SPLASH

Iasonas Lamprinidis, Vasileios Zoidis, Ioannis Michalainas, Savvas Tzanetis

May-June 2025

Code Repository:
SPLASH

Abstract

SPLASH - Sea Parameter Logging And Sensor Hub.
A brief overview of the project: goals, key features and overall system architecture.

Contents

1 Introduction

This project implements a *remote weather station*. The premise is to use multiple Arduino devices to create a network of data nodes. These nodes will monitor temperature, pressure and estimated wave shore impact/height at sea level. The collected data will then be displayed on a web application for fishermen, researchers, or hobbyists to use.

1.1 Motivation and Objectives

The primary motivation for this project is to enable **cost-effective**, **modular remote monitoring** of maritime weather conditions using low-power Arduino nodes. Real-time insights into *temperature*, *atmospheric pressure* and *wave activity* (approximated via accelerometer data) are crucial for coastal monitoring, fishing safety and climate research.

Objectives:

- Design a distributed sensor node that gathers environmental data.
- Implement wireless communication between sensor and receiver using RF modules.
- Display gathered information on a centralized platform.
- Optimize energy use through staggered data transmission and low-power components.

1.2 Report Structure

This report's structure mimics the workflow of our app. We will be explaining from the bottom up the architecture, design choices and development stages of our project.

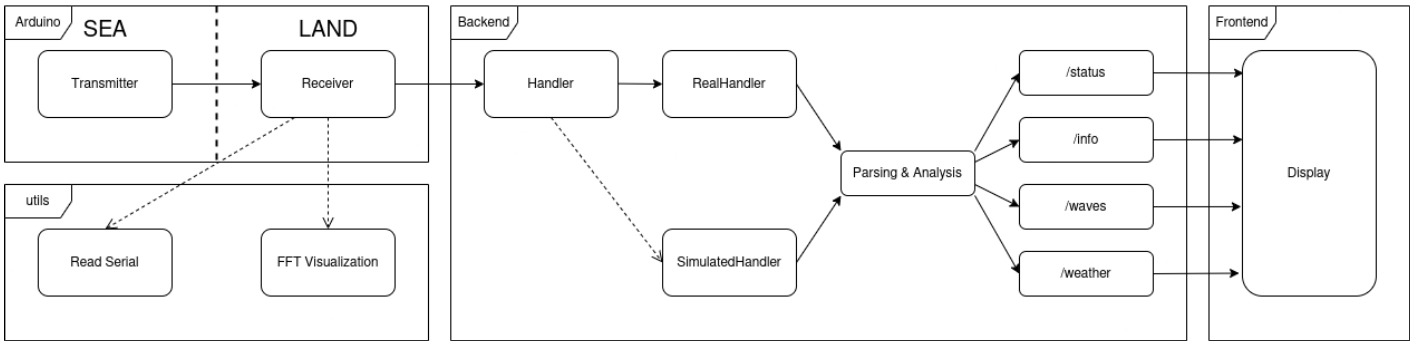


Figure 1: Workflow

2 Arduino

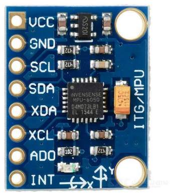
In this section, we discuss the Arduino system's features, capabilities and architecture. The role of the Arduino module is to acquire raw data from multiple sensors, process and format it into structured packets and wirelessly transmit it to the receiver for further handling. This hardware platform serves as the primary interface with the physical environment, enabling real-time environmental monitoring and data collection.

2.1 Hardware Components

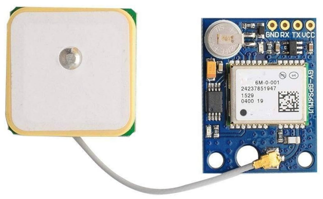
2.1.1 Sensor Selection

We aimed to develop a compact and cost-effective environmental sensing system capable of monitoring atmospheric conditions, motion dynamics and geographic positioning. This is the reason why the following three sensors were integrated into the transmitter node:

BMP280: Measures atmospheric temperature and pressure with high precision and stability.



MPU6050: A 6-axis accelerometer and gyroscope used to estimate wave impact or surface motion dynamics (via the z-axis).



NEO-6M: A GPS module, known for its compact size, affordability and reliable performance.

These sensors were selected for their balance between accuracy, affordability and compatibility with Arduino-based systems.

2.1.2 Wiring and Pinout

- **BMP280** is connected via I^{2}C using default SDA and SCL pins.
- **MPU6050** shares the same I^{2}C bus.
- **NEO-6M** is connected via UART using digital pin 2 and 3 for the Rx and Tx connections respectively.
- **RF22 module** uses SPI, with connections depending on the specific Arduino board used (for the Uno: MOSI = 11, MISO = 12, SCK = 13, SS = 10).

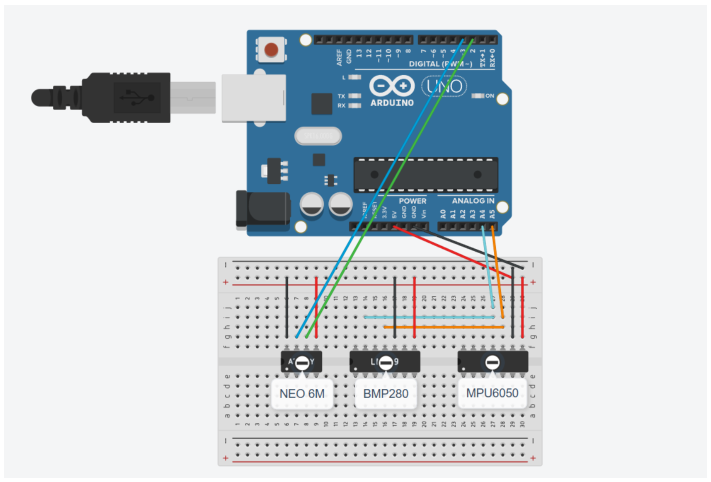


Figure 2: Circuit Design

2.1.3 Enclosure and Mounting

The enclosure is waterproof and weather resistant, as well as plastic, so that the various transmissions can pass uninterrupted. The MPU6050 sensor is oriented such that the z-axis faces upwards, enabling detection of vertical movement from wave activity.

2.2 Transmitter Module

2.2.1 Data Acquisition Loop

Sensor readings are acquired in each loop iteration:

- BMP280 returns temperature and pressure.
- MPU6050 returns raw acceleration and gyroscopic values.
- NEO-6M returns latitude and longitude.

Every second, a full data packet is composed, including coordinates, temperature, pressure, signal strength and vertical acceleration. In intermediate cycles, only acceleration data are transmitted to reduce energy consumption.

2.2.2 Transmission

Communication is carried out using RF22 modules configured with GFSK modulation at 440 MHz. Packets are transmitted using the sendtoWait() function, which includes acknowledgment. The configuration is set to use the maximum transmission power of 20 dBm, with modulation settings optimized to balance range and speed.

2.2.3 Power Management

Although advanced sleep modes are not implemented, energy efficiency is improved through selective data transmission (with full packets sent once per second), configurable transmission power and the use of power-efficient sensors.

2.3 Receiver Module

2.3.1 Reception

The receiver node mirrors the transmitter's configuration for compatibility. It uses the same frequency and modulation settings, continuously listens for incoming data packets using the `recvfromAck()` function and calculates the signal strength (RSSI) for logging and diagnostics.

2.3.2 Serial Output

Incoming data packets are printed directly onto the serial monitor in string format. This output is redirected to the backend and later to the GUI for end-user visualization and analysis.

3 Backend

In this section we discuss the backend's features and capabilities, as well as its architecture. The role of the backend is to capture raw data from the sensor, analyze it, package it and send it to the frontend to be displayed in a meaningful way.

3.1 Data Ingestion

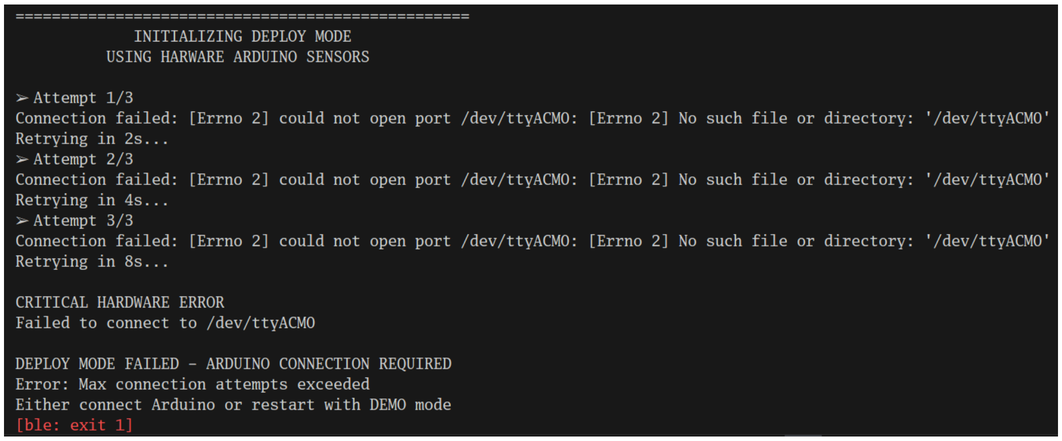
The data ingestion system is designed with a flexible architecture that accommodates both development and production environments through two specialized handlers; `SimulatedHandler` and `RealHandler`. Both handlers expose a consistent interface through the `SerialHandler` facade, which transparently selects the appropriate implementation based on the configured operational mode (`DEMO` or `DEPLOY`). This design pattern enables the application to operate identically regardless of data source, facilitating a smooth transition between development and production environments.

3.1.1 Simulated Handler

The `SimulatedHandler` provides a simulation environment for development and testing purposes. It generates realistic sensor data that mimics actual hardware outputs. The simulation incorporates deliberate variability -employing sinusoidal patterns with randomized amplitude, frequency and noise components- to realistically model sensor behavior. Data generation occurs in a dedicated thread running at 100Hz, maintaining a rolling buffer of 500 samples (equivalent to 5 seconds of historical acceleration data). This approach enabled us to test data processing pipelines, visualization systems and analytical algorithms without requiring physical sensor hardware.

3.1.2 Real Handler

The `RealHandler` establishes and maintains communication with physical Arduino-based sensor hardware via serial connection. It implements several resilience features including connection retry logic with exponential backoff, error handling for disconnection scenarios and automatic reconnection attempts. The handler continuously reads and parses incoming serial data through a dedicated thread, transforming raw string input into structured data objects according to the expected protocol format. Invalid or corrupted data is filtered through parsing logic that validates ranges and formats. Similar to its simulated counterpart, this handler maintains a time-series buffer of recent acceleration readings, enabling applications to analyze trends and patterns in the most recent sensor data.



```
==================================================
             INITIALIZING DEPLOY MODE
          USING HARWARE ARDUINO SENSORS

➢ Attempt 1/3
Connection failed: [Errno 2] could not open port /dev/ttyACM0: [Errno 2] No such file or directory: '/dev/ttyACM0'
Retrying in 2s...
➢ Attempt 2/3
Connection failed: [Errno 2] could not open port /dev/ttyACM0: [Errno 2] No such file or directory: '/dev/ttyACM0'
Retrying in 4s...
➢ Attempt 3/3
Connection failed: [Errno 2] could not open port /dev/ttyACM0: [Errno 2] No such file or directory: '/dev/ttyACM0'
Retrying in 8s...

CRITICAL HARDWARE ERROR
Failed to connect to /dev/ttyACM0

DEPLOY MODE FAILED - ARDUINO CONNECTION REQUIRED
Error: Max connection attempts exceeded
Either connect Arduino or restart with DEMO mode
[ble: exit 1]
```

Figure 3: RealHandler Resilience

3.2 Data Processing

3.2.1 Wave Analysis

The wave analysis module implements a frequency-domain approach to extract meaningful metrics from raw accelerometer data. It performs Fast Fourier Transform (FFT) to identify the dominant wave frequency, filtering out low-frequency components (below 0.1 Hz) to eliminate sensor drift and environmental noise. The system converts raw accelerometer readings from LSB units to standard m/s^2 using a calibration factor derived from the MPU6050 sensor's sensitivity at $\pm$2g range (datasheet). A physics-based heuristic transforms the acceleration variance into estimated wave height by applying appropriate scaling based on wave frequency, yielding actionable metrics that describe sea state conditions.

3.2.2 Mathematical Background

The shore impact analysis implements established coastal engineering models to translate wave metrics into practical inundation forecasts.[1] The system calculates fundamental wave parameters including period (reciprocal of frequency) and deep-water wavelength using standard gravity-based equations. The core of the analysis employs the Iribarren number (surf similarity parameter) to classify wave behavior as either dissipative or reflective/intermediate, applying the appropriate runup estimation formula for each regime. The model classifies hazard zones based on runup thresholds, with values exceeding 0.9 meters designated as high-risk (VE zone).[2] The final inundation distance is calculated by applying the slope correction factor, providing horizontal reach estimates critical for coastal planning.

3.3 API Endpoints

In this section we analyze the API endpoints available for the frontend to call

3.3.1 Status

The /status endpoint provides essential system telemetry for monitoring the operational state of the device. It returns a JSON payload containing connectivity status, power information (battery percentage) and geolocation coordinates derived from the sensor data stream. The endpoint also includes signal strength metrics (measured in dBm) to facilitate deployment troubleshooting and network quality assessment. By serving as a lightweight health check, this endpoint enables rapid verification of device functionality without interrogating the more resource-intensive data processing pipelines.

3.3.2 Info

The /info endpoint exposes environmental context data captured by the sensor array, specifically temperature (in degrees Celsius) and barometric pressure (in hectopascals).

3.3.3 Waves

The /waves endpoint delivers processed wave characteristics and their projected coastal impact. It retrieves the acceleration time-series buffer from the sensor handler and passes it through the wave analysis pipeline to extract frequency and height parameters. These metrics are then fed into the shore impact model to calculate vertical runup height, FEMA flood zone classification and horizontal inundation distance based on a predefined slope parameter (set to 0.05 const). This endpoint represents the system's primary analytical output, transforming raw sensor data into actionable coastal flooding information delivered in a standardized JSON format.

3.3.4 Weather

The /weather endpoint enriches the system's data with third-party meteorological information by interfacing with the OpenWeatherMap API. It dynamically constructs API requests using the device's current geolocation coordinates and securely transmits the authentication key from environment variables. The endpoint parses the response to extract wind parameters (speed and direction) along with general weather conditions and human-readable descriptions.

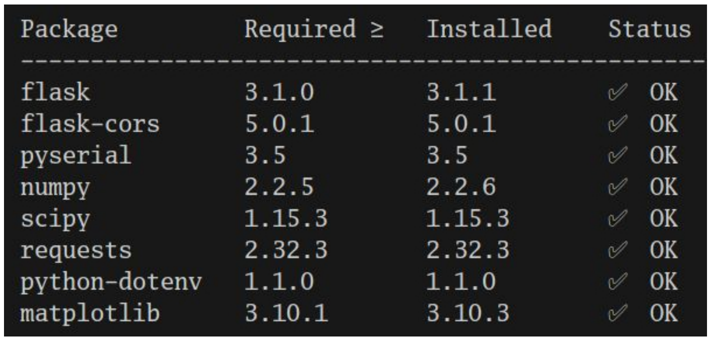
3.4 Test Functions and Utilities

3.4.1 Package Check

The package check utility verifies that all dependencies meet the required version specifications, preventing compatibility issues. It generates a formatted tabular output with clear visual indicators for each package's status: successfully meeting requirements (), outdated versions requiring updates (Too Old), or missing packages that need installation (Not Found).

3.4.2 Serial Screening

The serial screening utility provides direct access to the raw data stream from connected Arduino hardware, bypassing the application's processing pipelines. It establishes a serial connection to the configured port using the same parameters as the main application (9600 baud) and continuously reads, parses and displays each data frame in real-time. This tool was particularly useful during initial deployment to confirm proper hardware operation before launching the complete application stack.



```
Package          Required ≥   Installed    Status
----------------------------------------------------
flask            3.1.0        3.1.1        ✅ OK
flask-cors       5.0.1        5.0.1        ✅ OK
pyserial         3.5          3.5          ✅ OK
numpy            2.2.5        2.2.6        ✅ OK
scipy            1.15.3       1.15.3       ✅ OK
requests         2.32.3       2.32.3       ✅ OK
python-dotenv    1.1.0        1.1.0        ✅ OK
matplotlib       3.10.1       3.10.3       ✅ OK
```

Figure 4: Package Check

3.4.3 FFT Visualization

The FFT visualization tool enables detailed analysis of sensor data characteristics through both time and frequency domain representations. It captures a configurable duration of acceleration data (default 30 seconds) using the same SerialHandler interface as the main application, ensuring consistency between visualization and production operation. The tool produces a dual-panel plot showing raw acceleration values (in m/s^2) against time in the upper panel and the corresponding frequency spectrum via Fast Fourier Transform in the lower panel. The visualization highlights the detected dominant frequency with a vertical marker. This dual representation allowed us to correlate time-domain patterns with their frequency components, aiding in algorithm refinement and system tuning.

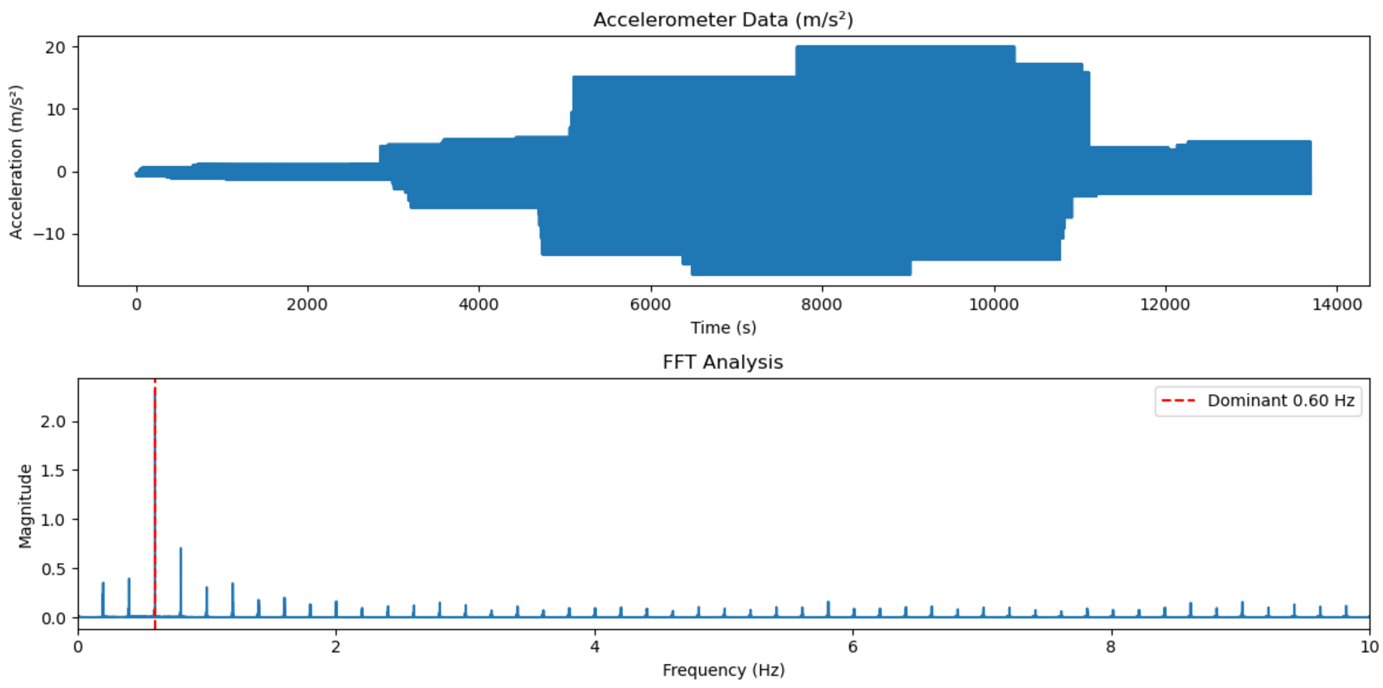


Figure 5: FFT Visualization

3.4.4 Cleaning

The cleaning utility implements system maintenance functionality through a shell script that removes Python cache files. This utility improves development workflow by preventing stale bytecode from causing unexpected behavior after code changes, particularly when performing major refactoring.

4 Frontend

This section focuses on explaining the part of the project that is responsible for visualizing the data we gathered in real-time and making it accessible to the public through a web application.

4.1 Tools and Frameworks Used

The application is built using several different tools and frameworks with the goal to create a modern and intuitive user experience. In more depth, we used **React** along with **TypeScript**, which serves as the foundation for the component-based architecture that we used, providing a responsive and interactive user interface, making it feel closer to a native application instead of a traditional website. TypeScript is used to add static typing, in order to enhance code reliability, readability as well as maintainability. We also used several libraries for enchancing the user experience, like:

- **Recharts**: Powers the data visualization for the wave height monitoring with interactive charts
- **Lucide React**: Supplies the iconography used throughout the interface for weather conditions, status indicators and UI elements
- **Framer Motion**: Provides fluid animations and transitions for alerts and UI elements
- **Tailwind CSS**: Implements custom styling with utility-first classes, creating consistent design patterns across components

This technology stack creates a fast, responsive application capable of handling real-time data visualization with minimal development overhead.

4.2 Look and Feel

The application follows a modern weather dashboard aesthetic with several notable design choices.

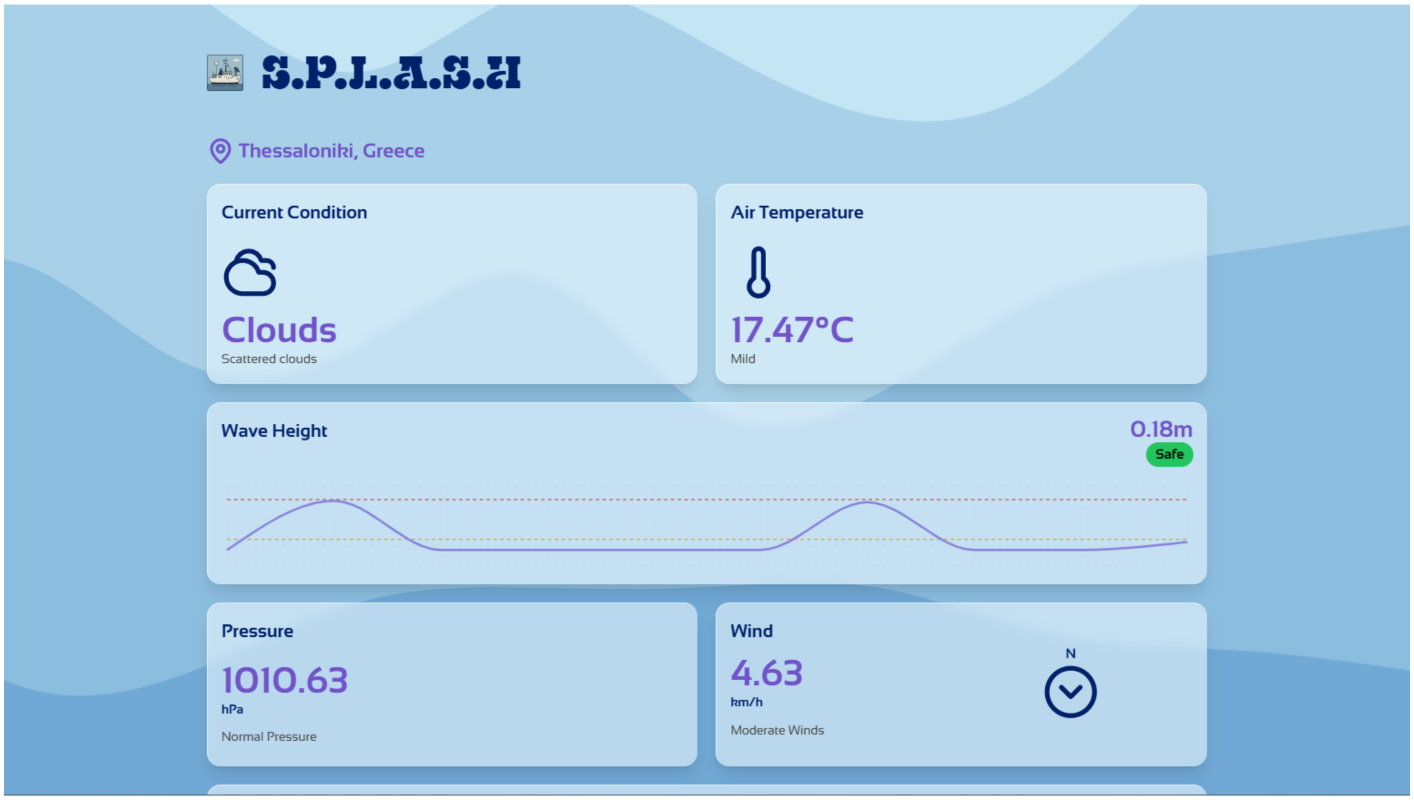


Figure 6: Web application look

More specifically, the website follows a *Glassmorphic* design inspired by popular weather websites, making it easy and familiar for the user to understand the data shown.

4.2.1 Component Based Design

The website used native widgets to display information. Each widget is has a title, explaining to the user what it shows and also gives descriptions for what the data translates to in the real world (i.e. comments if current pressure is above or below average). Widgets like *Current Condition* and *Wind* also have dynamic icons that change in accordance to the conditions that it shows. The widgets that are currently available to our application are:

- The *Current Condition* widget, which changes it's icon as well as color if the weather is *Cloudy*, *Raining* or *Sunny*
- The *Air Temperature* widget, that displays the current temperature as measured from the drone's on board sensor
- The *Pressure* widget, showing the atmospheric pressure as measured from the drone's on board sensor
- The *Wind* widget, showing a 'compass' icon along with the wind speed, that changes to the actual direction of the wind at the location of our drone.
- The *Avg Dist to Shore* widget, displaying the projected distance that the average wave will reach to the shore
- The *Drone Status* widget, giving valuable information about the drone itself, like battery and signal strength status
- The *Wave Height* widget, shows the actual height of the wave from sea level as well as a chart of the last 10 values, making the information more readable and useful to the user, giving a sense of scale to each measurement.

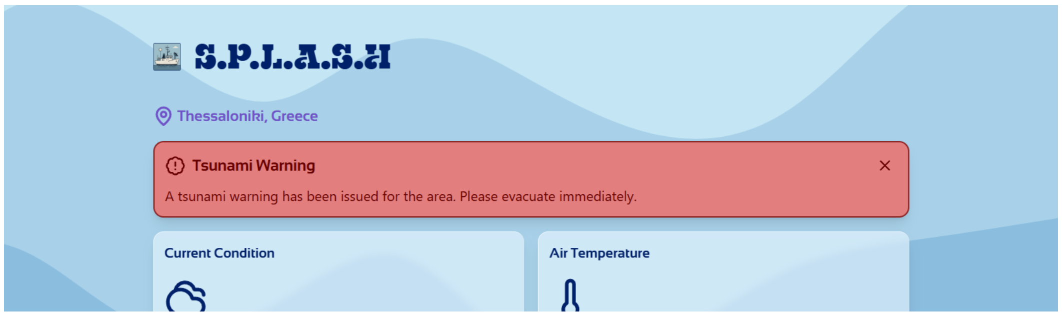
This design philosophy allows us to easily add new widgets to our application on future updates making it modular while keeping a modern look.

4.2.2 Notifications and Warnings

The web application also has the ability to notify the user with alerts regarding extreme weather events or other events regarding the status of the drone. More specifically these notifications are categorized based on their priority:

- **Low priority**: At its current state, low priority notifications may be for example low signal strength or if the application fails to connect with a backend server
- **High priority**: The notifications warn the user for possible extreme weather events, like a tsunami in the case that a abnormally high wave appears

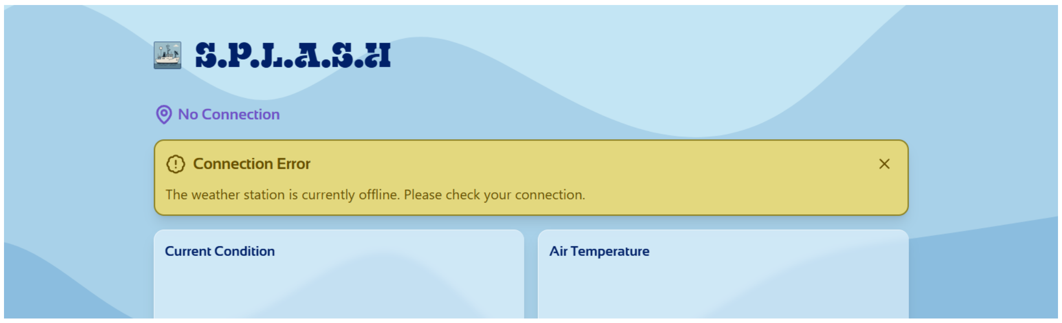
Low priority notifications are indicated with a *yellow* color, while high priority notifications are showed in *red*.


S.P.L.A.S.H

Thessaloniki, Greece

Tsunami Warning

A tsunami warning has been issued for the area. Please evacuate immediately.


S.P.L.A.S.H

No Connection

Connection Error

The weather station is currently offline. Please check your connection.

Figure 7: Notifications

4.3 Data API Integration

4.3.1 API Requests

Each widget handles API requests independently. Different widgets fetch data at different time intervals. This approach, while it isn't the most efficient when making API calls, makes the system more modular when creating new widgets and expanding the functionality of the application. More specifically:

- The *Current Condition* and *Wind* widgets, make API requests to the backend server every *3* minutes. This is done because our current approach relies on an external API for collecting this data and there is a limit to the number of requests the system can make daily.
- The *Wave Height* widget, makes request to our backend request every second, as this makes its corresponding graph more responsive and informative
- The rest of the widgets currently implemented, have intervals set at *30* seconds, as they do not rely on any external APIs and don't overload our backend server

Lastly, low as well as high priority alerts have requests interval set at *10* seconds, ensuring that notifications can arrive to the user in time.

4.3.2 Simulated and Live Modes

Considering the difficulties associated with setting up a drone and backend infrastructure for a live demo, we have also set up the ability for the user to test our application with the help of an always online backend server, providing the frontend with simulated data. This setting can be changed any time at the bottom of the main page. If the user wants to test a real implementation of our project, they can do so, be pressing the 'Live' button and filling in the IP and port number of their backend server.

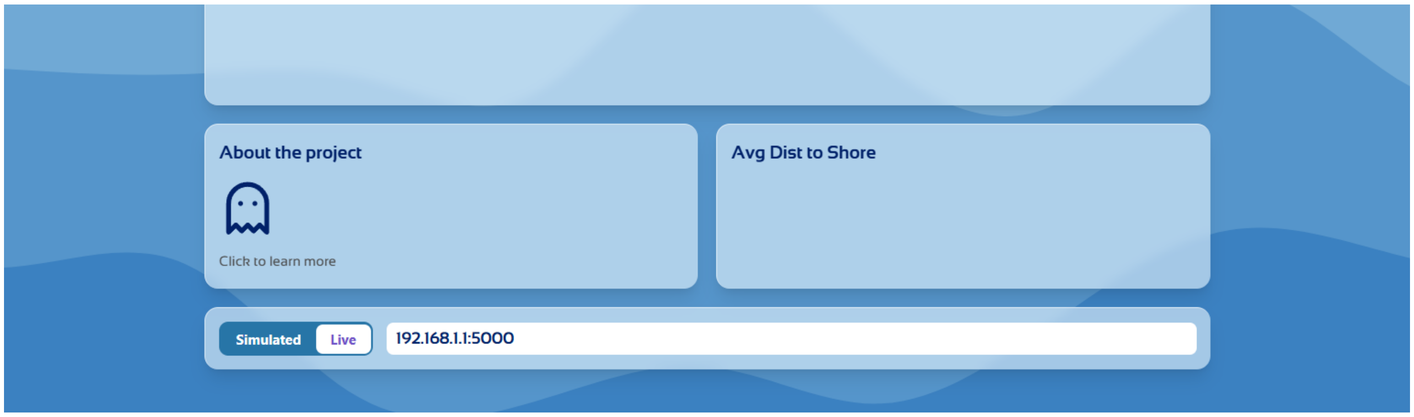


Figure 8: Live/Simulated Mode

5 Conclusions

5.1 Summary of Achievements

The *SPLASH* project successfully delivers a functional, low-cost and modular system for real-time maritime environmental monitoring.

A fully operational Arduino-based transmitter node was designed, integrating BMP280, MPU6050 and NEO-6M sensors to capture temperature, atmospheric pressure, vertical acceleration and geolocation data. Wireless communication was established using RF22 modules operating at 440 MHz with GFSK modulation, enabling efficient and reliable data transfer between the transmitter and receiver.

The backend system supports both simulated and real-time sensor data ingestion. It features a resilient serial communication handler, error recovery mechanisms and a rolling time-series buffer for acceleration data. Data processing includes an FFT-based wave analysis module capable of estimating dominant wave frequency and height. This was further extended with a physics-based model to compute coastal impact metrics and classify flood risk zones.

In addition to the core system, a range of supporting tools was developed, including utilities for serial data inspection, FFT visualization, dependency checking and codebase cleaning. The overall architecture emphasized modularity, allowing for easy expansion with additional sensors, backend features, or frontend components in the future.

A set of API endpoints was created to expose system data, including status, environmental conditions, wave characteristics and third-party weather information. These endpoints formed the foundation for a frontend web application. The frontend employed a modern glassmorphic design and interactive widgets to visualize data in real time, supporting both simulated and live data modes.

This set of accomplishments demonstrates the successful implementation of a scalable and user-friendly system for coastal monitoring.

5.2 Lessons Learned

Through the implementation of this project, we gained valuable experience in building a cyber-physical system and became more adept at capturing, interpreting and analyzing sensor data. Our programming and engineering skills improved significantly as we tackled challenges such as setting up a backend web server, developing a modular API architecture and crafting both the physical prototype and a waterproof enclosure. We also deepened our understanding of wireless communication, sensor integration and data visualization. This project provided us with a comprehensive, hands-on opportunity to bridge hardware with software and deliver meaningful environmental insights through a full-stack solution.

5.3 Future Work

Future improvements to the system may include:

- Implementing a mesh network of drones for wider coverage and resilience
- Integrating additional environmental sensors (e.g., humidity)
- Embedding the hardware into a single hydrodynamic boat
- Developing an automated navigation and docking system
- Utilizing solar power for energy autonomy
- Adding proper battery level monitoring to the `/status` endpoint

A User Manual

How to operate *SPLASH*:

- Power on the Arduino-based sensor module
- Ensure the receiver is connected to a computer and listening via serial
- Launch the backend server
- Open the web application to view real-time data

B Bill of Materials

This is a *rough* estimation of the total cost of the project.

Item	**Quantity**	**Price (€)**
Arduino Uno R3	2	25
RF22 card	2	32
Waterproof enclosure	1	22
Powerbank	1	10
NEO-6M GPS	1	7
MPU6050 Sensor	1	3
BMP280 Sensor	1	2
USB Power Cable	1	1
Jumper Wires	10	0.1
Prototyping board	1	.5
Total		**160.5**

Table 1: Bill of Materials

C Tools and Frameworks

The following tools and frameworks were used:

- Arduino IDE (Sensor Interaction)
- Python with Flask (Backend API)
- TypeScript with React (Frontend UI)

D Videos

- DEMO
- SPOT

References

[1] Stockdon, H. F.; Holman, R. A.; Reichmuth, A.; Van Ormondt, M. Estimation of shoreline position and change using airborne LiDAR data. *Coastal Engineering* **53**(3), 2006.

[2] Federal Emergency Management Agency (FEMA). Coastal Floodplain Mapping Report. November 2022.