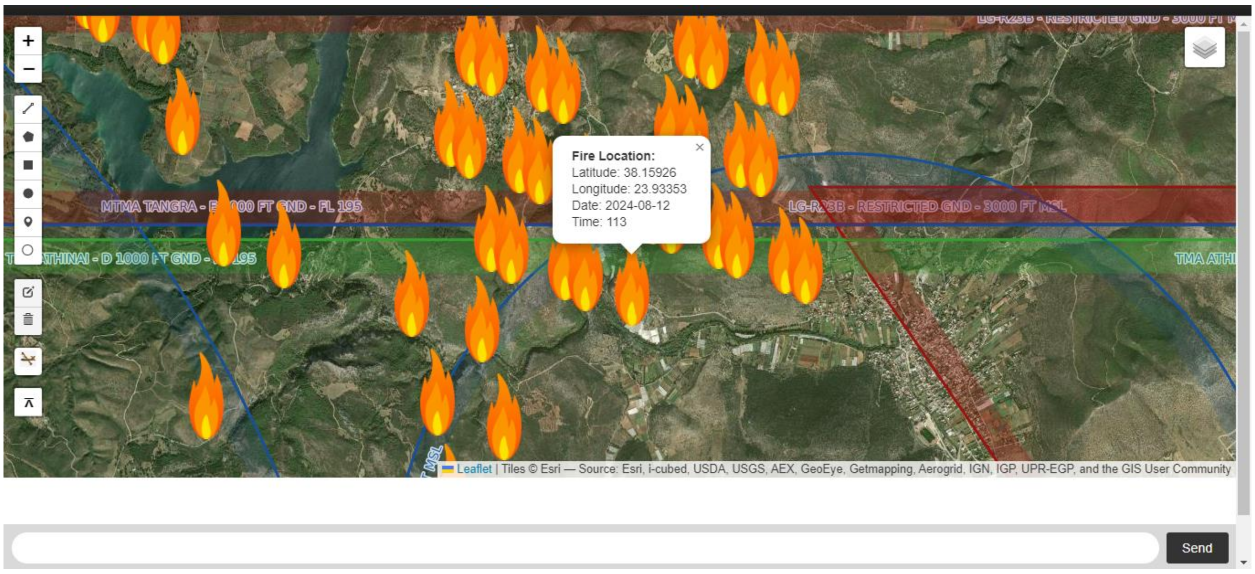

VHIWU
Altitude: 0 m
Speed: 62.26 m/s
Country: Australia
TMA ATHINAI - D 1000 FT GND
CTR ATHINAI - CTR GND - 7000 FT MSL
Send
Leaflet | Tiles © Esri — Source: Esri, i-cubed, USDA, USGS, AEX, GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and the GIS User Community

Figure 29: Fires on Attica region

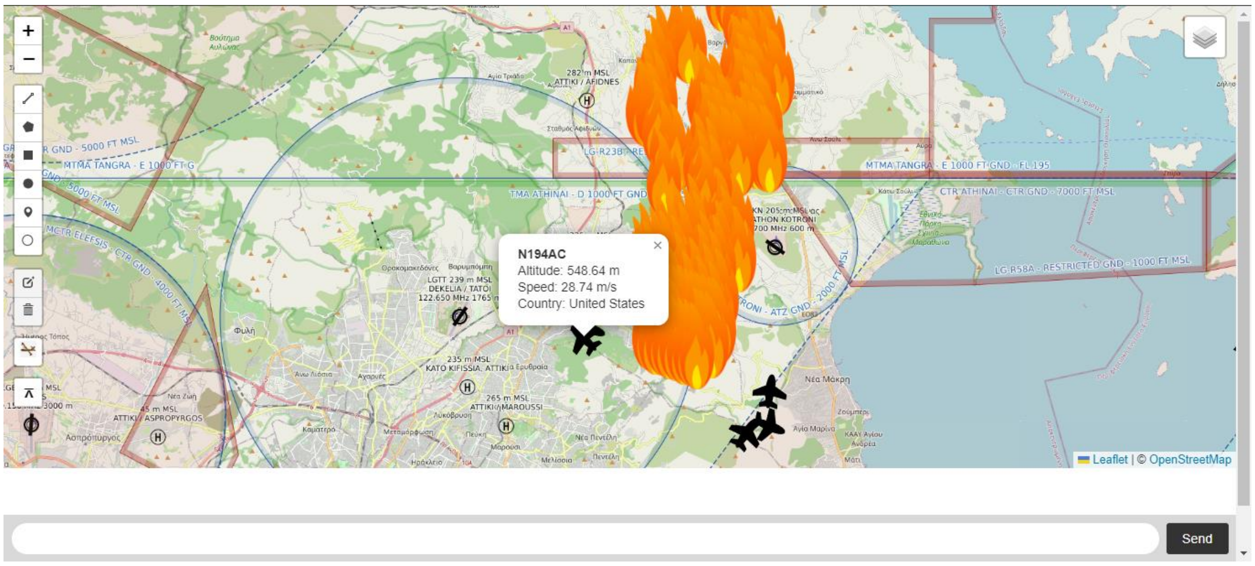
From another perspective on the satellite map, we can see how the fire hotspots are distributed, as well as the type of terrain (mountainous areas, forests, inhabited regions).


Fire Location:
Latitude: 38.15926
Longitude: 23.93353
Date: 2024-08-12
Time: 113

MTMA TANGRA - E 1000 FT GND - FL 195
ATHINAI - D 1000 FT GND - FL 195
LG-R23B - RESTRICTED GND - 3000 FT MSL
TMA ATHI
Leaflet | Tiles © Esri — Source: Esri, i-cubed, USDA, USGS, AEX, GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and the GIS User Community
Send

Figure 30: Satellite map fire projection

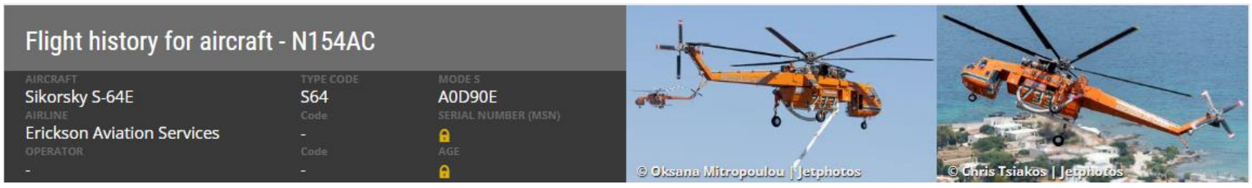
From the application, we observe increased aerial traffic of aircraft and helicopters approaching the area of interest from the southern side, with their movement indicating that they are operating from Tatoi Airport (LGTT). Additionally, the information provided by the application allows us to identify the types of aircraft that are engaged in operations.


N194AC
Altitude: 548.64 m
Speed: 28.74 m/s
Country: United States

MTMA TANGRA - E 1000 FT G
TMA ATHINAI - D 1000 FT GND
MTMA TANGRA - E 1000 FT GND - FL 195
CTR ATHINAI - CTR GND - 7000 FT MSL
LG-R58A - RESTRICTED GND - 1000 FT MSL
LGTT 239 m MSL DEKELIA / TATOI 122.650 MHz
Leaflet | © OpenStreetMap
Send

Figure 31: Air traffic information

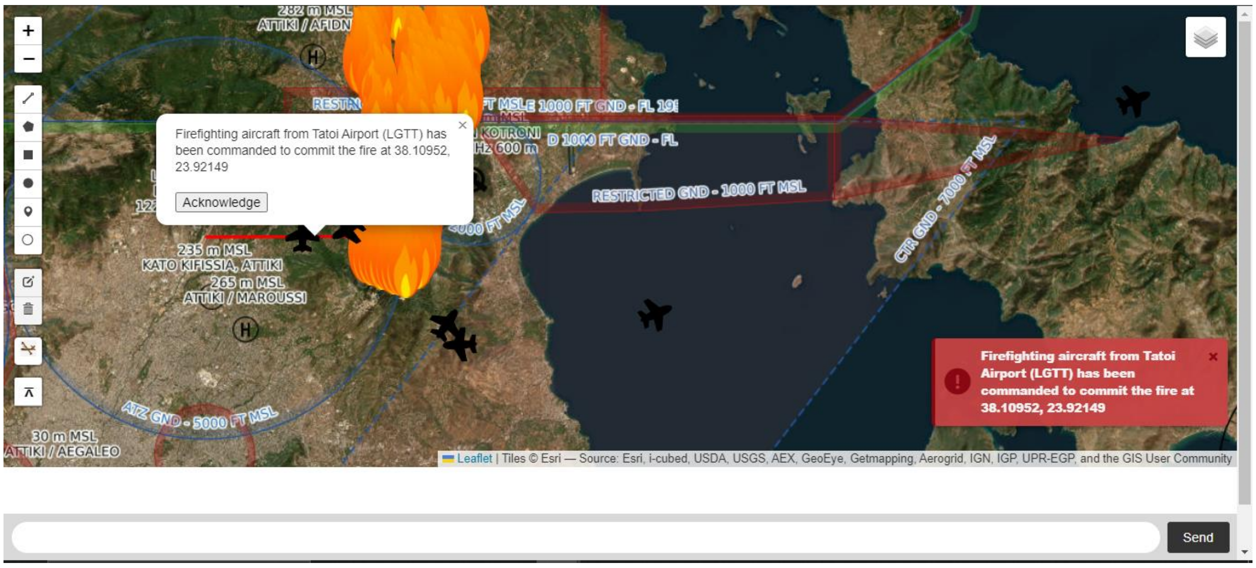
At this specific moment, a Sikorsky S-64E helicopter appears to be vectoring toward its home base, likely for refueling.


Flight history for aircraft - N154AC

AIRCRAFT	TYPE CODE	MODE S
Sikorsky S-64E	S64	A0D90E
AIRLINE	Code	SERIAL NUMBER (MSN)
Erickson Aviation Services	-	🔒
OPERATOR	Code	AGE
-	-	🔒

Figure 32: Sikorsky S-64E

A possible use case scenario that was considered is the replacement of the aircraft returning for refueling with another one from the nearest base to the area of interest. For this purpose, a firefighting command is dispatched. Specifically, the administrator at the control center sends a “commit” order for an aircraft to operate from LGTT airport to the area of interest at coordinates 38.10952, 23.92149.


Firefighting aircraft from Tatoi Airport (LGTT) has been commanded to commit the fire at 38.10952, 23.92149

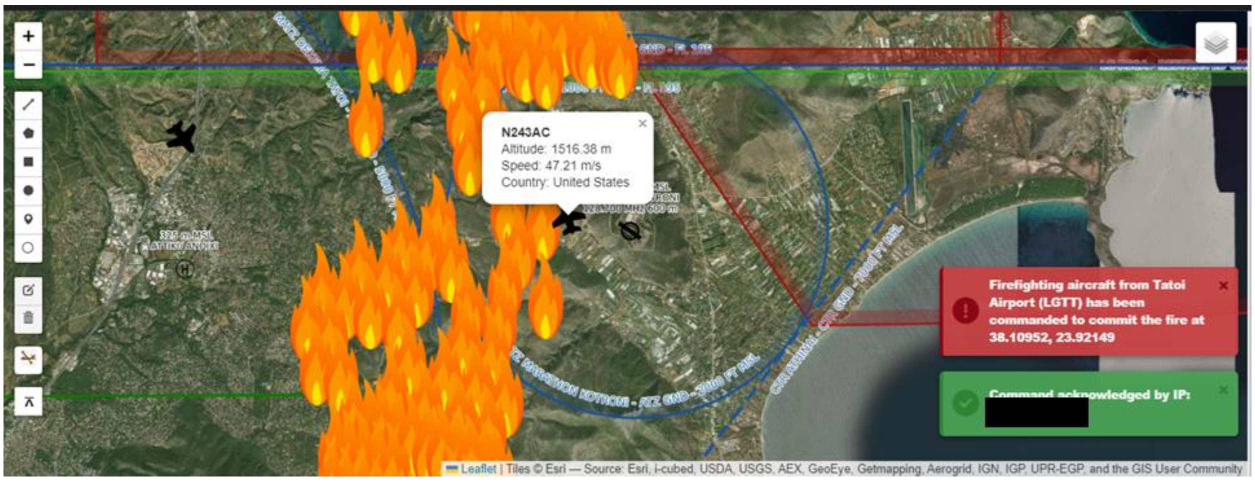
Acknowledge

Firefighting aircraft from Tatoi Airport (LGTT) has been commanded to commit the fire at 38.10952, 23.92149

Leaflet | Tiles © Esri — Source: Esri, i-cubed, USDA, USGS, AEX, GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and the GIS User Community

Figure 33: Firefighting command

This specific command is displayed in real-time to all users. Subsequently, the operator operating from Tatoi Airport, corresponding to the internal network position with the corresponding IP, accepts the command. What is expected from the administrator next is to see an aircraft departing from the base toward the area of interest.


N243AC
Altitude: 1516.38 m
Speed: 47.21 m/s
Country: United States

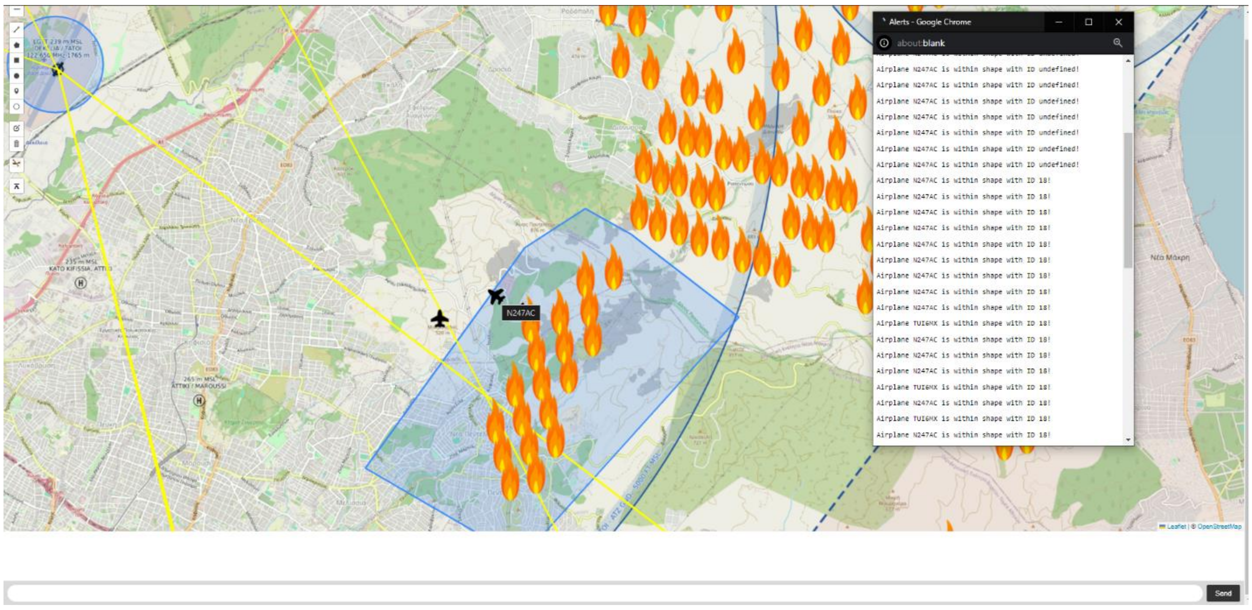
Firefighting aircraft from Tatoi Airport (LGTT) has been commanded to commit the fire at 38.10952, 23.92149

Command acknowledged by IP: [redacted]

Leaflet | Tiles © Esri — Source: Esri, i-cubed, USDA, USGS, AEX, GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and the GIS User Community

Figure 34: Command acceptance

The latest information can also be retrieved through the auxiliary window that displays alerts.



```
Airplane N247AC is within shape with ID undefined!
Airplane N247AC is within shape with ID undefined!
Airplane N247AC is within shape with ID undefined!
Airplane N247AC is within shape with ID undefined!
Airplane N247AC is within shape with ID undefined!
Airplane N247AC is within shape with ID undefined!
Airplane N247AC is within shape with ID undefined!
Airplane N247AC is within shape with ID 18!
Airplane N247AC is within shape with ID 18!
Airplane N247AC is within shape with ID 18!
Airplane N247AC is within shape with ID 18!
Airplane N247AC is within shape with ID 18!
Airplane N247AC is within shape with ID 18!
Airplane N247AC is within shape with ID 18!
Airplane N247AC is within shape with ID 18!
Airplane N247AC is within shape with ID 18!
Airplane TUI6MX is within shape with ID 18!
Airplane N247AC is within shape with ID 18!
Airplane N247AC is within shape with ID 18!
Airplane N247AC is within shape with ID 18!
Airplane TUI6MX is within shape with ID 18!
Airplane N247AC is within shape with ID 18!
Airplane TUI6MX is within shape with ID 18!
Airplane N247AC is within shape with ID 18!
```

Figure 35: Alert window message informations

4.3. Performance Measurements

For the performance measurements, available tools will be utilized. Within the scope of this work, the tests will primarily focus on usability rather than large-scale testing due to resource constraints.

Initially, tests were conducted to examine how the application responds to various screen sizes and devices beyond the computer, and the results are presented below.

Mobile

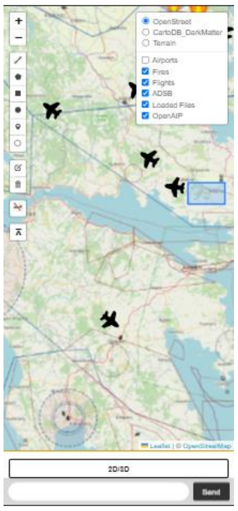


Figure 36: Mobile size screen

Tablet

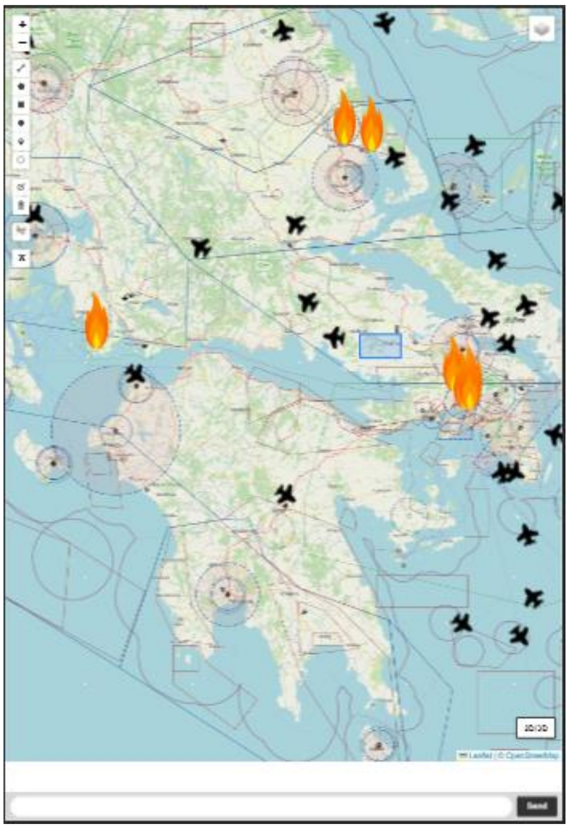


Figure 37: Tablet size screen

Tablet Landscape

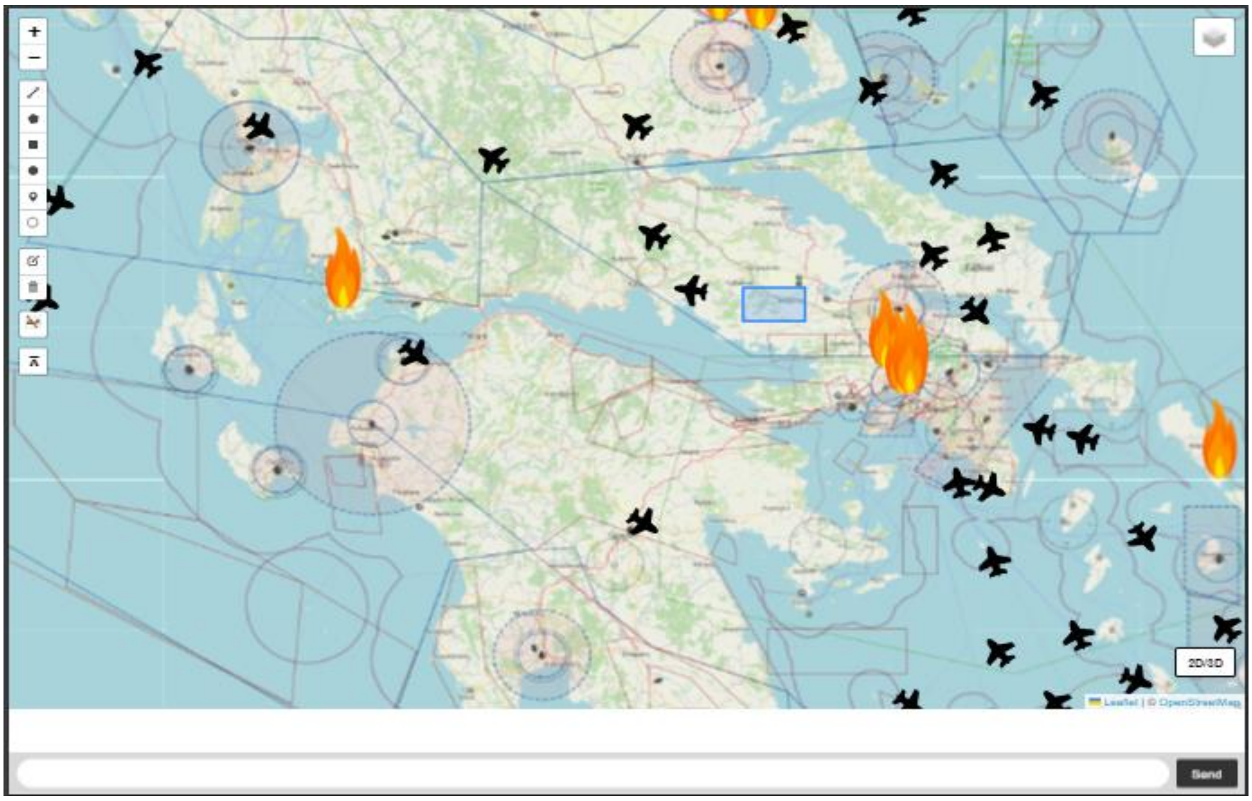
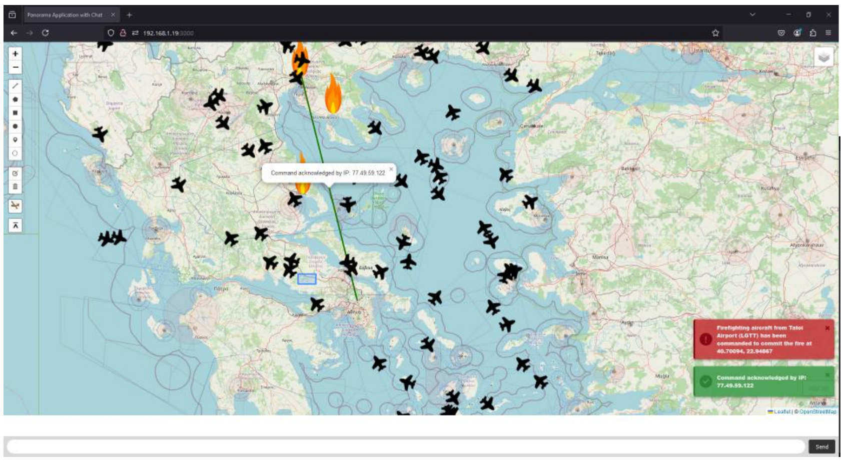


Figure 38: Tablet landscape view

Additional tests were conducted on different browsers.

Mozilla



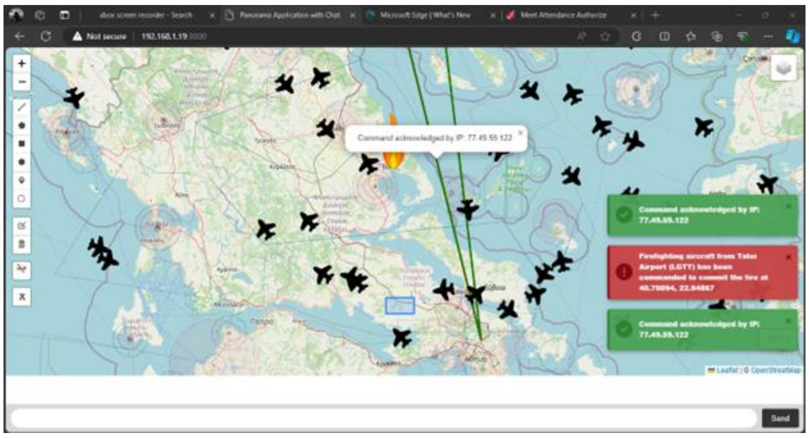
Command acknowledged by IP: 77.49.59.122

Firefighting aircraft from Tatoi Airport (LGTT) has been commanded to commit the fire at 40.70094, 22.94867

Command acknowledged by IP: 77.49.59.122

Figure 39: Mozilla Firefox browser

Edge


Command acknowledged by IP: 77.49.59.122

Command acknowledged by IP: 77.49.59.122

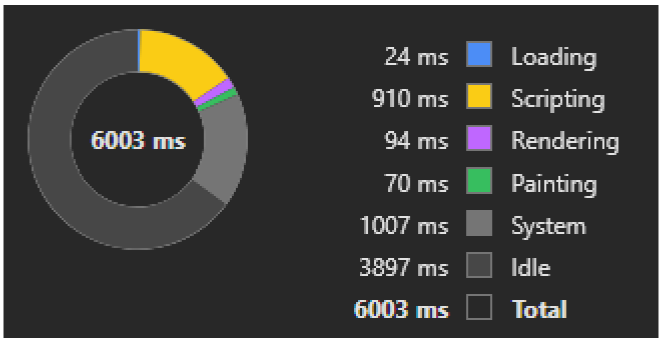
Firefighting aircraft from Tatoi Airport (LGTT) has been commanded to commit the fire at 40.70094, 22.94867

Command acknowledged by IP: 77.49.59.122

Figure 40: Microsoft Edge browser

In more measurable tests, we have the following results:

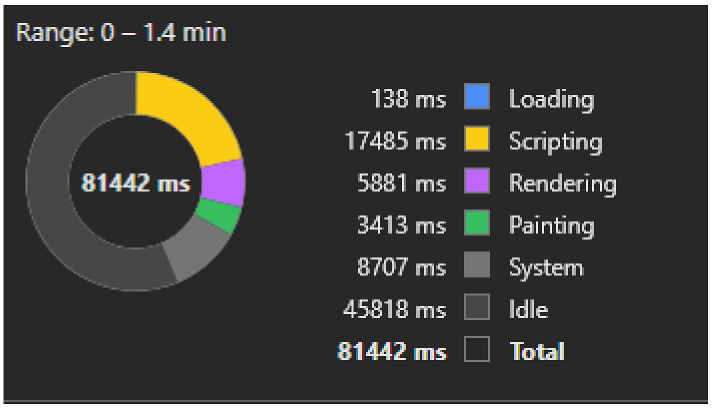
Upon the initial loading of the page, we observe the time required and the time demands for each case.



24 ms	🟦	Loading
910 ms	🟨	Scripting
94 ms	🟪	Rendering
70 ms	🟩	Painting
1007 ms	■	System
3897 ms	■	Idle
6003 ms	□	**Total**

Figure 41: Loading map page time

In the next measurement, we tested intensive use of the application, switching from 2D to 3D as well as utilizing drawing functions, which took approximately 81 seconds. Below are the results we recorded.



Range: 0 – 1.4 min

81442 ms

138 ms	🟦	Loading
17485 ms	🟨	Scripting
5881 ms	🟪	Rendering
3413 ms	🟩	Painting
8707 ms	▨	System
45818 ms	▨	Idle
81442 ms	☐	**Total**

Figure 42: Usage analysis

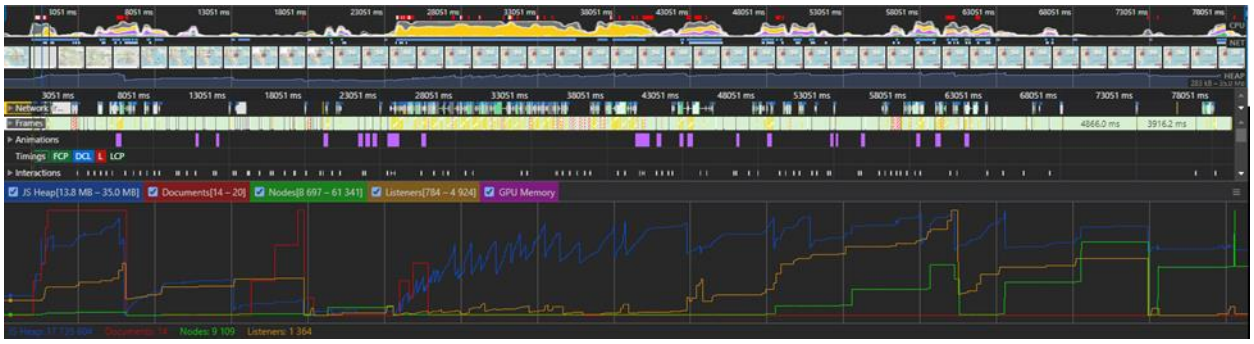


Figure 43: Functions analysis

Simultaneously, the operational requirements are illustrated in a chart for the 81-second time period.

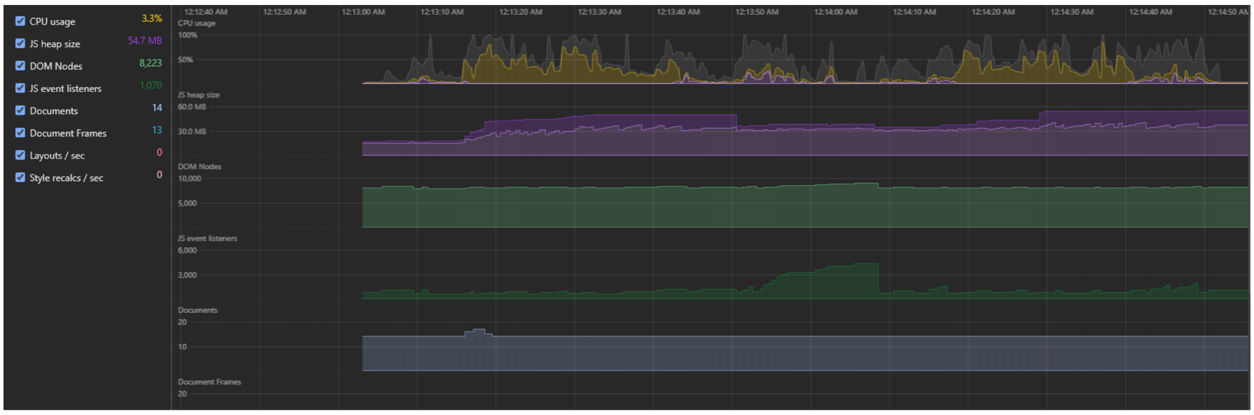
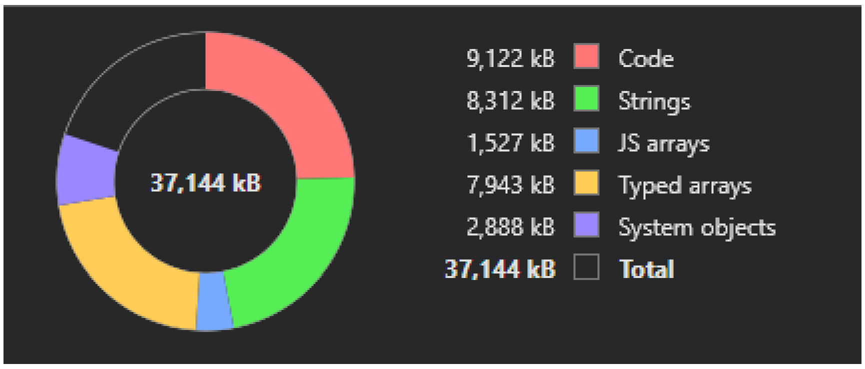


Figure 44: System requirements analysis

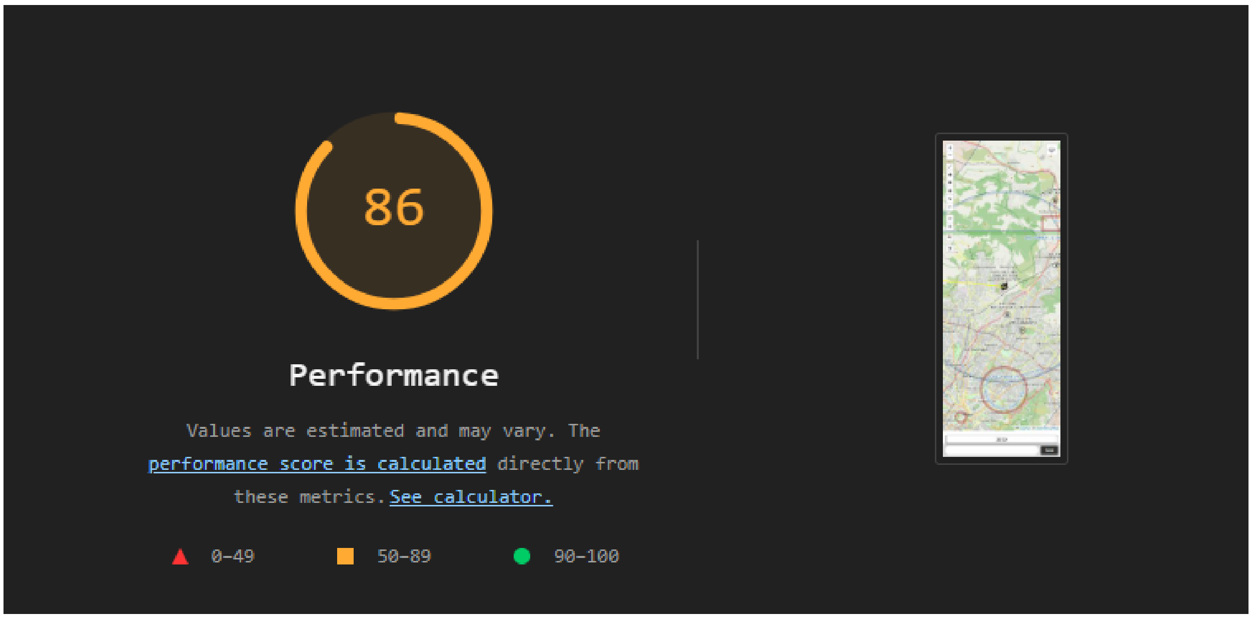
Meanwhile, the following image analyzes the Heap memory requirements.



9,122 kB	🟥	Code
8,312 kB	🟩	Strings
1,527 kB	🟦	JS arrays
7,943 kB	🟨	Typed arrays
2,888 kB	🟪	System objects
37,144 kB	☐	**Total**

Figure 45: Heap memory analysis

Finally, an overall performance rating of the application across different browsers is presented.



Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

Figure 46: Performance rating

4.4. Unimplemented Objectives

Although the application's needs were thoroughly analyzed and the usage method defined, there are several objectives that could not be achieved.

Security: A significant aspect that was not implemented is the use of HTTPS protocol. While encryption functions are used for credentials, HTTPS was not utilized, which would have ensured the integrity of messages and information. As a result, the application's evaluation reached only 86. The primary reason this objective was not executed is that it would increase code complexity, risking compatibility issues or hindering smooth usage within the scope of this work. However, it remains a future goal that could be implemented.

Data Caching: Another important aspect that was not implemented is the use of caching techniques for data. A highly efficient solution would have been the use of NoSQL databases for data storage and subsequent visualization. Although this functionality would have provided many advantages, it was not carried out because it requires a separate design module and is beyond the scope of this work.

Chapter 5: Conclusion and Future Work

The idea behind creating the application is to manage civil protection resources using modern methods. The goal is to develop an emergency management information system within a secure and controlled environment. The main function of the application is the coordination, communication, and collaboration among involved entities through crowdsourcing.

To achieve the objectives and necessary functions, the application was based on the structure of web map services. For this purpose, web development technologies were used, both on the frontend and backend levels. Specifically, users have access to the interface through an interactive map that operates in a browser environment. All data management and business logic services are handled on the backend and made available to users through a server. The data is distributed simultaneously, and every map created by a user is immediately available to others.

It should be noted that the application is a structure that can certainly be improved and further developed. Future development possibilities are listed below:

- Data Overlay: Currently, the application provides data on fires, aviation, and either satellite or topographic maps.
    - Weather: Further development could include adding more real-time weather data. Such data would enhance the application's features and expand its functionality.
    - GPS Jamming: This type of data could further assist in analyzing situations where the location of resources is of utmost importance.
    - Earthquakes: These data could be provided in real-time either as information or as alerts, further aiding in operations related to such incidents.

- Caching: A key objective during the development of the application is its optimal performance. Managing large amounts of data in real-time requires a high commitment of computational resources. Although caching techniques are provided by browsers in which the application runs, developing a caching method on the backend would provide significant advantages and improve the application’s performance, making it capable of handling even larger volumes of data.

- Security: The application is intended for use in control and central command structures. The security section of the application could be further developed with the use of more security techniques and on a larger scale.

- AI: A potential future development of the application could be the use of artificial intelligence. In this way, various data could be properly managed to automatically generate better results or suggest more efficient actions.

References

1. Lu, W., Ai, T., Zhang, X., & He, Y. (2017). An interactive web mapping visualization of urban air quality monitoring data of China. Atmosphere, 8(8), 148.

2. Cybulski, P., & Horbiński, T. (2020). User experience in using graphical user interfaces of web maps. ISPRS International Journal of Geo-Information, 9(7), 412.

3. Alesheikh, A. A., Helali, H., & Behroz, H. A. (2002, July). Web GIS: technologies and its applications. In Symposium on geospatial theory, processing and applications (Vol. 15, pp. 213-222). Hannover, Germany: ISPRS.

4. Levis, A. H., & Wagenhals, L. W. (2000). C4ISR architectures: I. Developing a process for C4ISR architecture design. Systems engineering, 3(4), 225-247.

5. Diallo, O., Rodrigues, J. J., & Sene, M. (2012). Real-time data management on wireless sensor networks: A survey. Journal of Network and Computer Applications, 35(3), 1013-1021.

6. Veenendaal, B., Brovelli, M. A., & Li, S. (2017). Review of web mapping: Eras, trends and directions. ISPRS International Journal of Geo-Information, 6(10), 317.

7. Tilkov, S., & Vinoski, S. (2010). Node. js: Using JavaScript to build high-performance network programs. IEEE Internet Computing, 14(6), 80-83.

8. Lei, K., Ma, Y., & Tan, Z. (2014, December). Performance comparison and evaluation of web development technologies in php, python, and node. js. In 2014 IEEE 17th international conference on computational science and engineering (pp. 661-668). IEEE.

9. Horbiński, T., & Lorek, D. (2022). The use of Leaflet and GeoJSON files for creating the interactive web map of the preindustrial state of the natural environment. Journal of Spatial Science, 67(1), 61-77.

10. Schäfer, M., Strohmeier, M., Lenders, V., Martinovic, I., & Wilhelm, M. (2014, April). Bringing up OpenSky: A large-scale ADS-B sensor network for research. In IPSN-14 Proceedings of the 13th International Symposium on Information Processing in Sensor Networks (pp. 83-94). IEEE.

11. Çolak, E., & Sunar, F. (2020). The importance of ground-truth and crowdsourcing data for the statistical and spatial analyses of the NASA FIRMS active fires in the Mediterranean Turkish forests. Remote Sensing Applications: Society and Environment, 19, 100327.