October University for Modern Science & Arts Faculty of Engineering

Department of Computer Systems Engineering

IoT Wireless Earthquake Station

A Graduation Project
Submitted in Partial Fulfillment of the Requirements of

B.Sc. Degree in Computer Systems Engineering

Part II

Prepared By
Abdulrahman Alsayed Sallam 212421
Mohamed Waleed Abdelfattah 203659

Supervised By
Dr. Ehab Salaheldin Awad

2024/2025

Abstract

This project presents a cost-effective IoT-based Earthquake Station designed to detect seismic activity using an accelerometer and Wi-Fi for real-time data transmission. The system measures ground motion and operates only during seismic events, ensuring optimal energy efficiency, an essential feature for remote or off-grid locations. The system is only activated when needed. The project emphasizes reliable earthquake detection, efficient communication, and energy-saving techniques, adhering to a streamlined design approach. To address potential challenges, the agile development methodology is adopted, enabling flexibility and continuous refinement throughout the project lifecycle.

Acknowledgement

We sincerely thank Dr. Prof. Ehab Salaheldin Awad for his great guidance and support in this project and for his supervision of our project. We are also grateful to Dr. Prof. Ahmed Ayoub and Dr. Prof. Ahmed Alenany for their invaluable advice and help.

Contents

List of Abbreviations

ANN	Aritificial Neural Network
API	Application Programming Interface
AUC	Area Under Curve
AWS	Amazon Web Services
BMKG	Meteorology, Climatology, and Geophysical Agency
CRNN	Convolutional-Recurrent Neural Network
EEW	Earthquake Early Warning
ER	Entity Relationship
FFT	Fast Fourier Transform
GSM	Global System for Mobile communication
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
InSAR	Interferometric Synthetic Aperture Radar
IoT	Internet of Things
Kbps	Kilo bits per second
LTA	Long-Term Average
Mbps	Mega bits per second
MEMS	Micro-Electro-Mechanical Systems
MPU	Motion Processing Unit
MTBF	Mean Time Between Failures
ROC	Receiver Operating Characteristic
STA	Short-Term Average
STM	Synchronous Transport Module
SVD	Singular Value Decomposition
TP-link	Twisted Pair link
UART	Universal Asynchronous Receiver-Transmitter
UDP	User Datagram Protocol
UI	User Interface
WEA	Wireless Emergency Alert

List of Figures

List of Tables

Chapter 1

Introduction

1.1 How Earthquakes Work

Earthquakes occur due to the sudden release of energy along faults in the Earth's crust, typically caused by tectonic plate movements [1]. This release generates seismic waves that travel through the Earth, carrying vital information about the earthquake's origin, magnitude, and intensity. Seismic waves are classified into three main types [2]:

- **P-waves:** The fastest seismic waves that travel through both solid and liquid layers of the Earth. They cause minimal damage but serve as the first indicators of an earthquake.
- **S-waves:** Slower than P-waves and capable of moving only through solid materials. These waves cause significant ground shaking and structural damage.
- **Surface Waves:** The slowest yet most destructive waves, responsible for the majority of structural damage during an earthquake.

1.2 Methods of Earthquake Detection

Traditional earthquake detection relied on **seismographs**, stationary instruments that record ground motion. Developed in the 19th century [3], these instruments provided the first quantitative measurements of earthquake intensity and location. However, they required **centralized processing** and lacked real-time capabilities, limiting their effectiveness for early warning applications.

With advancements in digital technology, earthquake monitoring has evolved significantly. Modern detection systems utilize **accelerometers**, which measure ground acceleration and detect sudden seismic events in real time. **Micro-Electro-Mechanical Systems (MEMS) accelerometers** have made earthquake detection more compact, cost-effective, and easily deployable in diverse environments [4].

One of the most critical aspects of modern earthquake detection is **P-wave early warning systems**. P-waves, which travel faster than S-waves and surface waves, can be detected before the more destructive shaking begins. By analyzing P-wave characteristics, it is possible to estimate the earthquake's magnitude and provide early warnings **seconds to minutes** before the main shaking occurs [2].

1.2.1 Project Relevance and Contributions

Building upon these principles, our project aims to develop a **cost-effective, energy-efficient, and scalable IoT-based earthquake detection system**. By leveraging **real-time P-wave analysis, signal processing techniques (such as Fast Fourier Transform), and efficient data transmission**, our system enhances early warning capabilities. Unlike traditional seismic networks that rely on centralized data processing, our system is designed for **distributed deployment**, enabling faster and more localized detection.

By integrating modern accelerometer technology, real-time data transmission, and advanced signal processing, this project contributes to the ongoing efforts in earthquake early warning systems, offering a reliable and practical solution for seismic monitoring.

1.3 Problem Definition

Earthquakes are among the most powerful and unpredictable natural disasters, with an average of 20,000 occurring globally each year [5]. Most of these are minor and barely noticeable, but around 15 major earthquakes annually are capable of causing widespread destruction in mere seconds [6]. A tragic example is the 2011 Tohoku earthquake in Japan, which triggered a massive tsunami, resulting in nearly 20,000 fatalities and billions of dollars in damage [7]. Earthquakes strike without warning, collapsing buildings, disrupting infrastructure, and leaving communities in chaos. With such immense risks to human life and economic stability, it is critical to develop systems that can mitigate their impact through early detection and warning mechanisms.

Although earthquakes cannot be predicted with certainty, early warning systems have been implemented in high-risk regions. Japan's Earthquake Early Warning (EEW) system [8], shown in Figure 1.1, detects initial seismic waves (P-waves) and provides warnings **seconds to minutes** before the arrival of more destructive S-waves [2]. These systems enable automated responses such as halting trains, stopping surgeries, and issuing evacuation alerts, significantly reducing injury and damage.

However, while these centralized EEW systems are highly effective, they rely on **large-scale sensor networks and extensive computational infrastructure**, making them costly and **inaccessible for low-income or remote areas**. Additionally, continuous data transmission and centralized processing introduce **latency and power consumption issues**, limiting their feasibility for battery-powered or IoT-based deployments.

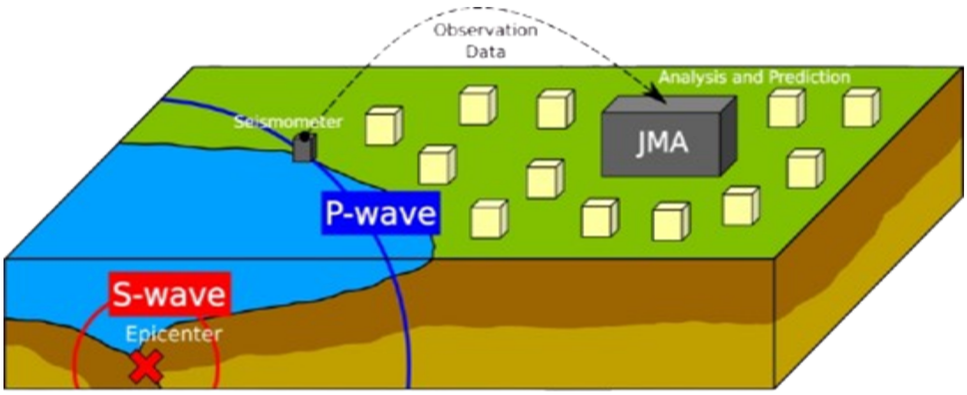

Figure 1.1: Block Diagram for Japan's EEW [9]

Research Gap & Project Motivation: To address these challenges, this project proposes a **cost-effective, energy-efficient,** and **scalable IoT-based earthquake detection system**. By leveraging **real-time P-wave analysis, optimized data transmission**, and **advanced signal processing techniques** (such as Fast Fourier Transform), this system aims to improve early warning capabilities while minimizing power consumption and infrastructure requirements. Unlike traditional EEW systems, the proposed solution focuses on **local detection and decentralized processing**, allowing for rapid response even in resource-constrained environments.

1.4 State of the Art

1. **IoT-Based Earthquake Detection Systems:** Many modern earthquake detection systems, including our project, Mugla Journal IoT-based earthquake warning system[10], Lovely Professional University earthquake alert system [11], and Perwej et al.'s IoT-based seismic monitoring system [12], they all leverage IoT technologies to monitor seismic activity in real time. These systems utilize networks of accelerometers and other sensors to detect ground motion and transmit data for analysis. IoT-based earthquake detection systems offer several key advantages:

    - **Scalability:** IoT networks enable the deployment of large-scale sensor arrays across various geographical regions, improving earthquake detection coverage.
    - **Cloud Integration:** Many systems integrate with cloud computing for centralized data processing, storage, and real-time accessibility.
    - **Versatile Applications:** These systems are used for various purposes, including **early warning alerts, scientific research**, and **structural health monitoring**.

2. **Mobile-Based Earthquake Alert Systems:** Systems such as ShakeAlert® in the United States [13], Ben et al.'s earthquake detection using GSM [14] and early warning system in mobile-based impacted areas [15] utilize smartphone networks and a combination of ground-based seismometers and mobile sensors to provide real-time earthquake warnings. These systems have proven to be **highly effective** but require significant infrastructure and government support for large-scale deployment. Key advantages of mobile-based alert systems include:

    - **Direct User Notifications:** Alerts are delivered instantaneously to users via mobile applications, ensuring rapid dissemination of warnings.
    - **Leveraging Existing Infrastructure:** These systems make use of cellular networks, making them highly **scalable and cost-effective**.
    - **Optimized for Urban Areas:** Mobile-based alerts work best in densely populated regions with **robust communication infrastructure**, where quick warnings can help mitigate casualties.

3. **Artificial Intelligence-Based Earthquake Detection Systems:** Recent advancements in earthquake detection have incorporated **Artificial Intelligence (AI)** and **Machine Learning (ML)** techniques to enhance accuracy and reduce false alarms. Systems such as **Khan et al.'s earthquake detection using supervised machine learning and novel feature extraction** [16] utilize **Artificial Neural Networks (ANNs)** to classify seismic events based on extracted features from accelerometer data. These systems analyze time-series seismic data using feature extraction methods to differentiate between real earthquakes and background noise.

    Key Advantages of AI-Based Systems:

    - **Improved Classification Accuracy:** AI models can learn from vast amounts of seismic data, improving their ability to distinguish between actual earthquakes and false alarms.
    - **Adaptive Learning:** Machine learning models can be retrained with new data, making them adaptable to different seismic conditions and regions.
    - **Feature Engineering for Enhanced Detection:** The integration of advanced feature extraction techniques (e.g., **FFT, SVD, MAX_ZC, and MIN_ZC**) enhances the model's ability to capture key earthquake characteristics.
    - **Reduced False Positives:** Unlike threshold-based detection methods, AI systems can incorporate multiple features to refine decision-making, reducing false alarms caused by environmental vibrations.

Limitations of AI-Based Systems:

- **Computational Cost:** Training and deploying AI models require **higher processing power**, which may not be feasible for low-power embedded systems.
- **Need for Large Datasets:** ML models require extensive labeled seismic datasets for training, which may not always be readily available.
- **Potential Overfitting:** If not properly trained, AI models may overfit to specific types of seismic activity, reducing generalization to new events.
- **No Current Reliable AI Based Warning:** Currently due to how unpredictable earthquakes are, there's no high accuracy AI's that have been designed to create early warnings before the p-waves, however as AI continues to evolve in the future it can be used to predict earthquakes before they occur but to today standards unfortunately they can't be used reliably, for example the best AI results for earthquake prediction before the P-wave was done by researchers at the university of Texas, where they were able to predict up to 70% of earthquakes before they happen in China, however their AI would only work in a specific area in china, they said "We're not yet close to making predictions for anywhere in the world, but what we achieved tells us that what we thought was an impossible problem is solvable in principle" [17].

1.5 Aim and Objectives

We aim to To develop a cost-effective, energy-efficient, IoT-based earthquake detection system that leverages P-wave data to predict the final earthquake magnitude (Richter scale) in real time providing early warning before the more destructive S-waves arrive. We would achieve our aim by following these objectives:-

- **Implement Real-Time Signal Processing:** Design and apply filtering, Fast Fourier Transform (FFT), and Short-Term Average over Long-Term Average (STA/LTA) algorithms to extract relevant seismic features from acceleration data in real time.
- **Develop Magnitude Estimation Algorithm:** Implement a method for computing Richter magnitude using P-wave and S-wave arrival times and peak ground acceleration from sensor input.
- **Integrate Hardware and Software:** Program the STM32 microcontroller in C to manage sensor input, process data locally, and transmit relevant seismic data through a Wi-Fi-enabled IoT framework.
- **Implement Event-Driven Activation Logic:** Design a vibration-triggered mechanism that activates the system only upon detection of potential seismic motion, reducing unnecessary processing cycles.

- **Support Multi-Station Operation:** Architect the system to allow deployment of multiple independent detection nodes, with coordinated data aggregation for broader seismic coverage.
- **Develop a Real-Time Monitoring Interface:** Build a responsive web-based interface to visualize live seismic data, including detected events, sensor status, and historical trends.
- **Evaluate System Accuracy and Reliability:** Compare the system's seismic event detection and magnitude estimation results with validated datasets, and iteratively adjust filtering thresholds and model parameters for improved performance.

1.6 Contributions and Uniqueness

- **Cost-Effective Sensor Design:** The system utilizes only an accelerometer rather than combining it with a gyroscope, reducing hardware costs while maintaining adequate accuracy through advanced signal processing techniques.
- **Energy-Efficient Operation:** A vibration-triggered mechanism ensures that the system activates only when seismic activity is detected. This targeted power usage conserves energy, making it ideal for deployment in remote or off-grid locations.
- **Real-Time P-Wave Analysis for Early Warning:** By processing P-wave data in real time, the system can predict the final earthquake magnitude before the arrival of the more destructive S-waves, thus providing crucial extra seconds for early warning.
- **Robust IoT Integration:** The system is built on an STM32 microcontroller, which offers low-level control and energy efficiency, coupled with Wi-Fi connectivity for real-time data transmission. This design supports centralized monitoring and scalable deployment across distributed sensor networks.
- **Advanced Feature Extraction:** Beyond standard metrics such as RMS acceleration, the system extracts detailed P-wave features including peak acceleration, duration, energy content, and frequency metrics (both peak and dominant frequencies) to enhance the accuracy of magnitude prediction.
- **Adaptability and Scalability:** The modular design allows for the easy integration of additional sensors or advanced algorithms (e.g., machine learning) in future iterations. This ensures that the system can be scaled and adapted to a variety of environmental conditions and evolving requirements.

1.7 Thesis Structure

This thesis is structured as follows:

- **Chapter 1 - Introduction:** Provides an overview of the project, including its objectives, significance, and a comparison with existing earthquake detection systems.
- **Chapter 2 - Literature Review:** Examines the principles of earthquake detection, signal processing techniques, and the evolution of seismic monitoring technologies.
- **Chapter 3 - Project Background:** Establishes the context and motivation behind the development of the earthquake detection system, highlighting real-world challenges, limitations of existing approaches, and the unique advantages offered by the proposed solution.
- **Chapter 4 - System Design:** Defines the system architecture, hardware components, software framework, and design specifications of the proposed earthquake detection system.
- **Chapter 5 - System Implementation:** Describes the hardware and software implementation, including microcontroller firmware development and the web-based monitoring platform.
- **Chapter 6 - Testing and Evaluation:** Discusses the testing methodologies, test cases, and evaluation results to validate the accuracy, reliability, and efficiency of the system.
- **Chapter 7 - Cost Analysis:** Provides a financial assessment of the system, covering hardware costs, operational expenses, and scalability considerations.
- **Chapter 8- Time Plan:** Showcases the main goal points of the project and how long the task will take with the usage of gantt chart.
- **Chapter 9 - Conclusions and Future Work:** Summarizes the findings, evaluates the project's impact, and outlines potential future improvements to enhance system performance.

Chapter 2

Literature Review

2.1 History

The field of earthquake detection has undergone a remarkable evolution over the past two centuries. In its earliest days, detection relied on traditional seismographs—stationary instruments designed to record ground motion during seismic events. Developed in the 19th century, these early seismographs revolutionized our understanding of earthquake dynamics by providing the first quantitative measurements of seismic intensity and location. However, these instruments were limited by their immobility and reliance on centralized systems, which significantly hampered real-time detection and timely warning dissemination [18].

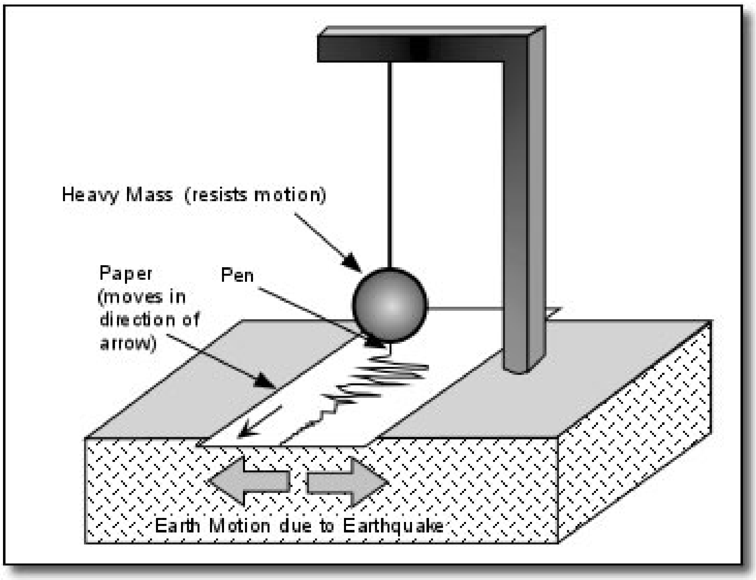

Figure 2.1: Model for 19th Century Seismographs [3]

The advent of digital technology in the late 20th century marked a major turning point. Computerized monitoring systems emerged, enabling faster and more accurate data processing. This digital revolution in seismology paved the way for the development of advanced methods capable of near real-time analysis of seismic events. Among the most significant breakthroughs was the invention of Micro-Electro-Mechanical Systems (MEMS) accelerometers. These sensors, with their compact size, low cost, and high sensitivity, have enabled the creation of scalable, IoT-based earthquake detection systems that are not only cost-effective but also capable of providing rapid alerts [4].

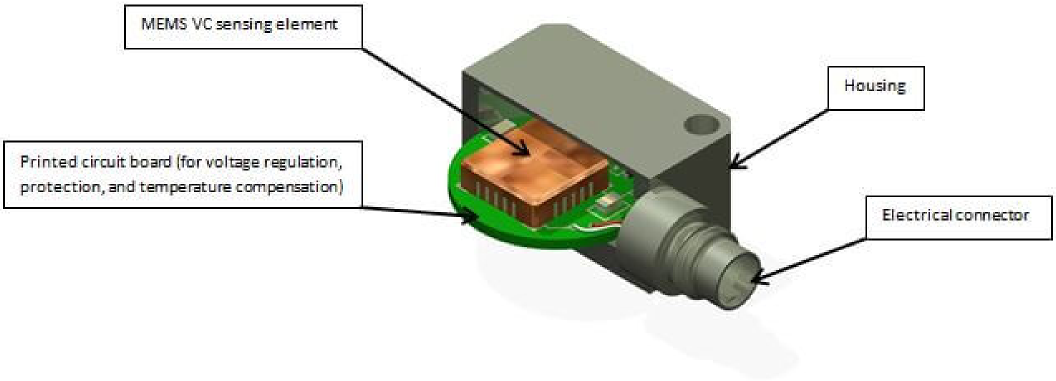

Figure 2.2: MEMS Accelerometer Model Diagram [19]

Today, modern earthquake detection systems integrate IoT frameworks, advanced signal processing, and real-time data analytics to monitor seismic activity continuously. This evolution from traditional, stationary seismographs to sophisticated digital systems underpins the development of innovative solutions like our proposed system, which combines cost efficiency with enhanced real-time capabilities and energy-saving operation.

2.2 Survey of Earthquake Detection Systems

In this section, we review several earthquake detection systems available in the literature. Each system is discussed in its own section to highlight its design, advantages, and limitations relative to our proposed solution.

2.2.1 Mugla Journal IoT-Based Earthquake Warning System

The system proposed in the Mugla Journal [10] presents an IoT-based earthquake early warning solution that monitors seismic activity in real time. It adopts a dual-sensor approach, integrating both an accelerometer and a gyroscope within an inertial measurement unit (IMU) to detect ground motion. This combination allows the system to capture both linear acceleration and angular velocity, thereby enhancing its sensitivity and ability to differentiate between various types of movement.

A key strength of the Mugla system is its effort to reduce false positives. This is achieved by deploying two identical sensor nodes in the same geographic area; an earthquake alert is only triggered if both nodes detect seismic activity concurrently. This redundancy minimizes false alarms from non-seismic events such as nearby construction or vehicular traffic. Once seismic activity is detected, the system transmits data via a Wi-Fi module to the ThingSpeak IoT analytics platform. ThingSpeak processes the data and disseminates earthquake alerts via Twitter, marking the beginning and end of the seismic event.

However, the system also presents notable drawbacks. First, the inclusion of a gyroscope increases hardware complexity and cost. Second, its reliance on ThingSpeak, a third-party cloud platform with a yearly subscription, introduces recurring costs and potential limitations in data ownership and customization. Most critically, the system does not estimate or classify the strength or magnitude of the earthquake, limiting its usefulness for damage prediction or severity assessment.

- **Advantages of the Mugla System:**
  - Improved detection accuracy through dual-sensor integration (accelerometer + gyroscope)
  - Redundancy via dual-node verification reduces false positives
  - Real-time alerting via ThingSpeak and Twitter integration
  - The usage of thingspeak eliminates the need for development on the server side making it simpler and faster release time.
- **Disadvantages:**
  - Higher one-time cost due to gyroscope inclusion
  - Higher recurring costs and limited customization due to reliance on ThingSpeak
  - No quantification of earthquake magnitude

- **Comparison with Our System:**

Our system opts for a more cost-effective and maintainable approach by using only an accelerometer, which sufficiently captures the necessary ground motion data for detection while only a very slight reduction to sensitivity compared to a dual-sensor setup, it significantly lowers the overall cost and complexity of the hardware. Furthermore, unlike the Mugla system, we do not depend on external platforms like ThingSpeak. Instead, our system processes and stores data on a self-hosted server, eliminating subscription fees and offering greater control and data privacy.

Most importantly, our system includes an AI-powered module capable of determining whether a seismic activity is due to an earthquake or not based on acceleration patterns. This added layer of analysis addresses a critical shortcoming of the Mugla system and allows our solution to provide more actionable information for early warning and emergency response planning, our system is also capable of calculating the richter magnitude of an earthquake while mugla's system can't.

Figure 2.3 illustrates the general design of the Mugla Journal IoT-Based Earthquake Warning System.

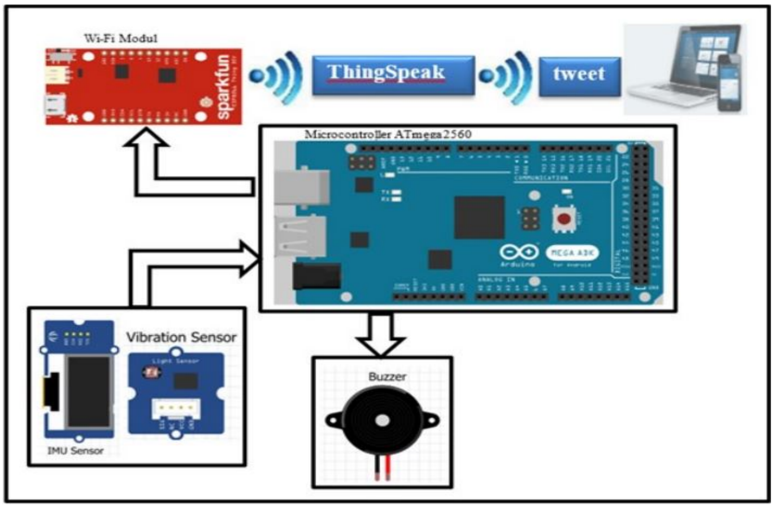

Figure 2.3: Mugla Journal IoT-Based Earthquake Warning System General Design [10]

2.2.2 Lovely Professional University Earthquake Alert System

The Earthquake Alert System developed by Lovely Professional University [11] is an IoT-based platform designed for real-time seismic monitoring and rapid dissemination of earthquake alerts. The system utilizes the ADXL335 accelerometer to capture ground motion, with data processed locally by an Arduino Uno equipped with an ATmega328P microcontroller. Seismic events are detected through a threshold based algorithm. When the sensed acceleration exceeds a predefined limit, the system transmits the data using a NodeMCU Wi-Fi module to the ThingSpeak cloud platform. On ThingSpeak, the data is visualized, logged, and used to trigger alert mechanisms.

One of the notable features of this system is its use of wireless sensor networks and real-time cloud communication to notify users through smartphones and web-based interfaces. This facilitates quick alerts and easy access to seismic data. However, the system's detection logic is primarily based on thresholding high-amplitude accelerations, corresponding to the onset of strong shaking (S-waves). As a result, it lacks the capability to detect the initial, lower-magnitude P-waves that precede destructive earthquakes, reducing its effectiveness as a true *early* warning system.

Additionally, reliance on the ThingSpeak cloud introduces recurring costs and restricts flexibility in customizing alert logic, data ownership, and integration with other systems. Furthermore, the system does not perform any form of magnitude estimation or advanced event classification, limiting its utility in post-event assessment and risk mitigation.

- **Advantages of the Lovely Professional University System:**
  - Simple and cost-effective design using widely available components (e.g., ADXL335, Atmega328p, NODE MCU)
  - Real-time alerts through ThingSpeak and smartphone/web interfaces
  - Straightforward implementation of threshold-based detection
- **Disadvantages:**
  - No detection of early P-waves, reducing warning time
  - No estimation of earthquake magnitude or severity
  - Dependence on ThingSpeak introduces recurring subscription costs and limits system customization
  - Threshold-only detection may lead to missed detections or false positives
- **Comparison with Our System:**

While both systems are designed for real-time seismic monitoring, our system introduces several critical enhancements. First, instead of relying on only analog thresholding, our system incorporates an AI model trained to detect seismic activity patterns and incorporates an analog thresholding as well, which can reduce false positives and increase reliability. Additionally, we host our own backend server infrastructure, allowing full control over data storage, visualization, and alerting without incurring recurring cloud costs.

Unlike the Lovely Professional University system, our solution includes a module to estimate the intensity or magnitude of detected tremors, offering more context to users and responders. Although we also use an accelerometer-only setup for simplicity and cost efficiency, we overcome the limitations of thresholding by employing intelligent detection logic rather than fixed cutoffs.

Figure 2.4 presents the block diagram of the Lovely Professional University Earthquake Alert System, illustrating its sensor network, processing unit, and cloud communication architecture.

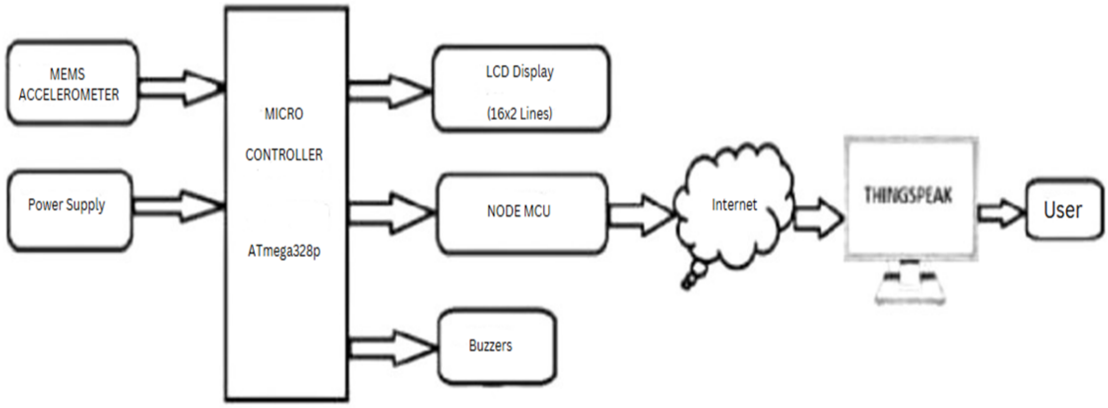

Figure 2.4: Lovely Professional University Earthquake System Block Diagram [11]

2.2.3 Perwej et al.'s IoT-Based Seismic Monitoring System

Perwej et al. [12] introduce an IoT-based seismic monitoring solution that utilizes a dual-sensor architecture comprising both accelerometers and gyroscopes. This approach enables the system to capture both linear and angular components of ground motion, significantly enhancing its ability to detect and characterize seismic events. By combining multiple data streams, the system can differentiate between actual earthquakes and non-seismic disturbances with higher accuracy.

One of the distinguishing features of this system is its capacity to detect P-waves relatively weaker seismic waves that travel faster and arrive before the more damaging S-waves. This early detection capability enables the issuance of alerts several seconds before significant ground shaking begins, which is crucial for public safety applications such as shutting down critical infrastructure or alerting populations at risk.

Sensor data is transmitted via an IoT communication protocol to a central processing hub, where advanced signal processing algorithms analyze the seismic activity. This architecture allows for scalable deployments, where multiple sensor nodes can be integrated into a wider network, improving both coverage and reliability. Furthermore, the system incorporates cross-verification from multiple sensors to reduce false positives, a technique particularly valuable in urban or industrial environments prone to background vibrations.

- **Advantages of Perwej et al.'s System:**
  - Dual-sensor setup (accelerometer + gyroscope) provides richer data and more reliable event detection
  - Capable of detecting P-waves for early warning before stronger shaking occurs
  - Reduced false positives through signal cross-verification
  - Designed to be cost-effective while offering real-time alerts
- **Disadvantages:**
  - Increased hardware complexity and cost due to the use of multiple sensors
  - Requires sophisticated signal processing infrastructure
  - May not be suitable for ultra-low-power or highly resource-constrained deployments
- **Comparison with Our System:**

While Perwej et al.'s system excels in sensitivity and early warning by leveraging both accelerometers and gyroscopes, our system intentionally simplifies the sensor design by relying solely on an accelerometer. This choice was made to minimize hardware costs and power consumption, making our solution more suitable for wide-scale, cost-sensitive deployments.

To compensate for the absence of gyroscopes, our system employs machine learning-based classification to differentiate between real seismic activity and noise, aiming to maintain high detection accuracy while keeping the hardware lean. Additionally, unlike Perwej et al.'s reliance on a centralized processing node, our system performs local preprocessing before transmitting to a self-hosted web server, offering full control over the architecture and reducing dependency on third-party platforms.

While Perwej et al.'s architecture may be better suited for early warning scenarios in critical infrastructure, our system emphasizes accessibility, affordability, and control, making it ideal for grassroots-level deployments, schools, small businesses, and public buildings.

Figure 2.5 illustrates the detailed block diagram of Perwej et al.'s seismic monitoring system, highlighting its multi-sensor integration and real-time communication flow.

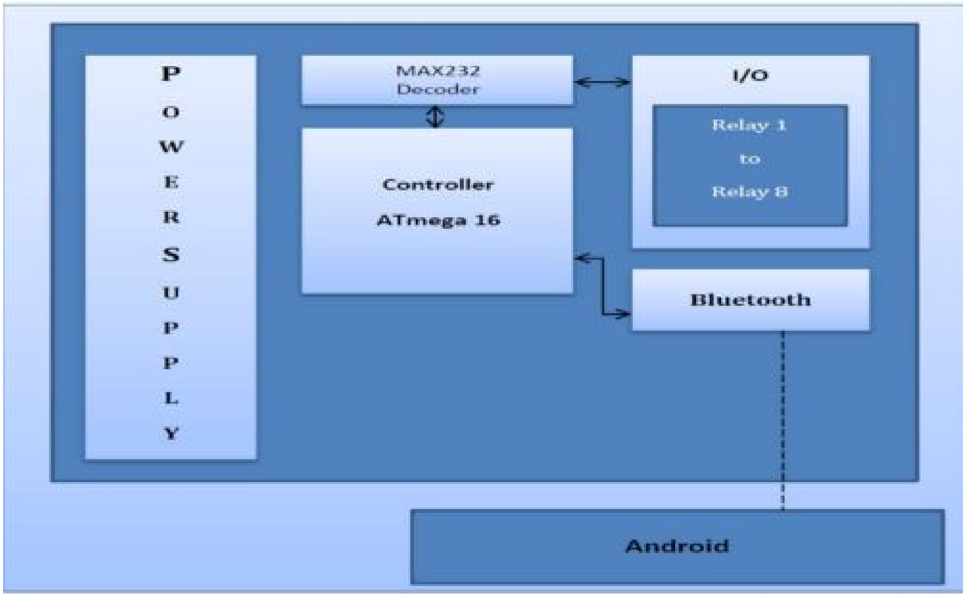

Figure 2.5: Perwej, Dr. Yusuf & Jitendra Earthquake System Block Diagram [12]

2.2.4 Ben et al.'s Earthquake Detection Using GSM

Ben et al. [14] propose a compact earthquake detection system that combines GSM communication with motion sensing via accelerometers and gyroscopes. This system is built around an Arduino microcontroller, which processes input from both types of sensors to detect seismic activity by capturing both linear and angular motion. Upon detection, the data is transmitted in real-time to a central monitoring unit using GSM technology, ensuring seismic events are promptly recorded and made accessible for further analysis.

A notable feature of this system is its hybrid communication strategy. While GSM is used for long-range data transmission, some configurations also support Wi-Fi for local connectivity, allowing flexible deployment in different environments depending on infrastructure availability.

However, the system primarily serves as a post-event monitoring tool, lacking the ability to issue pre-shaking alerts or detect early P-waves. This design choice simplifies implementation and reduces cost but limits its utility in applications requiring proactive safety measures, such as public warnings or automatic infrastructure responses.

- **Advantages of Ben et al.'s System:**
  - Utilizes both accelerometers and gyroscopes for enhanced motion detection
  - GSM module allows data transmission even in areas with limited internet infrastructure
  - Flexible configuration supporting GSM and Wi-Fi
  - Cost-effective and suitable for remote or underdeveloped areas

- **Disadvantages:**
  - Lacks early warning capabilities (no P-wave detection or prediction)
  - Primarily designed for data logging rather than immediate alerting
  - GSM transmission may incur additional operational costs (SIM fees, data plans)
- **Comparison with Our System:**

While Ben et al.'s system focuses on post-event data transmission and is well-suited for archival and academic purposes, our system is optimized for real-time detection and proactive response. Unlike their GSM-based alerts, our system relies on Wi-Fi to send data to a private web server, giving us complete control over the processing, alerting, and data visualization pipeline.

Additionally, by relying solely on an accelerometer and machine learning-based event classification, our system reduces hardware complexity and cost, making it more scalable for widespread deployment. While we also forego P-wave-based early warnings due to sensor limitations, our real-time alerts and low-latency architecture allow us to respond almost immediately to the onset of shaking.

In summary, Ben et al.'s system is advantageous in remote or resource-limited environments with cellular coverage but lacks the predictive and customizable alert features of our solution.

2.2.5 Earthquake Detection Using Supervised Machine Learning and Novel Feature Extraction

Khan et al. [16] present an innovative approach to earthquake detection that leverages supervised machine learning techniques combined with a novel feature extraction method. Unlike traditional earthquake early warning systems that aim to forecast seismic events before they occur, this system focuses on real-time classification determining whether the current ground shaking is due to an earthquake or is merely noise or non-earthquake activity.

The proposed method extracts a comprehensive set of features from time-series sensor data, including both amplitude-based and frequency-domain characteristics. This system uses three features which are inter-quartile range (IQR), zero crossing rate (ZC), and cumulative absolute velocity (CAV). IQR is the range of acceleration values between the 25th and 75th percentiles, ZC counts how many times the acceleration signal changes sign, representing a frequency measure, and CAV summarizes the total amplitude of the acceleration vector sum across all three components. They then use these three features in an ANN model with three neurons in the input layer, five neurons in the hidden layer and one neuron in the output layer as seen in figure 2.6, this model is trained to distinguish between genuine earthquake signals and various forms of background noise, achieving promising results on both the training dataset and unseen data.

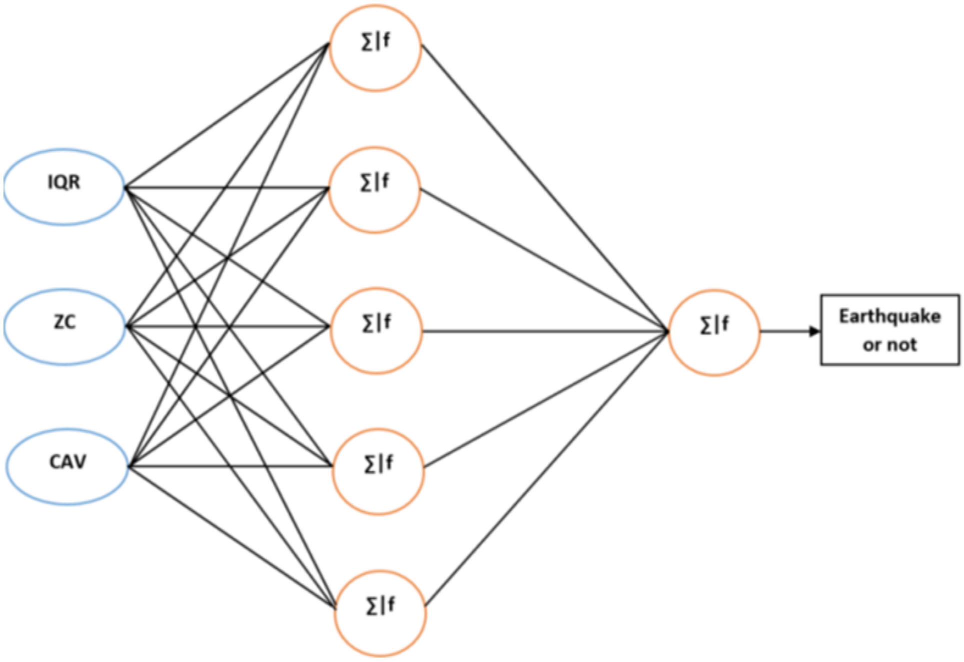

Figure 2.6: ANN Model of Khan et al. System [16]

A notable aspect of this study is its dual-environment approach: while the static model addresses challenges associated with sensor noise and fixed installations, the dynamic model is adapted to account for the complex vibratory patterns resulting from human activities. However, it is important to note that the system does not predict earthquakes before they occur—it merely classifies the current shaking as either an earthquake event or non-earthquake noise and then sends an alert to nearby devices using either Wi-Fi or Bluetooth.

They have tried six models with different features to see which would perform best, you can see the results in table 2.1.

Table 2.1: Earthquake Detection ANN Models Results

Model	TP	TN	FP	FN	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)	Features Used
Model 1	698	23,987	129	312	98.24	84.40	69.11	75.99	IQR, ZC, and CAV
Model 2	543	24,112	4	467	98.13	99.27	53.76	69.75	IQR, FFT, CAV
Model 3	815	23,536	580	195	96.92	58.42	80.69	67.76	IQR, ZC, FFT, CAV
Model 4	685	23,932	184	325	97.97	78.83	67.82	72.91	IQR, SVD_ZC, CAV
Model 5	781	23,814	302	229	97.89	72.11	77.33	74.63	IQR, ZC, CAV, SVD_Scale
Model 6	807	24,059	57	203	98.96	93.40	79.90	86.13	IQR, CAV, MAX_ZC, MIN_ZC, MAX_NON_ZC

In the table, you may notice some more features mentioned earlier, those are the SVD which is a linear algebra technique that breaks down a matrix into three simpler matrices, facilitating easier analysis and manipulation [20]. The SVD is used for different features in some of the models, these features are **SVD_Scale** which is the first singular value (S[0]) representing the average amplitude characteristic and **SVD_ZC** which is the zero-crossing rate calculated for the primary vector, denoted as U[:,0].

The results indicate that Model 6 is the most effective, achieving the highest accuracy (98.96%) and F1 score (86.13%), demonstrating a strong balance between precision (93.40%) and recall (79.90%). The inclusion of MAX_ZC, MIN_ZC, and MAX_NON_ZC features likely enhances its ability to distinguish seismic events from background noise. Model 2, while highly precise (99.27%), suffers from poor recall (53.76%), meaning it fails to detect a significant portion of earthquake events. Model 3 has the highest recall (80.69%) but at the cost of low precision (58.42%), leading to frequent false positives. Models 4 and 5 show a more balanced approach with moderate precision and recall but do not outperform Model 6. Model 1 is a strong contender with high accuracy (98.24%) but slightly lower recall (69.11%), meaning it still misses some earthquake events. Overall, Model 6 provides the most reliable performance, making it the best candidate for deployment, while further improvements could focus on refining feature selection to optimize both precision and recall.

Figure 2.7: Khan et al. Earthquake Alert Final Device [16]

- **Advantages**
  - Real-time classification using a lightweight ANN model makes it suitable for embedded systems.
  - Utilizes simple yet effective features (IQR, ZC, CAV) that are interpretable and computationally efficient.
  - Wireless communication via Wi-Fi or Bluetooth enables quick alerting without reliance on internet connectivity.
  - The dual static/dynamic model design accommodates both fixed installations and human-induced vibration environments.

- **Disadvantages**
  - The system does not predict earthquakes it only classifies current shaking.
  - Effectiveness depends heavily on training data quality and may not generalize well to unseen noise types.
  - Requires labeled datasets and feature engineering, unlike simpler threshold-based methods.
  - While their accuracy is high, thats due to the high amount of noise data compared to the earthquake data, so if the model decides to just outputs a negative prediction then it would output a high accuracy anyways making the model biased.
- **Comparison with Our System**

Khan et al.'s system and our proposed system share the overarching goal of distinguishing between genuine earthquake events and other types of ground motion. However, the two approaches diverge significantly in terms of architecture, implementation, and objectives. Khan et al.'s work utilizes a compact Artificial Neural Network (ANN) model that operates locally using hand-crafted features such as inter-quartile range (IQR), zero crossing rate (ZC), and cumulative absolute velocity (CAV). Their model performs real-time classification directly on the device and then sends alerts via Bluetooth or Wi-Fi to nearby users. This makes their system suitable for small-scale, localized deployments that require fast, on-site decision-making.

In contrast, our system is designed for centralized, server-based analysis and is aimed at reducing false alarms rather than merely detecting shaking. We use a Random Forest model, which is more robust and interpretable than the lightweight ANN used by Khan et al., and better suited to distinguishing subtle differences between actual seismic activity and spurious signals. Our system streams acceleration data from the device to a web server where processing and classification take place, allowing for more complex analysis, scalability, and easier updates. Additionally, our system is powered by a constant electricity source with a battery backup, enabling it to run continuously in fixed installations. While Khan et al.'s system excels in local, embedded intelligence, our approach emphasizes reliability, scalability, and remote accessibility, making it better suited for long-term deployment and integration with centralized monitoring infrastructures.

2.2.6 Early Warning System in Mobile-Based Impacted Areas

The Early Warning System in Mobile-Based Impacted Areas [15] is a mobile-based earthquake alert system developed to enhance the effectiveness of earthquake notifications and evacuation guidance. This project was designed to address the shortcomings of conventional warning systems, which rely on TV, radio, and websites, often resulting in delayed dissemination of critical earthquake alerts. The system works as shown in Figure 2.8.

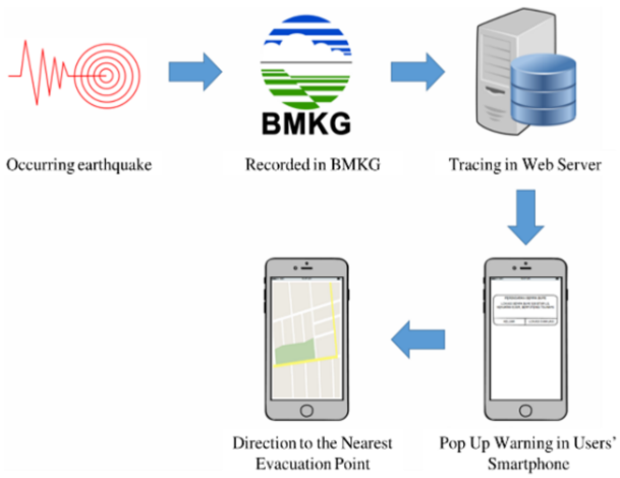

Figure 2.8: Direction Display to Evacuation Site Using Google Maps [15]

System Design and Functionality: The system leverages **mobile technology** and **real-time data processing** to deliver immediate earthquake warnings and evacuation directions. It integrates with:

- **Google Maps API:** Used to determine the user's location and provide navigation assistance to the nearest evacuation site as seen in Figure 2.9.

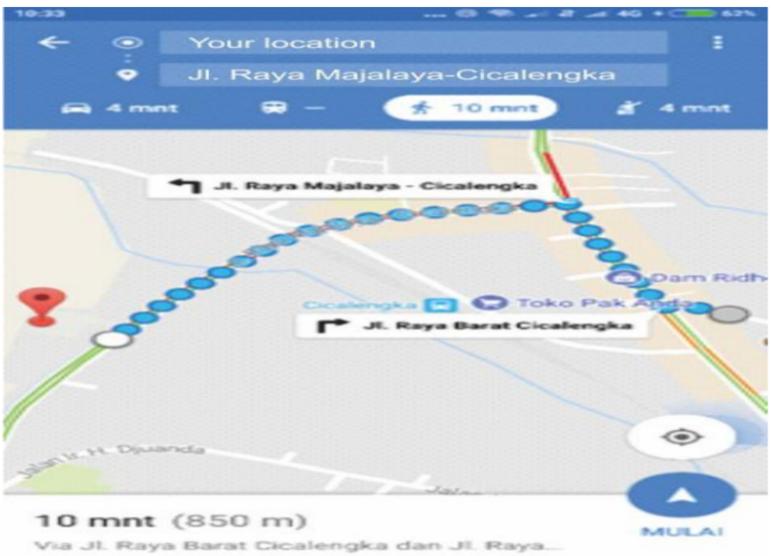

Figure 2.9: Direction Display to Evacuation Site Using Google Maps [15]

- **Firebase Cloud Messaging (FCM):** Enables instant notifications to users, accompanied by an alarm sound, ensuring that warnings are received promptly, as seen in Figure 2.10.

Figure 2.10: Notification Showing Earthquake Warning [15]

- **BMKG (Indonesian Meteorological Agency) Data:** The system fetches real-time earthquake data from BMKG's monitoring network as seen earlier in Figure 2.8.

Key Features of the System Include:

- **Instant Earthquake Alerts:** Users receive mobile notifications with earthquake details as soon as seismic activity is detected, as seen in Figure 2.10.
- **Evacuation Guidance:** The system identifies nearby evacuation centers and provides real-time navigation using Google Maps, as seen in Figure 2.9.
- **Historical Data Storage:** Past earthquake records are stored in the application's database for later review.
- **User-Friendly Interface:** Designed for accessibility, ensuring that even non-technical users can quickly understand and respond to alerts, as seen in Figure 2.11.

Figure 2.11: UI of their Mobile Application [15]

Advantages Over Conventional Systems: Compared to traditional earthquake warning methods, this mobile-based system ensures **faster and more reliable notifications**, particularly benefiting individuals who may not have immediate access to radio or television broadcasts. By utilizing smartphones, which are widely accessible, it significantly enhances community preparedness and response time during seismic events.

Figure 2.12 illustrates the architecture of the mobile-based early warning system.

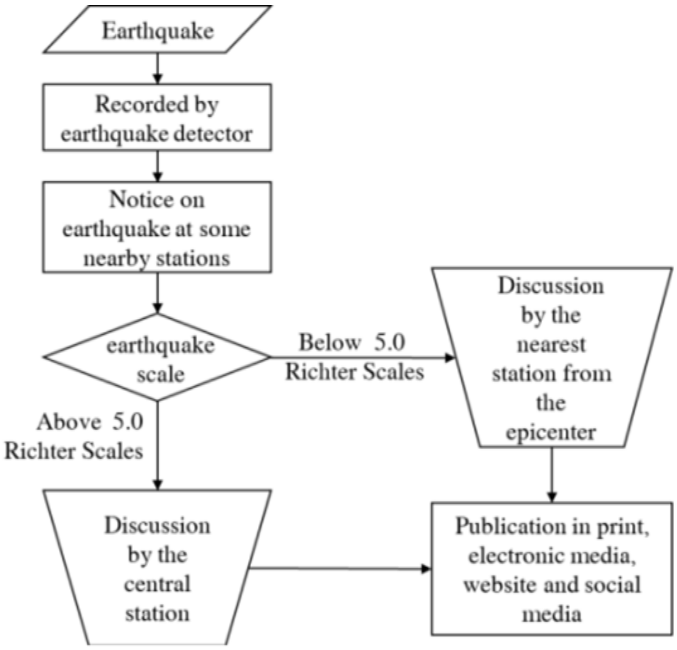

Figure 2.12: Mobile-Based Earthquake Early Warning System Architecture [15]

- **Comparison with Our System**
  While the mobile-based early warning system by Moch et al. excels in disseminating alerts and guiding evacuation through smartphone notifications, it presents several limitations. It relies heavily on mobile networks and external data sources such as BMKG. This dependence introduces potential delays in detection and notification, especially in areas with poor connectivity or during disasters when networks may be congested or offline. Additionally, the system lacks local sensing capabilities, meaning it cannot directly detect seismic activity at the user's location, which may reduce responsiveness to hyper-local events.

  In contrast, our system includes a physical accelerometer that captures seismic activity in real-time at the point of measurement, independent of internet latency or third-party data providers. This sensor data is analyzed locally using a Random Forest classifier to filter out false alarms and accurately identify earthquakes.

  While both of our systems make use of the **Google Maps API**, their usage is fundamentally different:

  - Moch et al. use Google Maps to provide users with real-time evacuation guidance.
  - Our system uses Google Maps within the web dashboard to **visualize the geographic location of each sensor station**, allowing users to correlate seismic activity with physical locations.

  Moreover, our solution is designed to be **web-based and infrastructure-resilient**, capable of operating in fixed installations such as schools, factories, or public buildings. Unlike mobile-first systems, which prioritize portability and individual alerts, our focus is on precision, autonomous operation, and centralized monitoring for safety-critical environments.

  While the mobile system offers mobility and user-facing features like historical browsing and navigation, our system delivers enhanced detection accuracy, lower latency, and deployment resilience—making it ideal for early-warning infrastructure that does not depend on mobile availability or cloud services.

2.2.7 Bassetti and Panizzi's IoT Crowdsensing Edge Network

Bassetti and Panizzi [21] propose a decentralized IoT-based earthquake detection system that leverages edge computing to overcome the limitations of centralized fusion-center-based architectures. Their system forms a partial mesh network where each detector node processes incoming data from probes and directly issues earthquake alerts via a gossiping protocol. Detection is done locally using a convolutional recurrent neural network (CrowdQuake), running on devices such as Raspberry Pi boards.

This architecture ensures resilience to node failure and improves user privacy by avoiding centralized cloud platforms. Data collected from NodeMCU-based probes (with ESP8266 and MPU6050) is streamed directly to a local detector, which also serves as the decision-making and alert-generating unit. The system is optimized for grass-roots, decentralized deployment.

- **Advantages of Bassetti and Panizzi's System:**
  - Fully decentralized mesh network with no single point of failure
  - Low-latency local detection via edge-based ML (CRNN)
  - Enhanced privacy by keeping data local
  - Cost-effective hardware with open-source implementation
- **Disadvantages:**
  - No estimation of earthquake magnitude
  - No support for P-wave detection
  - No central dashboard for remote monitoring or historical analysis
  - Very low sampling rate (10 Hz) which affects the accuracy
- **Comparison with Our System:**

While Bassetti and Panizzi's system emphasizes full decentralization, we chose a more modular, maintainable architecture using a **self-hosted centralized web server** for processing, classification, storage, and alerting. Our microcontroller acts only as a data acquisition unit (accelerometer), sending readings to a centralized processor. This clean separation of roles makes updates, maintenance, and security far more manageable.

In fact, one of the reasons we avoided the Bassetti-style model is that it relies on letting each edge device act as both sensor and server. As shown in Figure 2.13, this "fully local" architecture introduces several key challenges:

- **Scalability Issues:** Edge nodes like ESP8266 and Raspberry Pi can only handle a limited number of connections, making public or institutional deployment difficult.
- **Update Difficulty:** Updating server logic requires physical access or complex automation to update firmware or containerized environments on each node.
- **No Data Redundancy or Central Oversight:** There's no unified dashboard, backup, or long-term analytics capability.

Thus, while Bassetti's system offers high fault tolerance and decentralization, it lacks centralized control, scalable infrastructure, and rich feature sets like magnitude estimation or false alarm classification, which are integral to our solution.

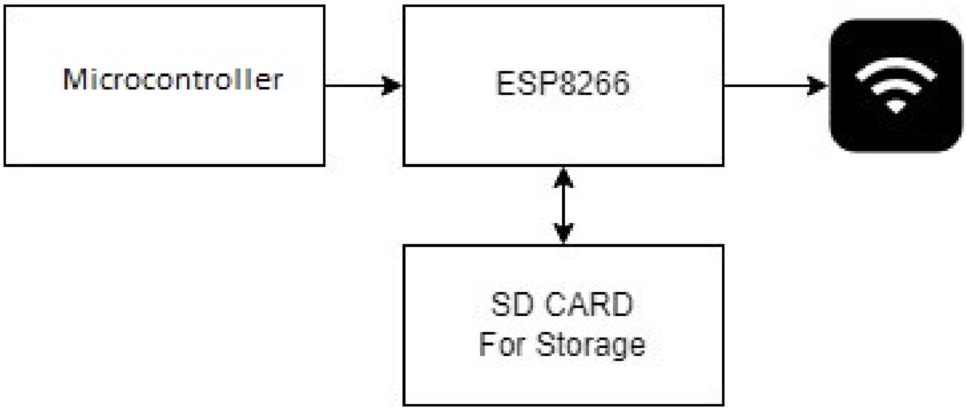

Figure 2.13: Bassetti et al. using ESP8266 for the webserver

2.3 Summary

The evolution of earthquake detection systems has progressed significantly, transitioning from traditional mechanical seismographs to modern IoT-based and mobile-integrated solutions. Early earthquake monitoring relied on **stationary seismographs**, which provided essential but **delayed information** about seismic events. The introduction of **digital computing and MEMS accelerometers** revolutionized the field, enabling real-time seismic data acquisition and efficient early warning capabilities.

Through our survey of existing earthquake detection systems, several approaches were identified:

- **IoT-Based Systems:** Platforms like the Mugla Journal System [10] and Perwej et al.'s IoT Seismic Monitoring System [12] employ networked sensors for **real-time data collection and cloud-based processing**.
- **Threshold-Based Detection:** Solutions like the Lovely Professional University Earthquake Alert System [11] rely on **Richter magnitude thresholds**, making it difficult to detect early, low-intensity P-waves. This limits their ability to provide **early warnings before strong shaking begins**.
- **GSM and Wireless-Based Detection:** Systems such as Ben et al.'s GSM-Based Earthquake Detection [14] focus on **data transmission via cellular networks**, ensuring real-time monitoring but **lacking pre-earthquake alerts**.
- **Machine Learning-Based Classification:** The ANN model developed by Khan et al. [16] introduces **feature extraction and artificial intelligence** for earthquake classification. However, this system does not predict earthquakes but rather **distinguishes between seismic events and background noise**.
- **Mobile-Based Warning Systems:** Projects like the Early Warning System in Mobile-Based Impacted Areas [15] leverage **smartphone notifications, GPS mapping, and cloud messaging** to **quickly alert users** and guide them toward safety.

- **Edge-Based Decentralized Detection:** The system by Bassetti and Panizzi [21] uses a **fully decentralized mesh network of detectors** that locally process seismic data using a CRNN model. Their edge-computing approach enhances **privacy, fault tolerance, and latency** but does not estimate magnitude and lacks centralized coordination or redundancy.

Comparison to Our Proposed System: While existing solutions provide valuable insights, many suffer from **high energy consumption, infrastructure dependency, or delayed warnings** due to reliance on post-event magnitude detection or limited signal interpretation. In contrast, **our system integrates a Random Forest classifier, magnitude estimation, and real-time filtering to provide an efficient, cost-effective, and power-saving early warning solution**. By leveraging **STM32 microcontrollers, Wi-Fi communication, and a robust detection algorithm**, **our system offers an innovative approach to accurate, rapid, and scalable** earthquake warnings.

Future Directions: Advancements in earthquake detection will likely continue to integrate **machine learning, edge computing, and decentralized IoT sensor networks**, further improving the **accuracy, scalability, and responsiveness** of early warning systems. Our project aims to contribute to this evolution by offering an efficient, adaptable, and cost-effective alternative.

Table 2.2: Comparison of Earthquake Detection Projects

Features	**Our Project**	**Mugla Journal IoT-Based Earthquake Warning System [10]**	**Lovely Professional University Earthquake Alert System [11]**	**Perwej et al. IoT-Based Seismic Activity Monitoring [12]**	**Ben et al. Earthquake Detection Using GSM [14]**	**ANN-based Earthquake Detection [16]**	**Bassetti and Panizzi IoT Edge Network [21]**
Sensors Used	Accelerometer	Accelerometer & Gyroscope	Accelerometer	Accelerometer & Gyroscope	Accelerometer & Gyroscope	Accelerometer	Accelerometer (in probe)
Data Transmission Method	Wi-Fi	Wi-Fi	Wi-Fi	Bluetooth	Wi-Fi	Wi-Fi/Bluetooth	Wi-Fi (probe) to local Raspberry Pi
Microcontroller	STM32 [22] (for low-level control and optimization)	ATmega2560	ATmega328p	Atmega16	Arduino	32-bit single-core ARMv6 processor	ESP8266 (probe) & Raspberry Pi (detector)
Early Earthquake Warning	Yes	Yes	No	Yes	No	Yes	Yes
AI Model Used	Random Forest	None	None	None	None	Artificial Neural Network (ANN)	Convolutional Recurrent Neural Network (CRNN)

Chapter 3

Project Background

3.1 Accelerometer results unit

The accelerometer measures acceleration in units of g force (g) where 1 g is equal to $9.81m/s^2$, the acceleration caused by Earth's gravity. It provides acceleration values along three axes: **x, y, and z.**

The total acceleration or magnitude can be calculated using the following equation:

$$\text{let } g = 9.81m/s^2$$

$$\text{Total Acceleration Magnitude } (m/s^2) = \sqrt{(x \cdot g)^2 + (y \cdot g)^2 + (z \cdot g)^2}$$

Where **x, y** and **z** are the acceleration values along the respective axes.

To see how acceleration magnitude is done coding wise, you can check algorithm 7 in the sample code appendix A.

Since the Richter magnitude scale measures ground displacement rather than acceleration, a logarithmic conversion is required to translate the accelerometer's acceleration data into a more understandable measure of earthquake size. This conversion involves comparing the total acceleration magnitude to a reference amplitude, which aids in estimating the energy released during the earthquake.

3.2 Acceleration to Displacement conversion

3.2.1 Conversion Overview

Every 0.1 seconds, the microcontroller will retrieve the raw data from the accelerometer for the x, y, and z axes. This data will then be converted into magnitude. After 1 second, a total of 10 magnitudes will be obtained. During this 1-second interval, each of these magnitudes will be integrated to convert them into displacement.
First step: Integrating accelerations to get velocity.

$$v(t) = \int a(t)dt$$

Second step: Integrating velocities to get displacement.

$$d(t) = \int v(t)dt$$

Note: Here, *t* is fixed to 0.1s as the operation will occur every 0.1s.

3.2.2 Integration Calculations

Here, we will use the **trapezoidal rule** [23] for its simplicity and better use of the computing power:

$$S = \frac{\Delta t}{2} \cdot (f(t_1) + f(t_2))$$

Where
S: Area under the curve (integral).
Δt: Time step between samples.
f(t): Function (acceleration or velocity in this case).

Using the trapezoidal rule, the acceleration is integrated over time to obtain the velocity. The trapezoidal rule approximates the area under the curve of a function

First Integration: Acceleration to Velocity

$$v(t) = v_{prev} + \frac{\Delta t}{2} \cdot (a_{prev} + a_{current})$$

where

$v(t)$: Current velocity

v_{prev}: Previous velocity

Δt: Time interval between samples $(0.1s)$

a_{prev} and $a_{current}$: Previous and current acceleration values.

Second Integration: Velocity to Displacement

$$d(t) = d_{prev} + \frac{\Delta t}{2} \cdot (v_{prev} + v_{current})$$

where

$d(t)$: Current displacement

d_{prev}: Previous displacement

Δt: Time interval between samples $(0.1s)$

v_{prev} and $v_{current}$: Previous and current velocity values.

3.2.3 Sampling

To improve accuracy, we can reduce the time between callback interrupts to 100ms or less, and adjust the value of the dt variable accordingly.

For example:

- $dt = 0.05$ **and timer callback = 0.05**s
  This will give us:
  $$\frac{1}{0.05} = 20 \text{ samples per second}$$
- $dt = 0.01$ **and timer callback = 0.01**s
  This will give us:
  $$\frac{1}{0.01} = 100 \text{ samples per second}$$
- $dt = 0.001$ **and timer callback = 0.001**s
  This will give us:
  $$\frac{1}{0.001} = 1000 \text{ samples per second}$$

That's the maximum number of samples we can get in 1 second because our accelerometer (MPU-6050) output data rate has a maximum frequency of 1000 Hz [24].

However to keep the station in lower power consumption mode we will remain using $dt = 0.1$ and timer callback = 0.1s until it's proven to have a noticeable issues with accuracy.

3.2.4 Richter Conversion

After accelration is converted to displacement, the next step is evaluating Richter′s magnitude using the following equation [25]:

$$M_L = log_{10}(A) - log_{10}(A_0)$$

Where

A_0: the Reference amplitude depending on the distance of the station from the earthquake's epicenter.
A: Amplitude of the seismic wave (D_{max} in this case).

To calculate A we would just take the amplitude of seismic waves, while A_0 we would need to calculate the distance to the epicenter then convert it to A_0 [25] so to calculate the distance to the epicenter we would use this equation [26]:

$$ED = 8.4(S - P)$$

Where
ED: Epicentral distance in kilometers
S: Arrival time of the first S-wave in seconds.
P: Arrival time of the first P-wave in seconds.

As you see in the equation, we need to find how to get the P-wave and S-wave arrival time, as mentioned before our station only works when it detects a vibration, so the arrival time for the P-wave would always be at 0 seconds. For the S-wave arrival time we will use STA/LTA ratio which is a measurement used in signal processing and seismology to detect sudden changes in a signal. The STA/LTA ratio highlights the moments when there is a sudden increase in signal energy, making it easier to identify wave arrivals [27].

To calculate the STA we would use this equation [28]:

$$\text{STA}(t) = \frac{1}{N} \sum_{i=t-N+1}^{t} |x(i)|$$

Where

N: Number of samples for the short-term window.

x(i): Signal amplitude at time i.

Then to calculate the LTA we would use this equation [28]:

$$\text{LTA}(t) = \frac{1}{M} \sum_{i=t-M+1}^{t} |x(i)|$$

Where

M: Number of samples for the long-term window.

Then to calculate the STA/LTA ratio we simply divide the STA with LTA [28]:

$$\text{STA/LTA Ratio}(t) = \frac{\text{STA}(t)}{\text{LTA}(t)}$$

The STA/LTA ratio is monitored in real time. When the ratio surpasses a predefined threshold, it is used to mark the estimated arrival of the S-wave. Otherwise, the system continues to monitor until this condition is met.

LTA is rarely ever 0, however if LTA is 0, then to avoid errors from occurring we would simply make the STA/LTA ratio 0.

After calculating the Epicentral distance, we then can convert it into $-log_{10}(A_0)$ using the following graph:

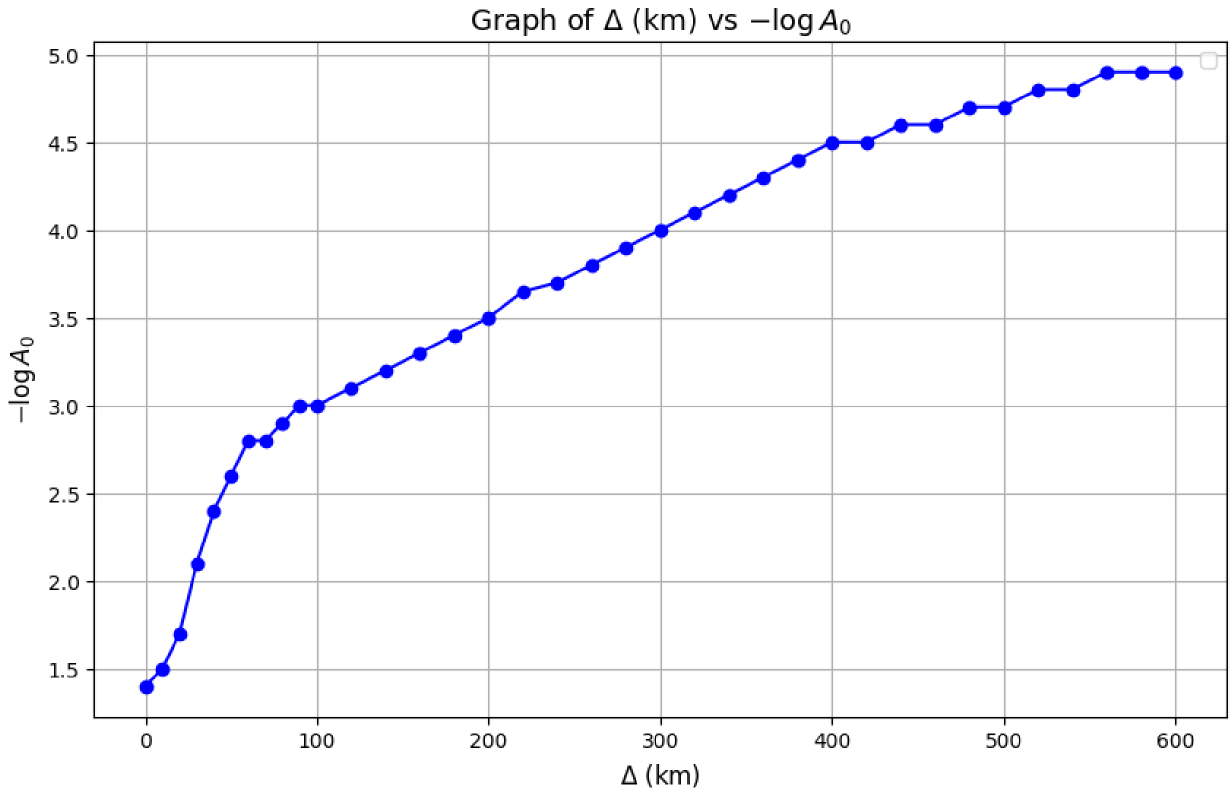

Figure 3.1: Graph Between Δ km and A_0

Which follows this equation:-

$$y = -1.6077510^{-10}\,x^4 + 2.142910^{-7}\,x^3 - 0.000100878\,x^2 + 0.023678\,x + 1.45704$$

where

x: Is the epicentral distance.

y: Is the -log(A_0).

To see how all of this is used in coding to calculate the richter go to algorithm 5 in the sample code appendix A.

3.3 Earthquake Frequency

Seismic waves generated by earthquakes cover a broad range of frequencies, which are crucial in understanding how earthquakes affect structures and are detected by seismic instruments. The frequency content of an earthquake primarily depends on the magnitude, depth, and geological conditions of the region.

3.3.1 Classification of Earthquake Frequencies

Seismic waves can be classified based on their frequency ranges:

- **High-Frequency Waves (1 - 20 Hz):** These waves are associated with small to moderate earthquakes and typically have shorter wavelengths. They are more destructive to smaller structures such as houses and bridges [29].
- **Low-Frequency Waves (0.001 - 0.1 Hz):** Generated by large earthquakes, these waves have long wavelengths and can travel significant distances. They primarily affect tall buildings, bridges, and large infrastructure projects [30].

3.3.2 Frequency of P-Waves and S-Waves

P-waves (primary waves) and S-waves (secondary waves) have different frequency characteristics:

- **P-Waves:** These are the fastest seismic waves and typically have higher peak frequencies (1–20 Hz) [31]. Since they arrive first and are minimally destructive, they are useful for early warning systems.
- **S-Waves:** These waves travel slower than P-waves and generally have lower peak frequencies (0.01–5 Hz). They cause stronger ground shaking and are responsible for most of the damage in an earthquake.

3.3.3 Impact of Frequency on Earthquake Detection

The frequency of seismic waves plays a critical role in earthquake detection and monitoring. High-frequency waves attenuate faster, limiting their detection range, while low-frequency waves travel longer distances with less energy loss. This distinction is important in designing sensor networks:

- Low-frequency waves are more detectable by broadband seismometers, which are used for monitoring large-scale seismic events.
- High-frequency waves are crucial for detecting near-field earthquakes and providing rapid alerts.
- Filtering techniques, such as band-pass filters, are applied to distinguish seismic signals from environmental noise.

3.3.4 Relevance to This Project

In this project, our earthquake detection system utilizes frequency thresholds to distinguish between normal ground vibrations possibly caused by a car or a truck driving by for example, and potential seismic events by analyzing the dominant frequencies in acceleration data.
Understanding earthquake frequency characteristics is essential for optimizing detection algorithms and ensuring reliable real-time monitoring, so we need to find a way to turn our acceleration data into frequency.

3.4 Acceleration to Frequency

We need a way to be able to differentiate between if something is an earthquake or for example if its a car driving by, and to differentiate between those we decided to use frequency, and in order to get the frequency we would use fourier transform.

3.4.1 Fourier Transform

Fourier transform is a mathematical operation that converts a given function into another, revealing how much of various frequency components are present in the original function [32].

Fourier transform is done using the following equation:-

$$\mathcal{F}(f(t)) = F(w) = \int_{-\infty}^{\infty} f(t)e^{-jwt}\,dt$$

- **f(t):** Original function in the time domain. (in our case the acceleration or velocity function)
- **F(w):** Resulting function in the frequency domain.
- **w:** Angular frequency.

To determine the dominant frequency, we take the acceleration function, apply the Fourier transform, and extract the peak frequency component, which represents the most significant oscillatory behavior in the data. However, computing the Fourier transform directly can be computationally expensive, especially for real-time earthquake detection and it would take more time therefore we'll use something called fast fourier transform.

3.4.2 Fast Fourier Transform (FFT)

To improve computational efficiency, the FFT algorithm is employed. FFT is an optimized implementation of the Discrete Fourier Transform (DFT) that reduces computational complexity from $O(N^2)$ to $O(N \log N)$ [33], making real-time processing feasible.

FFT operates using a **divide-and-conquer** approach, recursively breaking down the input signal into smaller components. It exploits symmetries in the Fourier transform, particularly the periodicity and redundancy of complex exponentials, to significantly reduce computation time.

In the earthquake detection system, FFT is used to analyze acceleration signals from the MPU-6050 sensor. By transforming the time-domain acceleration data into the frequency domain, the system can:

- **Extract dominant seismic frequencies**.
- **Differentiate between normal vibrations and earthquake events** based on spectral characteristics.
- **Enhance real-time processing efficiency**, ensuring fast response times for early warning capabilities.

This approach allows the system to achieve accurate and reliable seismic event detection while maintaining computational efficiency, making it suitable for real-time earthquake monitoring applications, to know more about how it was done check algorithm 2 in the sample code appendix A.

3.4.3 Example of Acceleration to Frequency

To showcase an example we'll use an actual earthquake data, we'll use data from Kobe 1995-01-16 earthquake from the strong-motion virtual data center [34], Firstly we'll use the acceleration graph

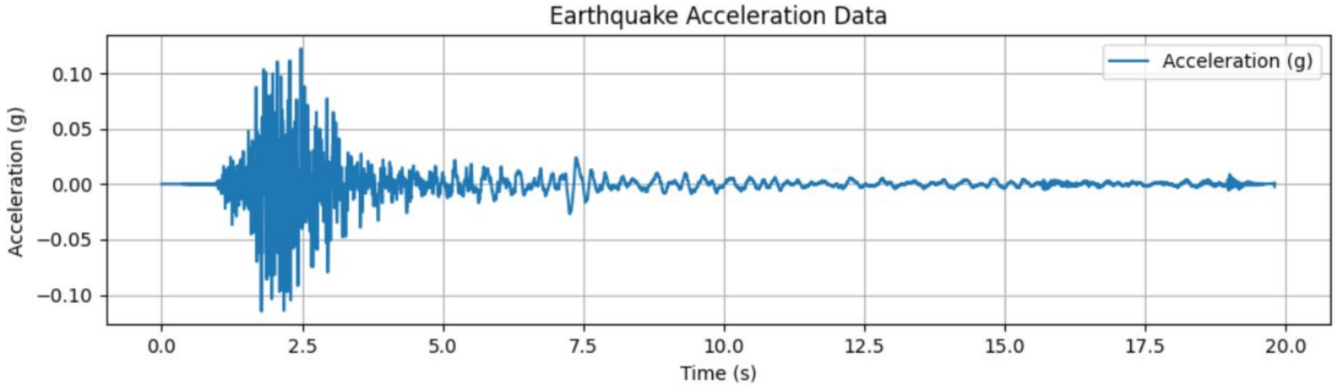

Figure 3.2: Kobe Earthquake Acceleration Graph

Next, we take the acceleration graph and apply FFT on it, turning the graph into frequency just as seen in figure 3.3.

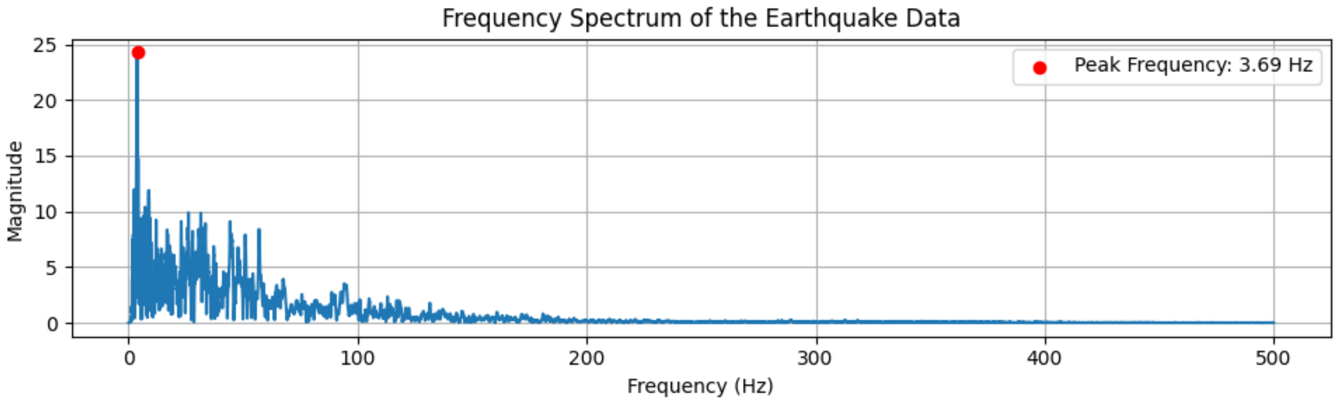

Figure 3.3: Kobe Earthquake Frequency Graph (After Fourier Transform)

We could use either acceleration or velocity or displacement graph to get the frequency however we chose to use the acceleration because to get the velocity or displacement calculations would need to take place but with acceleration we get it right away from the accelerometer readings so using the acceleration would be faster in determining whether its an earthquake or not. Now it's needed to determine the earthquake frequency, as you notice in the graph seen in figure 3.3 we have highlighted the peak frequency, that's because we will use that as the earthquake frequency so for this earthquake it would have a frequency of 3.69 Hz, as you may have noticed the frequency fits earthquakes since they can have frequency ranging from 1 - 20Hz [29] so to differentiate between an earthquake and other kinds of vibrations we would conclude that any earthquake with a frequency of between 1-20 Hz would be considered as a possible earthquake.

3.5 Server Side

A critical component of this project is the server side, which serves as the central system for receiving, processing, storing, and displaying data collected from the earthquake detection stations. The server not only acts as a reliable data aggregator but also ensures real-time visualization and remote accessibility. This backend infrastructure must be both modular and platform-agnostic to allow easy deployment and maintainability.

To meet these requirements, we used **Docker** as our primary deployment strategy. By containerizing each service (database, backend, frontend), we can guarantee consistent behavior regardless of the underlying host environment be it Windows, Linux, physical hardware, or a cloud instance. All that's required is Docker, thereby eliminating dependency management concerns.

3.5.1 Physical Server

As shown earlier in **Figure 4.2**, our physical server architecture deliberately keeps one key component the **Python data handler script** outside the Docker environment. This script is crucial, as it acts as a mediator between the ESP-01 Wi-Fi module (which transmits raw sensor data) and the Dockerized services. Keeping it outside the container provides direct access to local ports and allows for low-latency, reliable communication with hardware interfaces.

The Python script then relays the cleaned and structured data to the backend service inside a container. The backend connects to a **MySQL** container that stores earthquake event data in a persistent volume. Docker volumes ensure data durability even when containers are restarted or updated.

Although combining the backend, frontend, and database into a single image could simplify deployment, we intentionally chose to implement a **microservices architecture**. This modular design allows each component to be independently updated, tested, scaled, or replaced enhancing the system's long-term maintainability and flexibility.

To make the physical server accessible over the internet, especially for remote monitoring, we use router-level features such as the *Virtual Server* (port forwarding) available in TP-Link routers. This provides external access to our local server while maintaining sufficient control over network security.

3.5.2 Cloud Server

In the cloud-hosted version of the server, depicted in **Figure 3.4**, we redesign the communication pipeline to eliminate the need for a local Python handler. Instead, the ESP8266 microcontroller sends data directly to a public HTTP endpoint, typically hosted on a cloud-based virtual machine or serverless backend.

This transition is made possible because the **ESP8266** supports both Wi-Fi connectivity and HTTP communication, enabling it to send real-time data to a remote database via RESTful API calls.

The ESP8266 was selected for the following reasons:

- **Wi-Fi support:** Easily connects to existing networks without extra modules.
- **Built-in HTTP request capability:** Can post data directly to a backend.
- **Cost-effectiveness:** Offers a great balance between features and price.
- **Optimal performance:** More efficient than the minimal ESP-01 and more appropriate (cost-wise) than the ESP-32 for this specific task.

This approach significantly reduces the server's dependency on local hardware, improving scalability and enabling seamless deployment across multiple locations.

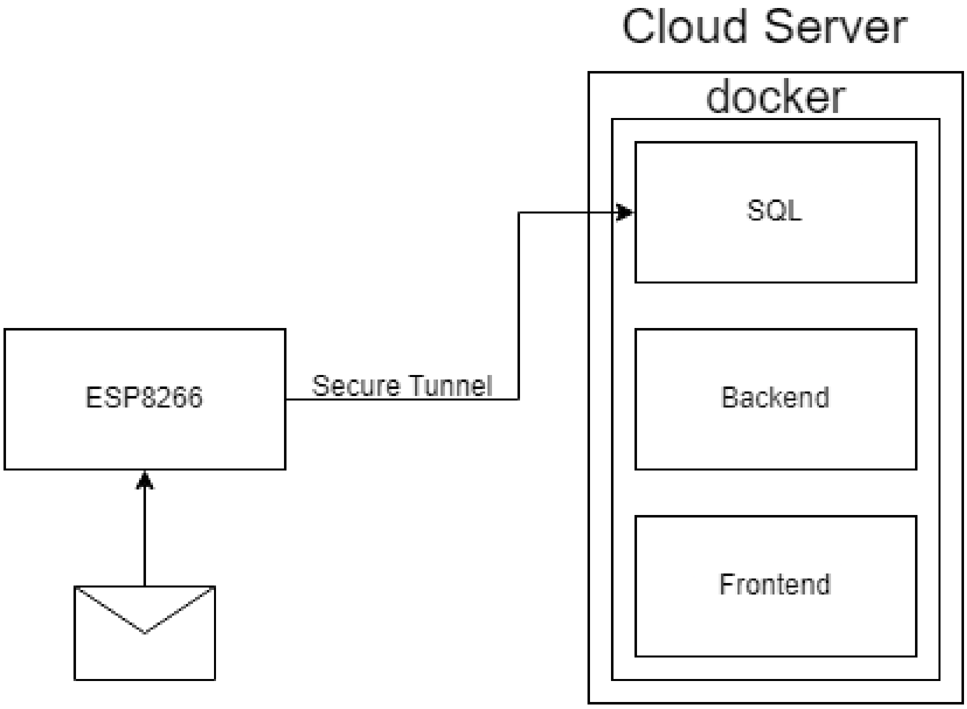

Figure 3.4: Cloud Server Architecture

3.5.3 Comparison

Table 3.1: Comparison of Physical and Cloud Servers

Criteria	Physical Server	Cloud Server
Control	Full hardware and software control	Managed environment with limited hardware access
Access to Physical Ports	Direct access allows easy integration with microcontrollers	No direct hardware access; only remote API communication
Scalability	Suited for single-location setups	Can handle multiple devices across regions
Maintenance	Manual updates and monitoring	Automated by cloud provider (backups, updates, monitoring)
Reliability	Dependent on local uptime and power stability	High availability via redundant infrastructure
Remote Access	Requires extra configuration (e.g., port forwarding)	Accessible globally with public DNS/IP
Operational Costs	One-time hardware purchase	Pay-as-you-go or subscription model
Internet Dependency	Can function offline temporarily	Must be online to send/receive data
Setup Complexity	Simpler to understand and debug locally	Requires familiarity with cloud tools and services
Security	Controlled locally; less exposed but depends on user	Requires cloud security hardening (firewalls, IAM, etc.)

3.6 False Alarm Detection

To rule out if a seismic event is an earthquake or not, we would follow two main steps, first we check the peak frequency and see if it fits in the thresholds, if it doesn't then we automatically rule it out as not an earthquake, however if it does fit that threshold then we send the acceleration data to an AI model which can determine if it's an earthquake or not.

3.6.1 Peak Frequency Check

We begin by analyzing the peak frequency of accelerations to identify potential earthquakes. If the peak frequency falls within the earthquake detection range of 1-20 Hz, we proceed with classification using various machine learning models. If the frequency does not fall within this range, the data is classified as a false alarm.

3.6.2 Machine Learning Models

We've tried multiple machine learning models to see which one would perform best, and each model will be tested with the first 200, 500 and 1000 acceleration data to see which would be the best, the ideal model should provide high accuracy at the least number of acceleration data since that means it can determine if it's a false alarm faster, since this system is mainly a warning system, its extremely important for the fastest warning possible but at the same time at as good of an accuracy as possible. For all the models we would do whats known as a 70/30 split where 70% of the data would be used for training and 30% for testing, we'll be using 100 earthquake signals and 100 noise signals (not earthquakes).

3.6.3 Naive Bayes Model

Naive Bayes is a probabilistic machine learning model based on Bayes' theorem, which assumes that features are independent of each other (hence "naive"). It works by calculating the posterior probability of each class (in this case, earthquake or not), given the features of the data. It then classifies the data into the class with the highest posterior probability [35].
Below are the results of using the Naive Bayes model for false alarm detection across datasets with 200, 500, and 1000 accelerations:

Table 3.2: Naive Bayes Model Results for False Alarm Detection

Accelerations	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)	False Positive Rate (%)	True Negative Rate (%)	False Negative Rate (%)
200	58.93	100.0	23.33	37.84	0.0	100.0	76.67
500	63.64	100.0	28.57	44.44	0.0	100.0	71.43
1000	65.45	100.0	29.63	45.71	0.0	100.0	70.37

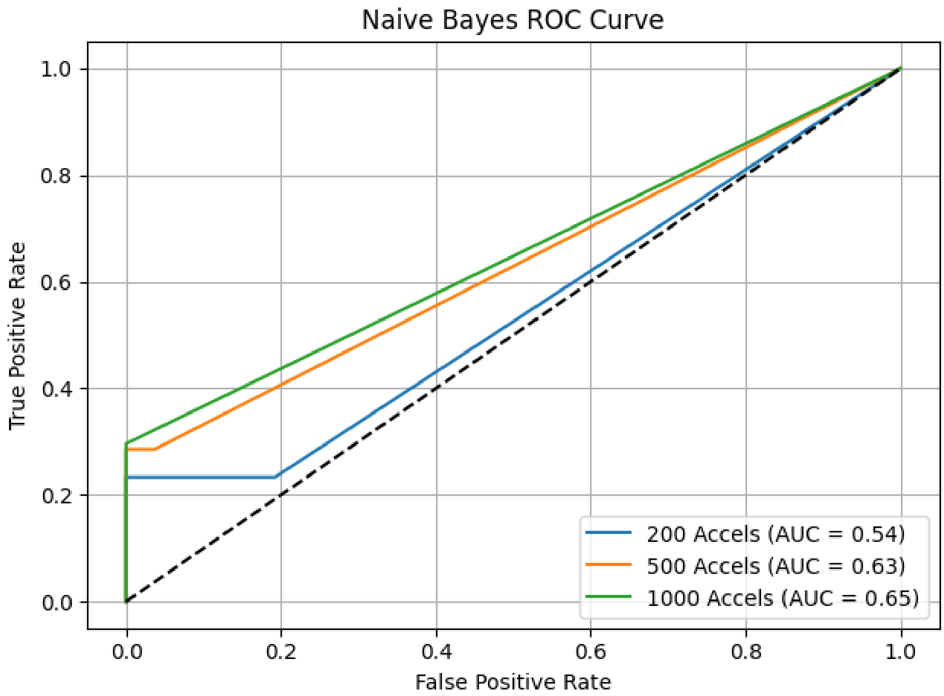

Figure 3.5: ROC Curve of Gradient Boosting Model

Comments on Naive Bayes Results:

- **200 Accelerations:**
  - **Accuracy:** 58.93% — Slightly better than random guessing, but too weak for deployment.
  - **Precision:** 100% — Every predicted earthquake was correct.
  - **Recall:** 23.33% — The model detected less than a quarter of actual earthquake events.
  - **F1 Score:** 37.84% — Reflects poor recall despite perfect precision.
  - **False Negative Rate:** 76.67% — A very high proportion of real earthquakes went undetected.
  - **ROC AUC:** 54.29% — Barely above chance level, suggesting weak discriminative power.
  - **Comment:** The model is extremely conservative, avoiding false alarms entirely but at the cost of missing most real earthquakes. Such behavior makes it ineffective for any early warning system.

- **500 Accelerations:**
  - **Accuracy:** 63.64% — Slight improvement due to increased input data.
  - **Precision:** 100% — All earthquake predictions were correct.
  - **Recall:** 28.57% — A small increase, but still poor.
  - **F1 Score:** 44.44% — Slight improvement, yet still unsatisfactory.
  - **False Negative Rate:** 71.43% — Most earthquakes still go undetected.
  - **ROC AUC:** 62.96% — Better than at 200 accelerations but still mediocre.
  - **Comment:** While it improves with more data, the model remains far too cautious. The extremely high threshold for earthquake classification limits its usefulness in real-time scenarios.
- **1000 Accelerations:**
  - **Accuracy:** 65.45% — Best of the three configurations, but still far from ideal.
  - **Precision:** 100% — Maintains flawless precision on earthquake predictions.
  - **Recall:** 29.63% — Only marginally better than at 500 accelerations.
  - **F1 Score:** 45.71% — Moderate improvement, but still reflects the underlying imbalance.
  - **False Negative Rate:** 70.37% — Nearly three out of four earthquakes are missed.
  - **ROC AUC:** 64.81% — The highest AUC among the three, but still insufficient.
  - **Comment:** Despite minor improvements, the model's conservative bias persists. It fails to provide the kind of sensitivity needed in early warning systems, making it inadequate for real-world deployment.

3.6.4 Gradient Boosting Model

Gradient Boosting is an ensemble learning technique where multiple weak learners (usually decision trees) are trained sequentially. Each tree attempts to correct the errors of the previous tree by focusing on the data points that were misclassified. Over time, the model improves its performance through this iterative correction process [36].
Below are the results for the Gradient Boosting model:

Table 3.3: Gradient Boosting Model Results for False Alarm Detection

Accelerations	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)	False Positive Rate (%)	True Negative Rate (%)	False Negative Rate (%)
200	85.42	81.48	91.67	86.27	20.83	79.17	8.33
500	87.50	84.00	91.30	87.50	16.00	84.00	8.70
1000	91.49	91.30	91.30	91.30	8.33	91.67	8.70

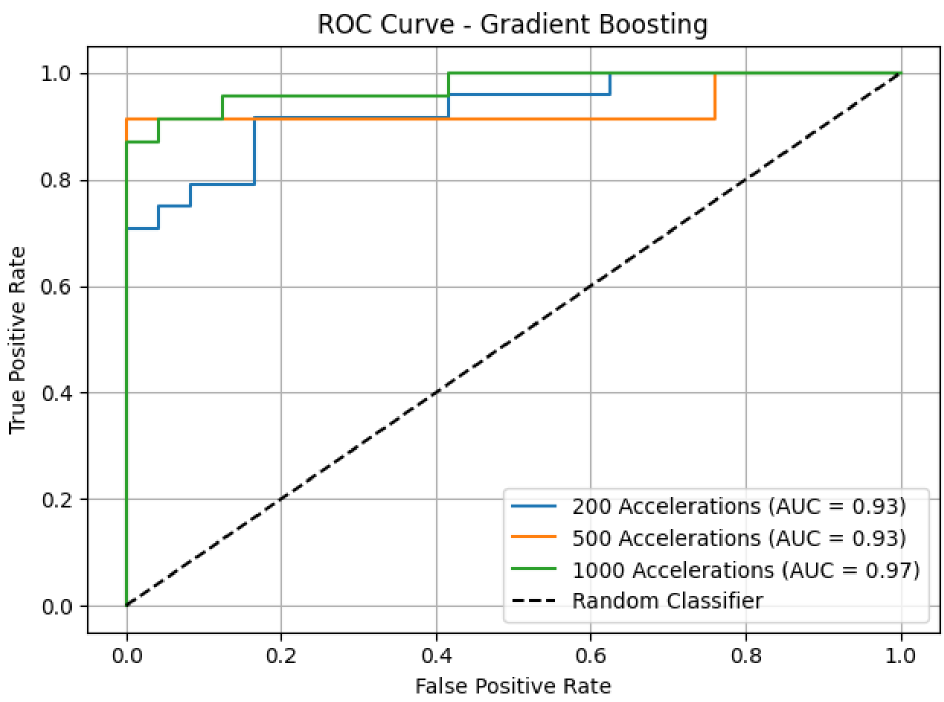

Figure 3.6: ROC Curve of Gradient Boosting Model

Comments on the Gradient Boosting Results:

- **200 Accelerations:**
  - **Accuracy:** 85.42% — Indicates good overall classification performance.
  - **Precision:** 81.48% — Most predicted earthquake events were correct, though some false positives remain.
  - **Recall:** 91.67% — The model successfully captured nearly all real earthquake events.
  - **F1 Score:** 86.27% — Demonstrates strong balance between false positives and missed detections.
  - **False Positive Rate:** 20.83% — False alarms are present but may be tolerable in early warning contexts.
  - **ROC AUC:** 93.06% — The model performs very well across all threshold levels, showing strong class separability even at low input sizes.
  - **Comment:** At just 200 acceleration points, the model performs well with excellent recall and a high AUC. However, its higher false positive rate may still require additional filtering to avoid unnecessary alarms.
- **500 Accelerations:**
  - **Accuracy:** 87.50% — Slight improvement, indicating more balanced predictions.

- **Precision:** 84.00% — Slightly better than at 200, meaning fewer false positives.
- **Recall:** 91.30% — Still excellent, maintaining strong detection of real earthquakes.
- **F1 Score:** 87.50% — Reflects a high-quality classification.
- **False Positive Rate:** 16.00% — Lower false alarms than before, making this configuration more favorable.
- **ROC AUC:** 93.39% — Slightly improved over the previous case, affirming the model's reliability across decision thresholds.
- **Comment:** The model performs better as input length increases, reducing false positives while preserving detection quality. The high AUC reinforces its reliability for critical systems.

• **1000 Accelerations:**
  - **Accuracy:** 91.49% — Highest accuracy among the three configurations.
  - **Precision & Recall:** 91.30% — A perfect balance, with very few missed events and low false positives.
  - **F1 Score:** 91.30% — Indicates overall robustness in prediction.
  - **False Positive Rate:** 8.33% — Very low, signaling minimal false alarms.
  - **ROC AUC:** 97.46% — Excellent ROC performance; the model offers nearly perfect discrimination between earthquake and non-earthquake cases.
  - **Comment:** This configuration yields the most consistent and reliable performance. The high AUC, combined with balanced precision and recall, make it the most dependable setting, especially for safety-critical applications where minimizing missed detections and false alarms is paramount.

3.6.5 LightGBM Model

LightGBM (Light Gradient Boosting Machine) is another variant of gradient boosting that is designed to be more efficient and scalable. It builds trees in a leaf-wise manner (rather than level-wise as in traditional gradient boosting), which often results in faster training (up to 20 times faster) and better performance on large datasets. It also uses histogram-based algorithms, which can speed up the process and handle large datasets efficiently [37].
Below are the results for the LightGBM model:

Table 3.4: LightGBM Model Results for False Alarm Detection

Accelerations	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)	False Positive Rate	True Negative Rate	False Negative Rate
200	81.25	75.86	91.67	83.02	29.17	70.83	8.33
500	93.75	95.45	91.30	93.33	4.00	96.00	8.70
1000	93.62	95.45	91.30	93.33	4.17	95.83	8.70

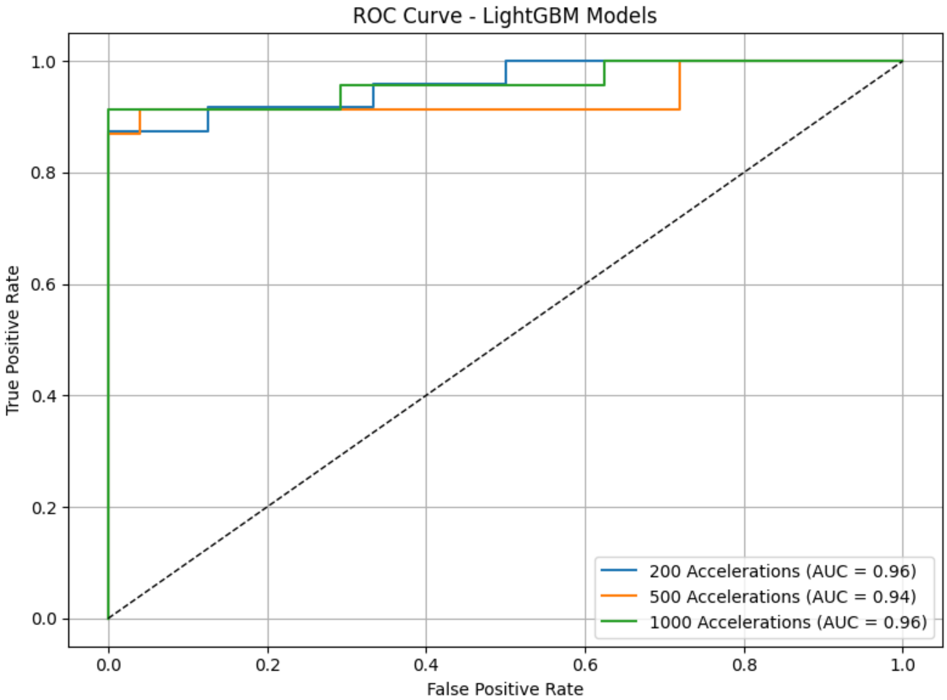

Figure 3.7: ROC Curve of LightGBM Model

Comment on LightGBM Results:

- **200 Accelerations:**
  - **Accuracy:** 81.25% — Decent performance, but less consistent than expected at lower input volume.
  - **Precision:** 75.86% — Indicates that one in four alarms was a false positive.
  - **Recall:** 91.67% — The model still captured most of the real earthquakes.
  - **F1 Score:** 83.02% — Reflects a modest balance between detecting real events and avoiding false alarms.
  - **False Positive Rate:** 29.17% — Almost one-third of the non-earthquake cases were falsely flagged.
  - **ROC AUC:** 96.01% — Despite imperfect predictions, the classifier's ability to separate classes is excellent.
  - **Comment:** This configuration shows promising separation power but a high false positive rate. It would require post-processing (e.g., threshold tuning) before deployment.

- **500 Accelerations:**
  - **Accuracy:** 93.75% — A substantial improvement over the 200-acceleration case.
  - **Precision:** 95.45% — Nearly all predicted earthquake events are correct.
  - **Recall:** 91.30% — Maintains strong detection of real earthquakes.
  - **F1 Score:** 93.33% — Indicates very strong overall classification performance.
  - **False Positive Rate:** 4.00% — Minimal false alarms enhance reliability.
  - **ROC AUC:** 93.57% — Confirms strong class separability and reliable scoring.
  - **Comment:** With low false alarms and high recall, this is a balanced and deployment-ready configuration. The AUC score reinforces that the model ranks earthquake likelihood effectively.
- **1000 Accelerations:**
  - **Accuracy:** 93.62% — Almost identical to the 500-acceleration result.
  - **Precision:** 95.45% — Excellent precision with minimal false alarms.
  - **Recall:** 91.30% — No loss in sensitivity compared to the 500-sample case.
  - **F1 Score:** 93.33% — Shows consistent performance even as input data increases.
  - **False Positive Rate:** 4.17% — Maintains low false alarms.
  - **ROC AUC:** 96.01% — Matches the 200-acceleration AUC, showing that the model maintains strong class separation even as the input grows.
  - **Comment:** This setup ensures reliable classification with great separation power. However, compared to 500-acceleration input, it doesn't offer significant gains, implying 500 might be the most efficient configuration.

3.6.6 Random Forest Model

Random Forest is an ensemble method that constructs multiple decision trees during training and outputs the mode of the classes predicted by individual trees. The model mitigates overfitting by averaging the results of many decision trees, each trained on a random subset of the data [38].
Finally, here are the results for the Random Forest model:

Table 3.5: Random Forest Model Results for False Alarm Detection

Accelerations	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)	False Positive Rate	True Negative Rate	False Negative Rate
200	94.64	96.55	93.33	94.92	3.85	96.15	6.67
500	96.36	100.0	92.86	96.30	0.00	100.0	7.14
1000	96.36	100.0	92.59	96.15	0.00	100.0	7.41

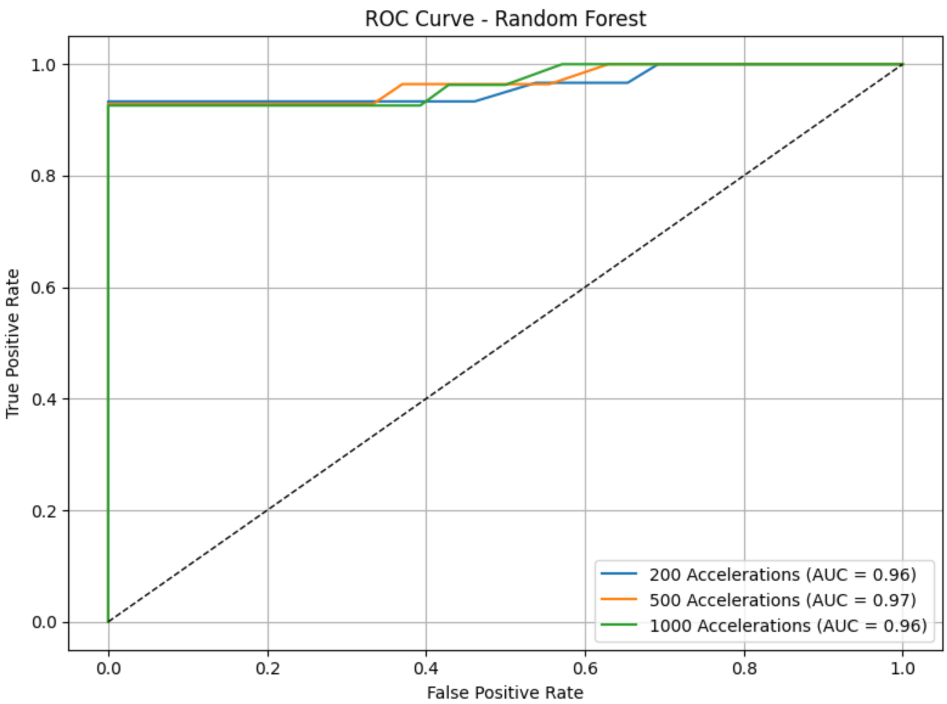

Figure 3.8: ROC Curve of Random Forest Model

Comments on Random Forest Results:

- **200 Accelerations:**
  - **Accuracy:** 94.64% — Strong and reliable performance from a small dataset.
  - **Precision:** 96.55% — Nearly all detected earthquakes were correct, showing very few false positives.
  - **Recall:** 93.33% — Most actual earthquakes were successfully identified.
  - **F1 Score:** 94.92% — A near-perfect balance between detection and false alarms.
  - **False Positive Rate:** 3.85% — Very few false alarms.
  - **ROC AUC:** 96.09% — Excellent class separation with high reliability even at lower input size.
  - **Comment:** The model is highly accurate and well-balanced even with only 200 data points, making it suitable for rapid deployment in real-time systems.
- **500 Accelerations:**
  - **Accuracy:** 96.36% — Excellent performance across the board.
  - **Precision:** 100.0% — All detected earthquakes were actual events — zero false positives.
  - **Recall:** 92.86% — Very few missed detections.
  - **F1 Score:** 96.30% — Near-perfect score balancing precision and recall.

- **False Positive Rate:** 0.00% — No false alarms occurred.
- **ROC AUC:** 96.63% — Outstanding separability and highly dependable decision thresholds.
- **Comment:** At 500 acceleration readings, Random Forest provides its strongest and most reliable output, making it the optimal configuration for this system. The combination of speed, accuracy, and ROC AUC make this setup ideal.

• **1000 Accelerations:**
  - **Accuracy:** 96.36% — Maintains same excellent performance as at 500 readings.
  - **Precision:** 100.0% — Perfect detection with no false earthquake alerts.
  - **Recall:** 92.59% — Slightly lower than 500 but still excellent.
  - **F1 Score:** 96.15% — High accuracy maintained with a stable prediction balance.
  - **False Positive Rate:** 0.00% — Zero false alarms again.
  - **ROC AUC:** 96.49% — Demonstrates consistent and highly effective class separation.
  - **Comment:** The performance is nearly identical to the 500-reading configuration, but requires more data to achieve the same result. Thus, the 500-point setup remains the best balance between speed and accuracy.

3.6.7 Which Model Would Be Chosen?

After comparing the performance of all tested models across different acceleration sample sizes, Random Forest emerges as the most suitable model for our earthquake false alarm detection system. While each model has its own strengths and weaknesses, Random Forest provided the best trade-off between classification accuracy and the amount of input data needed.

Naive Bayes showed very poor recall, particularly at smaller input sizes, and failed to detect the majority of real earthquakes despite perfect precision. This overly conservative behavior makes it unsuitable for our use case, where missing an actual earthquake is far more dangerous than raising a false alarm.

Gradient Boosting performed admirably, with strong recall and high AUC scores even at just 200 readings. However, it showed slightly more false positives than Random Forest and did not surpass it in overall accuracy, especially when both models were tested on the same sample size.

LightGBM also performed very well at higher acceleration inputs, achieving high accuracy and precision. However, its lower performance at 200 accelerations and only marginal gains after 500 made it less efficient than Random Forest in real-time scenarios.

Random Forest, by contrast, achieved a high accuracy of 94.64% at just **200 accelerations**, and rose to 96.36% at 500 readings matching its performance at 1000. This stability indicates that 500 readings are sufficient for optimal performance, but at 200 accelerations readings still maintains a good accuracy and can output a result at less than half the readings making it more efficient. It maintained excellent precision and recall, along with a low false positive rate. Furthermore, Random Forest's architecture allows for rapid training and inference, making it ideal for a system that must quickly determine whether a seismic event is a genuine earthquake or a false alarm.

So in conclusion, Based on its ability to achieve high accuracy quickly (with as few as 200 accelerations), its balanced performance across all key metrics, and its faster computation time compared to other models, **Random Forest** is the chosen model for our system.

3.7 Summary

This chapter outlines the foundational components and methodologies employed in the IoT-based earthquake detection system. It begins with the interpretation of accelerometer data, which is recorded in gravitational units and converted into total acceleration magnitude. Through a two-stage trapezoidal integration method, acceleration is translated into displacement, enabling the estimation of earthquake strength using the Richter magnitude scale. A key feature of the system is its ability to determine the epicentral distance using P-wave and S-wave arrival times, which further refines magnitude calculation.

The chapter also explores earthquake frequency characteristics and how Fast Fourier Transform (FFT) is used to extract dominant frequency components from seismic data. Low-pass filtering is applied to remove noise and enhance detection reliability. Additionally, the chapter discusses server-side design—both physical and cloud—focusing on data logging, visualization, and system scalability. Finally, several false alarm reduction strategies are introduced, including both frequency thresholding and machine learning models, with a Random Forest model emerging as a preferred classifier based on accuracy and balance.

These techniques collectively form the scientific and computational backbone of the system, establishing the basis for real-time, accurate earthquake detection.

Chapter 4

Proposed System Design

4.1 Industry Standard

The industry standards for earthquake detection systems integrate advanced technologies to ensure accurate and timely monitoring and alerts. Below are some key aspects of modern systems and technologies:

1. **Sensor Networks:** Systems like Japan's Earthquake Early Warning (EEW) [8], California's ShakeAlert [39] and the Canadian Earthquake Early Warning [40] use real-time data from sensor networks to provide warnings seconds before destructive seismic waves arrive, they utilizes dense arrays of high-sensitivity seismometers and MEMS sensors. MEMS sensors are compact, cost-effective, and widely deployed in various settings, including smartphones and IoT devices [4]. These systems detect the p-wave of an earthquake and once its detected it send the warning right away, then it continues measuring earthquake seismic data until its over to analyze the data [40].

2. **Frequency and Signal Filtering:** Advanced frequency analysis methods, such as Fourier Transforms, and filtering mechanisms like band-pass, these filters are used to differentiate between seismic activity and other vibrations, it can also be used to analyze the seismic activity data more easily [41].

3. **Satellite-Based Technologies:** Systems such as InSAR use satellite imagery to detect ground deformation, providing valuable insights into seismic activity [42].

4. **Communication Protocols:** IoT-based communication protocols improve the transmission of real-time data between distributed sensors and the system. These protocols are designed to ensure low latency and high reliability [43], crucial for effective warnings.

4.2 Requirement Analysis

To cover our system requirements, lets divide it into four sections. Hardware requirements, software requirements, functional requirements and non-functional requirements.

4.2.1 Hardware Requirements

The hardware requirements consists of 5 main parts which are the **sensors, microntroller, communication module, power supply**, and the **peripheral components.**

- **Sensors:** Firstly, we need sensors to be able to detect an earthquake. Generally, most projects use both an accelerometer and an gyroscope to detect an earthquake, however we want to save costs while not majorly affect the results so we chose to use just an **accelerometer** as our sensor, however there's three kinds of accelerometer which are **1-Axis accelerometer, 2-Axis accelerometer and 3-axis accelerometer** so lets cover each one briefly.
  - **1-Axis Accelerometer:** Measures acceleration along a single axis, however since its only one axis its accuracy won't be the best for our project, so we can reject this option.
  - **2-Axis Accelerometers:** Measures acceleration along two perpendicular axes (X and Y), while its more accurate than the 1-axis accelerometer. It still won't have enough accuracy for our system, so we can reject this option as well.
  - **3-axis Accelerometer:** Measures acceleration along three perpendicular axes (X, Y, and Z). These are the most common and versatile accelerometers, used in applications like smartphones, fitness trackers, and most importantly earthquake detection devices, and it has the best accuracy therefore we have decided this is the best type of accelerometer to use for our project

- **Microcontroller:** For our microcontroller we want a cheap microcontroller however we want it to have a good memory, low abstraction level and to be able to receive data from the sensors and send it, so our choices were between **ATmega32, Arduino** and **STM32** so lets go between each one of them
  - **ATmega32:** Out of these 3 microcontrollers, this is the cheapest microcontroller and it provides a low level abstraction, however its memory size is smaller than what we would need for this project so we can eliminate this choice.
  - **Arduino:** Its the most expensive option and it provides a high-level abstraction so we can elimnate this choice too.
  - **STM32:** Eventhough its a little more expensive than ATMEGA32 but its still relatively cheap, it also provides a low-level abstraction, it has a good memory size and low power consumption [44] so it fits all the requirements of our project.
- **Communication Modules:** Our project consists of two sides, the station side and the server side so for these two sides to communicate we need a communication module that would be capable of reliably communicating almost anywhere, we have 3 have choosen options as a way for data transmission which are **Bluetooth, Wi-Fi** and **Radio waves**.
  - **Bluetooth:** A Bluetooth module can provide reliable data transmission however bluetooth only works within very short distances of couple meters, when we need communication over a long so we can remove bluetooth as an option.
  - **Radio waves:** To communicate with radio waves we would use LoRa which provides reliable data transmission and can transmit over long distances up to 10 km and radio waves could be used anywhere making the system very remote, however LoRa has a low data rate of 0.3-27Kbps [45] and radio waves have an issue with multipath propagation which would force us to place the station in an empty area with no obstacles or buildings otherwise it would cause delays and reflected signals arriving at different times could confuse the system [46], and another major issue with LoRa is its legallity where its illegal in some countries like Egypt [47], therefore we chose not to choose it.
  - **Wi-Fi:** A Wi-Fi module can provide reliable and fast data transmission (2.7 Mbps [48]) and it can be used anywhere as long as there is a WiFi connection making it very remote, therefore we decided to choose it for our project

- **Power Supply:-**
  - **Li-ion Batteries:** Provides the power source for the station.
  - **Battery Case:** Simplifies battery connection and replacement.
- **Peripheral Components:-**
  - **STM32 Programmer:** To upload and burn code to the STM32 microcontroller.
  - **Wires and Breadboard:** Used for prototyping and circuit connections (Breadboard will be used in the testing phase then later it will be placed onto a PCB board).

4.2.2 Software Requirements

The software requirements consists of 4 main parts which are **development environments**, **Programming Languages**, **Server Management** and **visualization and monitoring**

- **Development Environments:**We needed a development environment with a low level of abstraction, minimal configuration, and compatibility with STM32. To identify the most suitable platform for our project, we assessed three different Integrated Development Environments (IDEs). The options we reviewed:
  1. **Arduino IDE:** Eventhough its easy to use, arduino IDE has a high level abstraction [49] therefore we did not select this IDE. This restriction prevents us from accessing and modifying key components of the underlying code and hardware interactions, which are critical for gaining insights and optimizing the system.
  2. **Eclipse IDE:** It is a low level abstraction however it has difficult configurations and doesn't support STM32 without the usage of a third party software therefore we did not select this.
  3. **STM32CubeIDE:** This IDE offers low-level abstraction along with advanced configuration options and specialized tools designed for the STM32 microcontroller. It enhances hardware configuration, debugging, and integration with various STM32 libraries, streamlining the development process. As a result, we have chosen this IDE for our project.
- **Programming Languages:-**
  - **C:** Primary language for microcontroller programming.
  - **Python:** For server-side development and communication handling.
  - **React ,Tailwind, and CSS:** For web development.

- **Server Management:-**
  - **Docker:** To containerize server applications for deployment across various environments.
  - **MySQL or SQLite:** For data storage and retrieval.
- **Visualization and Monitoring:-**
  - **Web Interface:** A browser-based application for real-time monitoring and data visualization.

4.2.3 Functional Requirements

- The system **shall continuously monitor** ground vibrations using the 3-axis accelerometer.
- The system **shall process** raw sensor data to detect seismic events by calculating acceleration, velocity, and displacement.
- Upon detection of significant seismic activity, the system **shall trigger** the microcontroller to activate and process the event.
- The system **shall transmit** seismic data via the Wi-Fi module to a central server in real time.
- The system **shall calculate** the earthquake magnitude (Richter scale) from the measured displacement.
- The system **shall log** all seismic events, including time stamps, sensor readings, and calculated magnitudes, in a database.
- The system **shall provide** a web-based interface that displays real-time and historical seismic data for monitoring and analysis.
- The system **shall allow** remote configuration of sensor thresholds and calibration parameters.

4.2.4 Non-Functional Requirements

- **Performance:**
  - The system **must process** sensor data and generate alerts within one second of detecting an event.
  - The data transmission latency over Wi-Fi should be minimal to support real-time monitoring.

- **Reliability:**
  - The system **must accurately** detect seismic events with a false alarm rate below a specified threshold.
  - The hardware and software components should have high mean time between failures (MTBF) to ensure continuous operation.
- **Scalability:**
  - The system **must support** multiple sensor nodes and allow for expansion without significant modifications.
  - The server architecture must be scalable to handle increased data volumes.
- **Energy Efficiency:**
  - The system **must operate** in an energy-efficient manner, activating only when significant vibrations are detected.
  - The power consumption of the hardware components should be minimized to extend battery life.
- **Usability:**
  - The web interface **should be** intuitive and user-friendly, allowing non-technical users to easily monitor seismic events.
  - Configuration settings should be accessible and clearly documented.
- **Maintainability:**
  - The system **should be designed** in a modular fashion so that individual components (hardware or software) can be updated or replaced with minimal impact on the overall system.
  - The code and documentation **must be** clear and well-organized to facilitate future maintenance.
- **Portability:**
  - The system **should be** deployable in remote or off-grid locations, requiring minimal infrastructure.

4.2.5 Proposed System Block Diagrams

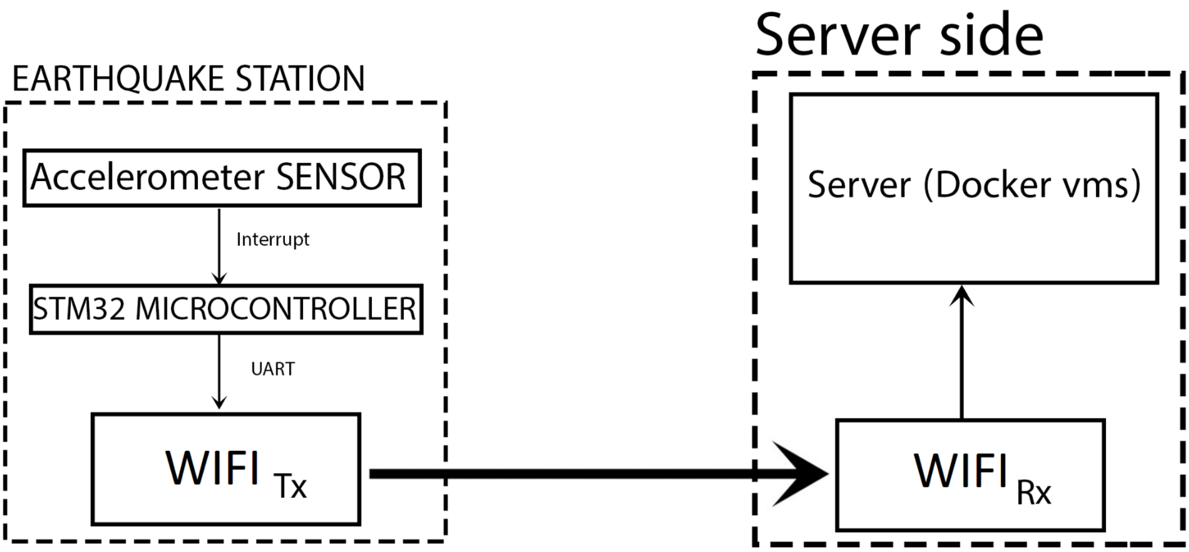

Figure 4.1: General Block Diagram of our System

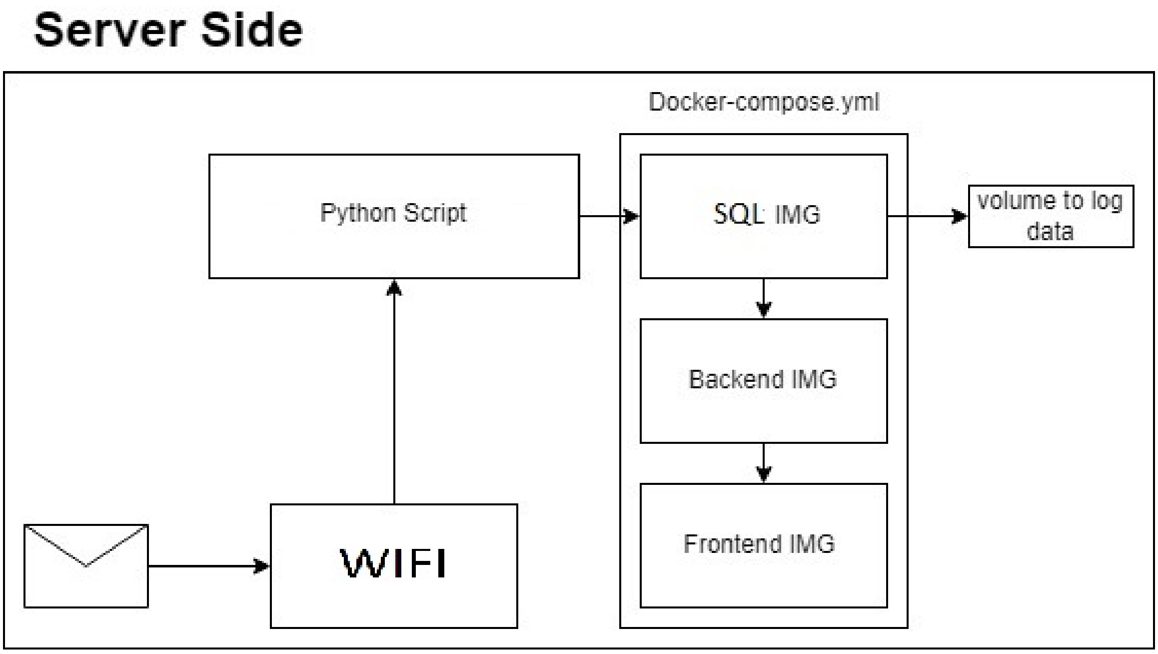

Figure 4.2: More Detailed Block Diagram for Server Side

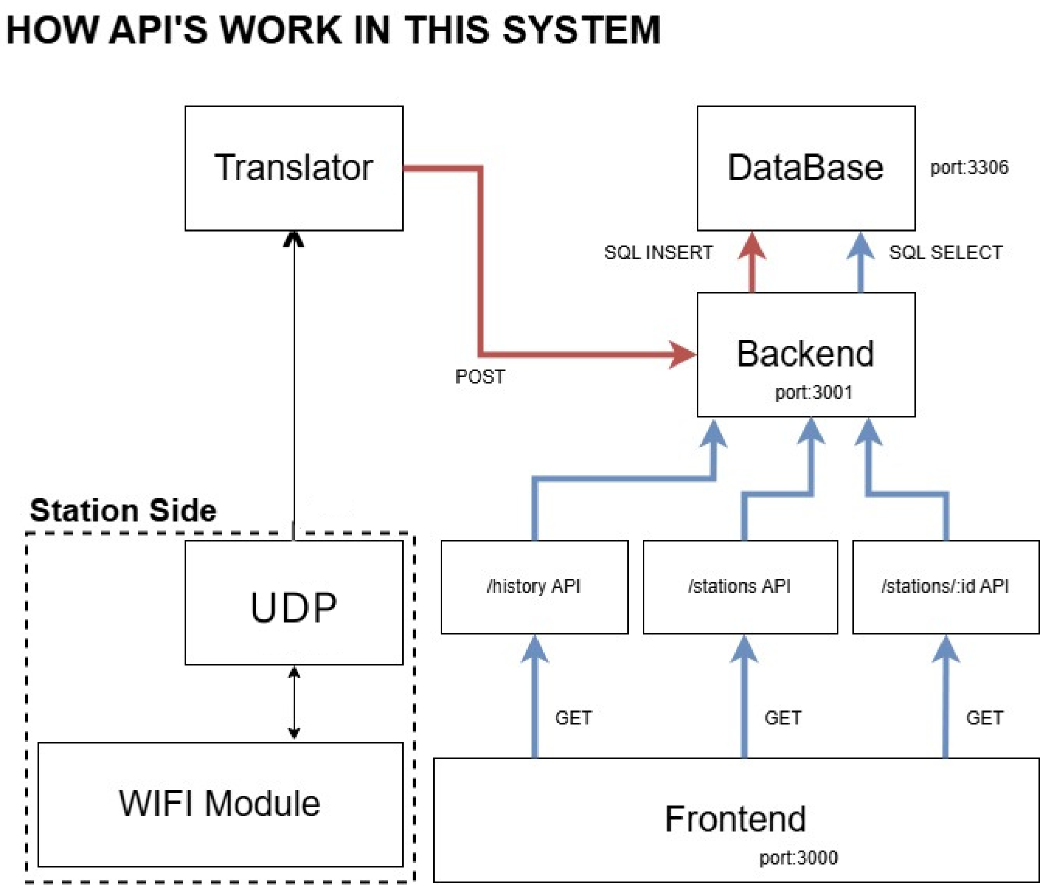

Figure 4.3: Block Diagram of How our API Works

Our API as shown in fig 4.3 works by the following:-

- **Station Side:** The Wi-Fi module sends seismic data using UDP.
- **Translator:** Fetches seismic data from the station-side API using GET requests, then it processes and formats the data as required and sends it to the backend using a POST request.
- **Backend (port 3001):** It receives data from the translator and uses SQL INSERT commands to store it in the database (port 3306). It also handles SQL SELECT commands to retrieve data from the database.
- **Database (port 3306):** Stores seismic data received via the backend and provides stored data to the backend when requested.
- **Frontend (port 3000):** Fetches data from the backend through various APIs as seen in fig 3.6 above, and it displays the data for users, enabling visualization or monitoring of seismic events.

4.2.6 Detailed Use Case Diagram

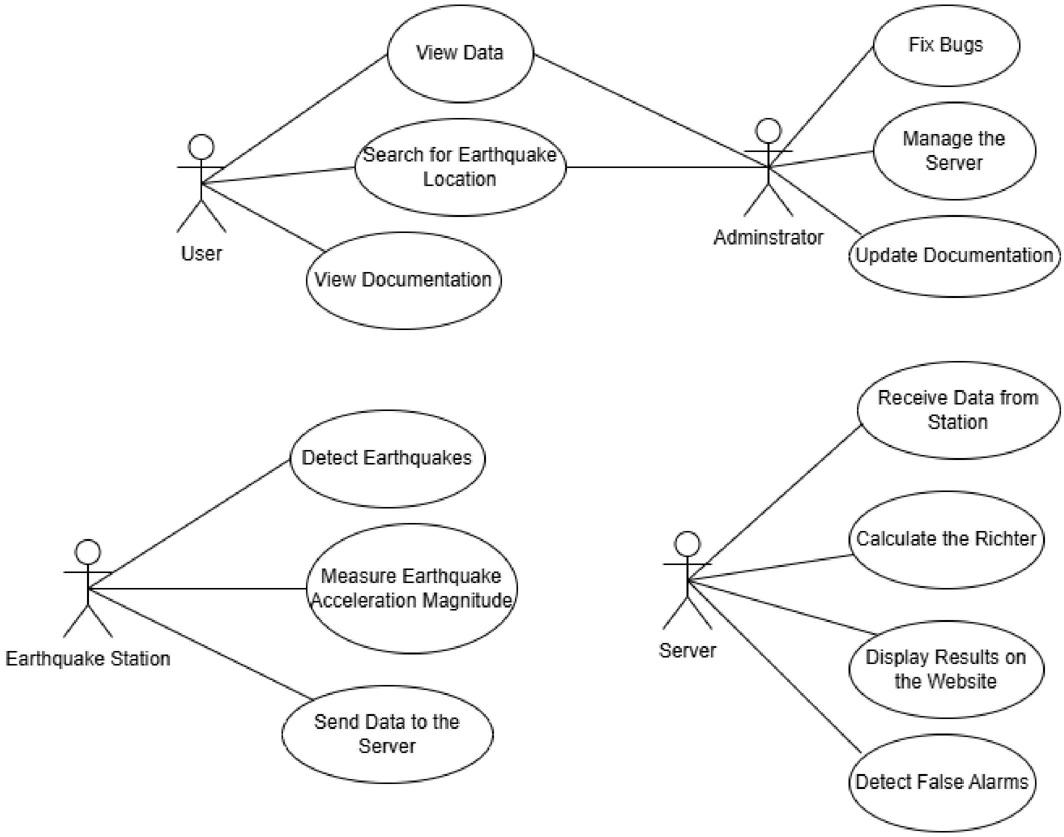

Figure 4.4: Use Case Diagram of our System

4.2.7 Detailed Sequence Diagram

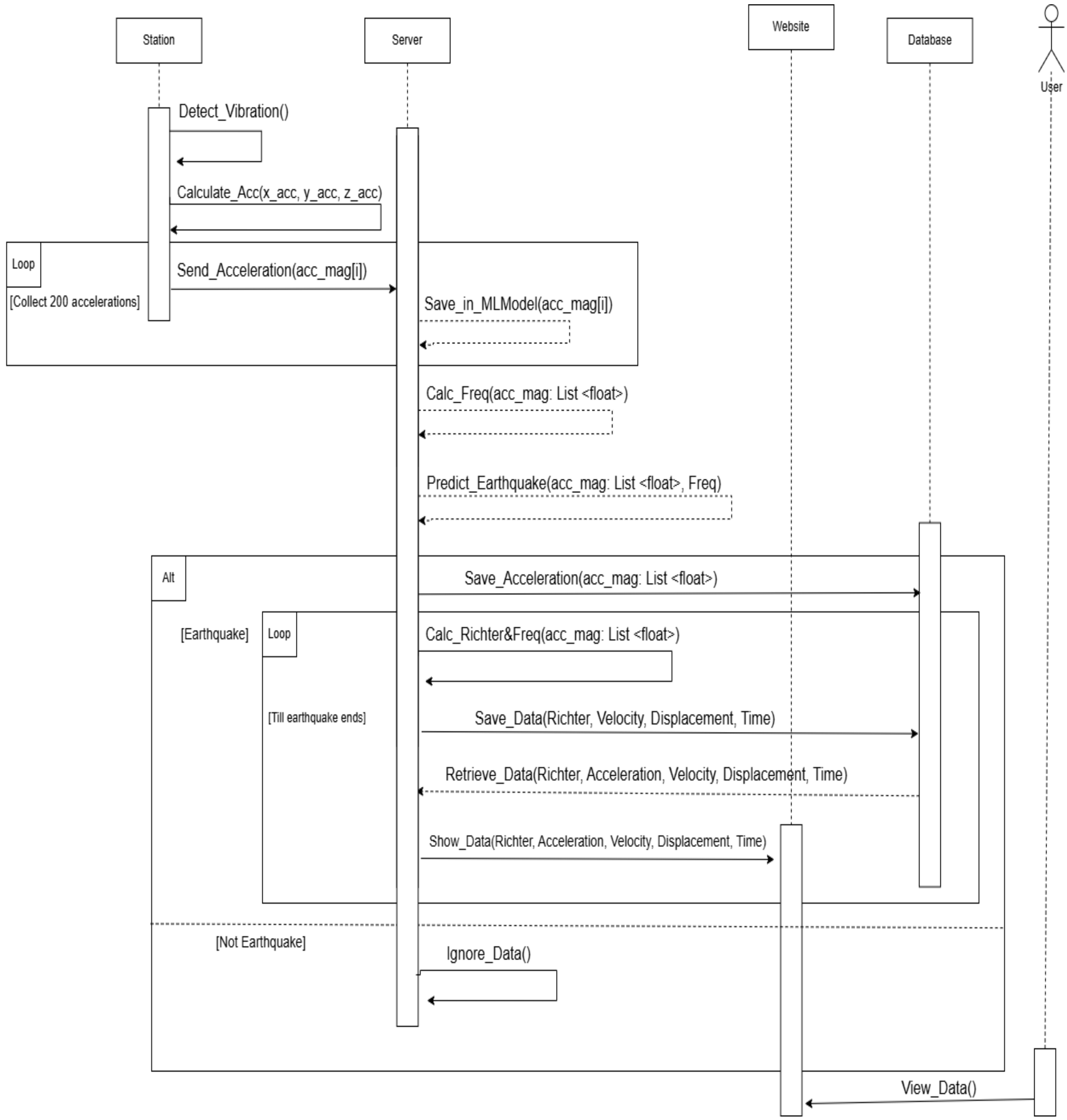

Figure 4.5: Sequence Diagram of our System

4.2.8 Detailed Activity Diagram

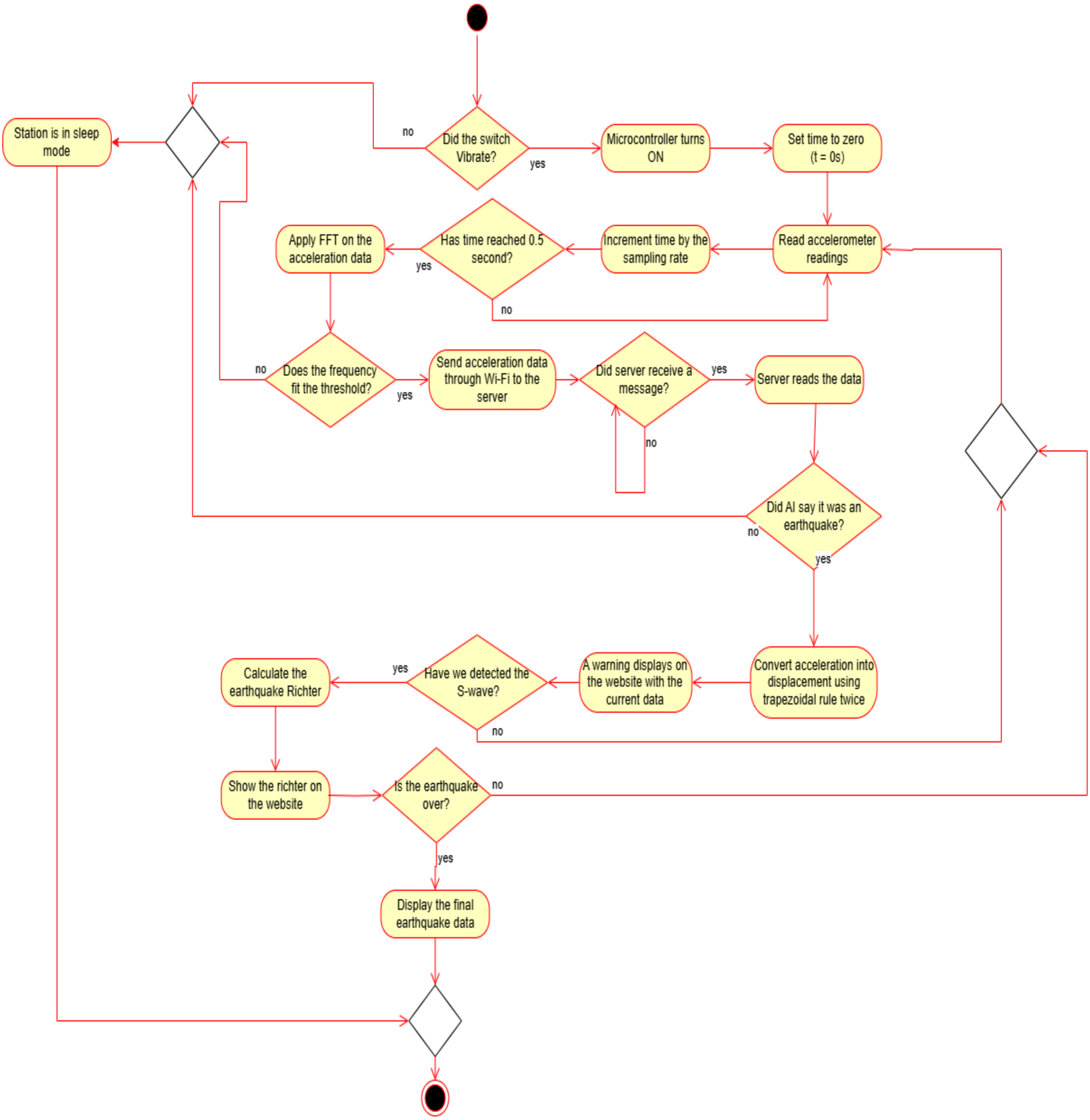

Figure 4.6: Activity Diagram of our System

4.2.9 Entity Relationship Diagram

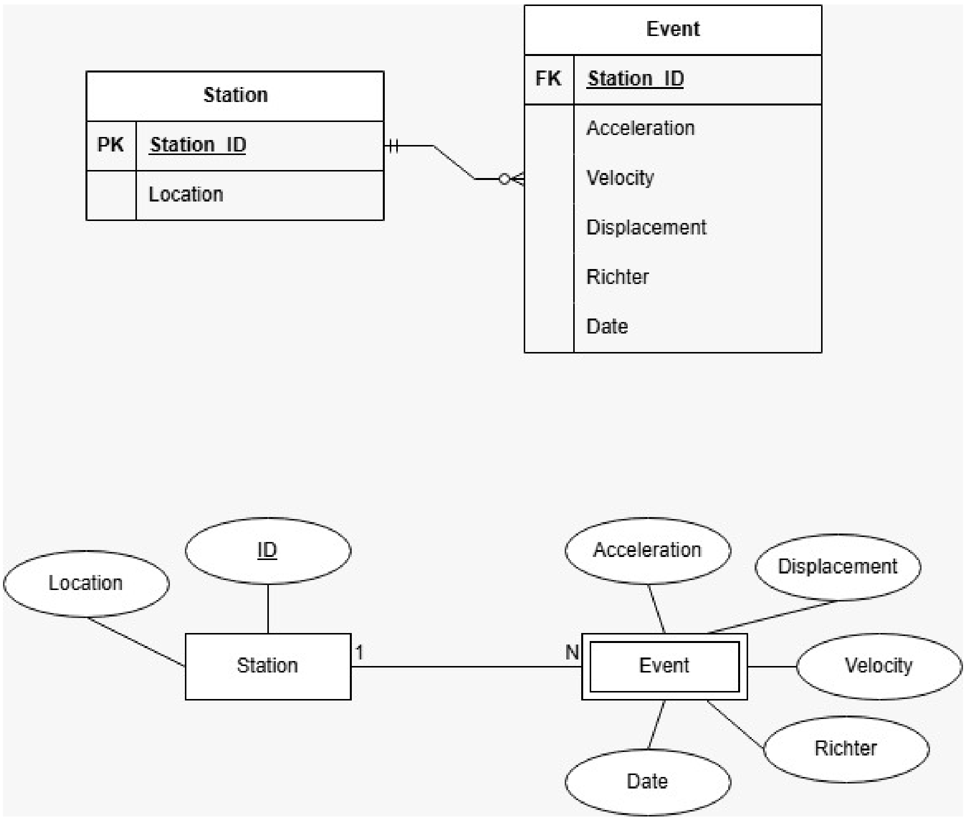

Figure 4.7: Entity Relationship Diagram of our System

The SQL code for the ER diagram is provided in algorithm 3 in the sample code appendix A.

- **Station table**, which has the following attributes:
  - **Station ID (Primary key):** Is an ID for each station and is used to differentiate between each station.
  - **Location:** Used to describe where the station is.
- **Events table**, which has the following attributes.
  - **Station ID:** Here station ID is a foreign key, meaning the events table is related to the station table through the Station ID.
  - **Acceleration, Velocity and Displacement:** They each are used to store the acceleration, velocity and displacement.
  - **Richter:** Used to place the richter after its been calculated.
  - **Date:** Used to store the date of when the earthquake has occured.

4.2.10 System Design Wireframe (Interface)

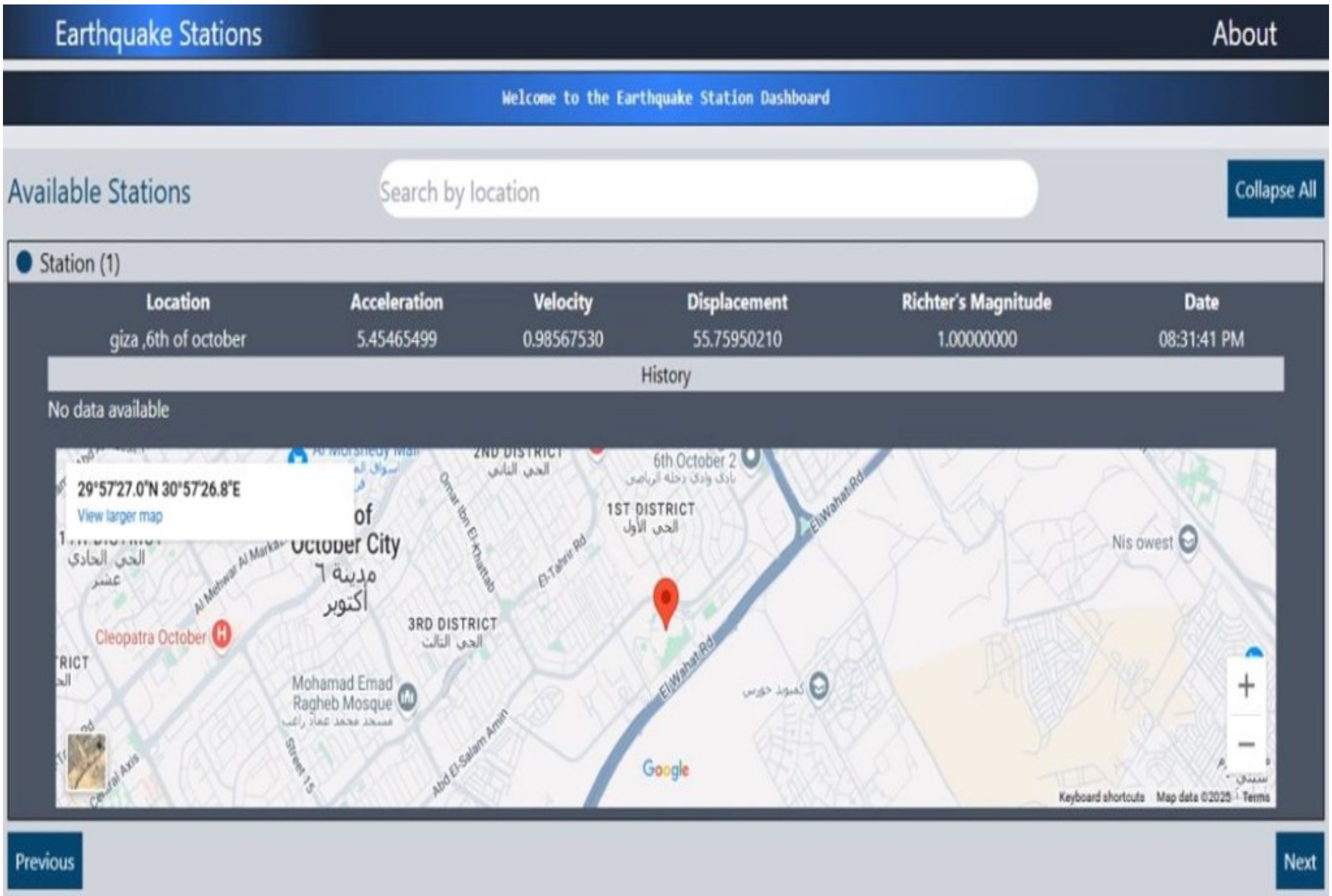

Figure 4.8: High-Fidelity Wireframe of our System

4.3 Circuit Diagram

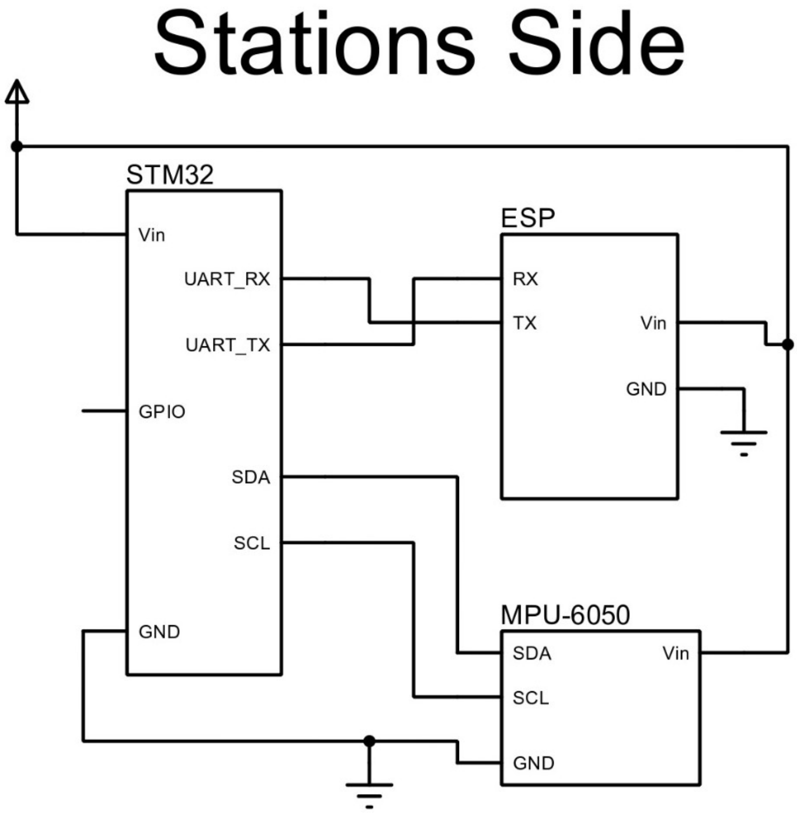

Figure 4.9: Circuit Diagram of our System

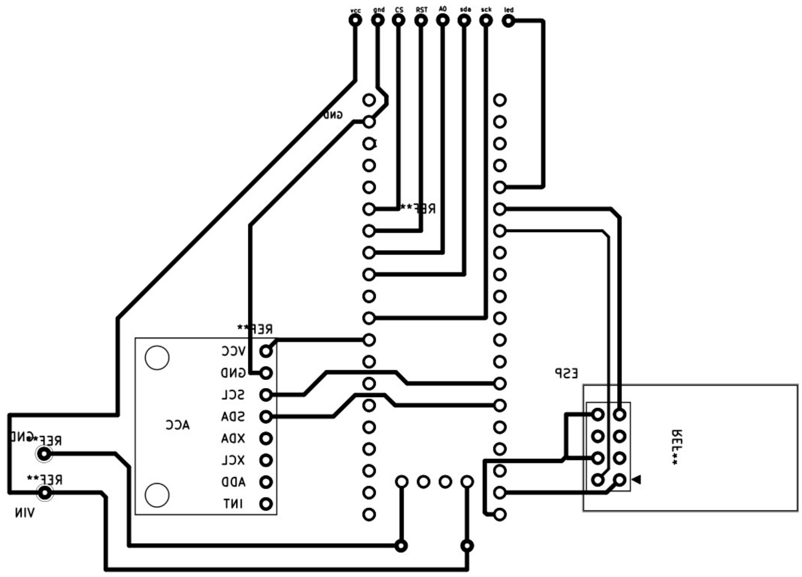

Figure 4.10: PCB Design Diagram of our Station

4.4 Case Design

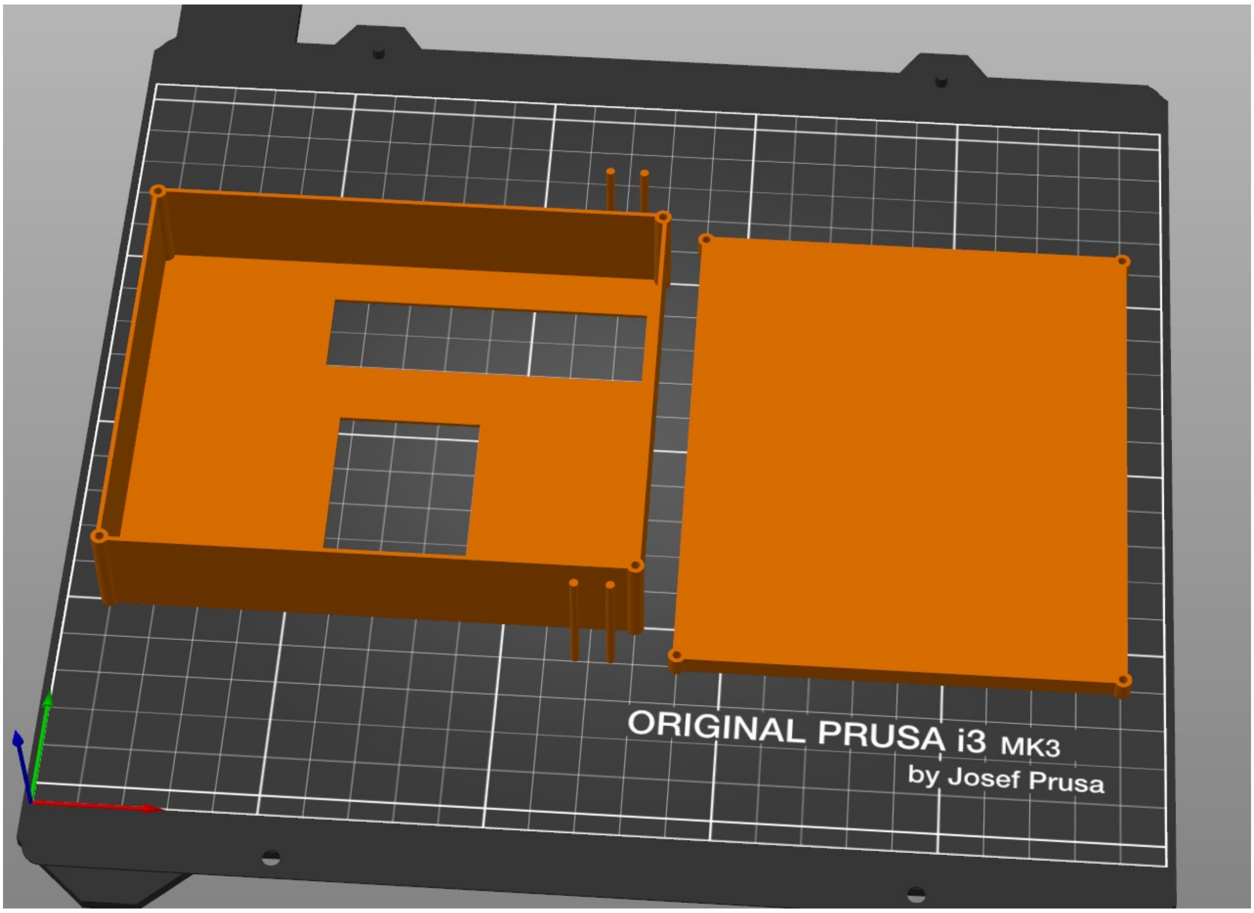

Figure 4.11: Case Design for the Station

4.5 Summary

The system consists of key hardware components, including a single accelerometer for earthquake detection, an STM32 microcontroller for low-level control and sufficient memory, and a Wi-Fi module for fast and reliable data transmission. It is powered by Li-ion batteries with a battery case for easy maintenance. Peripheral components like a USB to TTL connector and STM32 programmer are used for prototyping and uploading code. For software, STM32CubeIDE is chosen for efficient development, while C is used for microcontroller programming and Python for server-side tasks. The server is managed with Docker, using MySQL or SQLite for data storage, and a web interface provides real-time monitoring and visualization. This combination of hardware and software ensures the system is cost-effective, scalable, and suitable for remote earthquake detection.

Chapter 5

Implementation

5.1 Hardware Implementation

The hardware implementation of the earthquake detection system is designed to ensure **high accuracy, low power consumption, and real-time response** to seismic events. The system comprises **three main subsystems**: the **sensor module**, the **processing and communication unit**, and the **power management**. These components work together to detect seismic activity, process acceleration data, and transmit the results to a centralized server. You can view the circuit design in section 4.3

5.1.1 Sensor Module (Seismic Data Acquisition)

The **MPU-6050** inertial measurement unit (IMU) is used for real-time motion sensing. This sensor provides **three-axis acceleration data**, which is critical for detecting and characterizing seismic events.

- **Key Features & Implementation:**
  - **Three-axis accelerometer:** Measures ground acceleration in **X, Y, and Z** directions, allowing precise detection of ground movement.
  - **Configurable Sensitivity:** The accelerometer is set to $\pm 2g$ **sensitivity** to capture small-scale tremors with high accuracy.
  - **High Sampling Rate:** Configured to operate at **1,000 Hz** (1,000 samples per second) to ensure fast response to rapid ground motions.
  - **I2C Communication:** Data is transmitted to the microcontroller via the **I2C protocol**, minimizing latency in data retrieval.

- **Calibration & Testing:**
  - The sensor is **calibrated** using a **known reference surface** to eliminate bias in acceleration readings.
  - Initial tests were conducted using **controlled vibration sources** to verify measurement accuracy.

5.1.2 Processing and Communication Unit

The **STM32 Bluepill** microcontroller is responsible for **real-time processing and transmission** of seismic data. It is programmed to analyze acceleration data and send alerts when an earthquake event is detected.

Key Functionalities:

- **Data Transmission via ESP-01 Wi-Fi Module:**
  - The microcontroller interfaces with the **ESP-01 Wi-Fi module** via **UART** to send real-time seismic data.
  - **UDP protocol** is used to ensure **low-latency, connectionless data transfer** to the central server.
  - A **heartbeat signal** is periodically sent to verify system connectivity.
- **Event-Based Processing for Power Efficiency:**
  - The microcontroller remains in **sleep mode** until a significant vibration is detected, this saves power allowing the circuit to last longer when its using its backup battery.
- **Communication Testing:**
  - A simulated earthquake was generated using a **bass shaker**, and the resulting data was successfully received and verified at the server.
  - The **latency of UDP transmission** was measured and consistently optimized to remain under **4 milliseconds**, demonstrating ultra-low-latency performance.
  - For comparison, public early warning systems such as Japan's public alerting channels typically achieve best-case latencies of **2–3 seconds** [50], while the ShakeAlert system in the United States has reported **over 5 seconds** of communication delay for more than 85% of users during Wireless Emergency Alert (WEA) testing in Oakland, California [50].
  - These results highlight that our UDP-based communication approach offers significantly faster delivery than conventional early warning systems, making it suitable for real-time seismic detection and alerting.

5.1.3 Power Management System

The system is designed for **energy efficiency** to ensure **long-term, autonomous operation** in remote locations.

Key Components:

- **3.7V Li-ion Battery (Rechargeable, 1500mAh):** Provides stable power for continuous operation.
- **Low-Power Sleep Mode:**
  - The STM32 enters **sleep mode** when no motion is detected.
  - The accelerometer **remains active at ultra-low power consumption** to detect potential seismic activity.

Testing & Optimization:

- The system was tested in an **off-grid environment** to validate its ability to run on battery power for **extended durations**.
- Power consumption was analyzed under **normal conditions** and during **seismic events**, ensuring an optimized balance between responsiveness and energy efficiency.

5.2 Software Implementation

The software implementation of the earthquake detection system is designed to deliver a robust, accurate, and real-time analysis of seismic events by integrating several key components. These components include firmware development for the microcontroller, server-side processing for advanced signal analysis and data management, and a web-based user interface for real-time monitoring. The integration of these elements ensures that the system can detect and analyze seismic events promptly and issue early warnings to the end users, even in resource-limited or remote environments.

5.2.1 Firmware Development (STM32 - C Programming)

The STM32 microcontroller acts as the system's front-line processor, responsible for capturing and analyzing real-time acceleration data from the MPU-6050 sensor. The firmware is developed in the C programming language using STM32CubeIDE, which provides low-level access to the hardware, enabling efficient management of processing and power consumption. Key functionalities implemented in the firmware include:

- **Reading Acceleration Data:** The microcontroller continuously samples acceleration data from the MPU-6050 sensor across the X, Y, and Z axes. This high-frequency sampling ensures that even brief seismic events are captured.

- **Signal Processing and FFT Analysis:** In order to distinguish actual seismic events from ambient noise, the firmware applies a Fast Fourier Transform (FFT) algorithm. By converting the time-domain acceleration data into the frequency domain, the system is able to identify dominant frequency components and filter out irrelevant vibrations. This approach is crucial for minimizing false alarms, as further detailed in Section 3.4.1.
- **Data Transmission:** Once the data is processed and relevant features are extracted, the firmware packages the information and transmits it via the ESP-01 Wi-Fi module to the central server. This transmission is designed to be both reliable and energy-efficient.
- **Event-Triggered Power Management:** To maximize battery life and ensure long-term operation in remote deployments, the firmware employs an event-driven strategy. The microcontroller remains in a low-power state until it detects significant vibrations, at which point it switches to full processing mode.

In addition, the firmware includes error-handling routines and periodic calibration mechanisms to adapt to varying environmental conditions, ensuring continuous and accurate performance.

5.2.2 Server-Side Processing (Python)

The server-side component is implemented in Python and plays a critical role in the centralized processing of seismic data collected from multiple sensor nodes. Its responsibilities include:

- **Data Reception and Parsing:** The server listens to UDP port, extracts the transmitted data, and converts it into a structured format suitable for further analysis.
- **Database Management:** All incoming earthquake event data is stored in a MySQL database. This organized storage enables both real-time processing and historical analysis of seismic events.
- **Seismic Signal Analysis:** Advanced algorithms, including the STA/LTA method, are applied to the data to detect the arrival times of P-waves and S-waves. This detection is critical for accurately estimating the earthquake's onset and intensity, this step is explained in further details in section 3.2.4.
- **Richter Magnitude and Epicentral Distance Calculation:** The server computes the earthquake's Richter magnitude by applying the logarithmic Richter formula on the measured ground displacement. Furthermore, by comparing the arrival times of the P- and S-waves, it estimates the epicentral distance from each sensor station, this step is explained in further details in section 3.2.4.

- **Real-Time Alert Generation:** Upon confirmation of a significant seismic event, the server issues real-time alerts that are displayed on the user interface and can be sent as notifications to mobile devices.

The server-side application is containerized using Docker, which facilitates deployment across diverse operating systems and ensures consistency across different development and production environments.

5.2.3 User Interface

The web-based dashboard is the primary interface through which users interact with the earthquake detection system. It is developed using modern web technologies (React, and Tailwind CSS) and provides a clear, intuitive visualization of seismic data. Its key features include:

- **Real-Time Data Visualization:** The dashboard displays live sensor data, including acceleration, velocity, displacement, and computed Richter magnitude. Graphs, charts, and gauges are used to represent seismic activity in an easily interpretable format.

- **Historical Data Analysis:** Users can access archived seismic data to study past events, identify trends, and perform in-depth analyses of earthquake patterns.

- **Alert Notifications:** The interface is designed to generate immediate visual and auditory alerts when a seismic event is detected. Additionally, notifications can be pushed to mobile devices, ensuring users are promptly informed.

- **User-Centric Design:** The interface is optimized for both desktop and mobile devices, ensuring accessibility and ease-of-use. The design emphasizes clarity and functionality to facilitate rapid decision-making during emergencies.

Overall, the web interface bridges the gap between the complex backend processing and the end users, providing actionable insights in real time.

5.3 System Integration

The overall system is architected as a multi-layered solution comprising the **Sensor Layer**, **Communication Layer**, and **Processing and Visualization Layer**. This integrated design ensures seamless data flow from the point of measurement to the end-user display, enabling the system to deliver early earthquake warnings with minimal delay.

5.3.1 Sensor Layer

The sensor layer encompasses the hardware responsible for detecting seismic activity. This layer is designed to be both robust and energy efficient:

- **IMU (MPU-6050) Accelerometer:** The MPU-6050 sensor measures ground acceleration along three axes and provides the critical input required for seismic detection.
- **Microcontroller (STM32):** This low-power microcontroller processes the raw sensor data in real time, performing essential operations such as FFT analysis to identify seismic frequencies and filter out noise.
- **Power Management:** An efficient power management subsystem ensures that the sensor node operates only when significant vibrations are detected, thereby conserving battery life.

This layer is optimized to function under diverse environmental conditions and is critical for the overall reliability of the earthquake detection system.

5.3.2 Communication Layer

The communication layer is responsible for reliably transmitting the processed sensor data to the central server:

- **Wi-Fi Module (ESP-01):** The sensor nodes are equipped with Wi-Fi modules that use the UDP protocol to send real-time data to the server.
- **Network Protocols:** The system leverages standard UDP for lightweight and low-latency communication between the station and the backend server. UDP was chosen over TCP due to its reduced overhead and suitability for real-time applications, where the priority is fast delivery over guaranteed packet order. In the context of earthquake detection, minor packet losses are acceptable compared to the delay introduced by connection-oriented protocols. The seismic data packets are transmitted immediately upon detection, ensuring timely updates to the monitoring platform.

Effective communication is essential to guarantee that the seismic data reaches the processing server with minimal latency, thereby enabling rapid detection and alert dissemination.

5.3.3 Processing and Visualization Layer

At the top of the system stack, the processing and visualization layer is tasked with analyzing the aggregated sensor data and presenting it to end users:

- **Data Processing Server:** The central server runs complex signal processing algorithms to detect seismic events, calculate the Richter magnitude, and estimate the earthquake's epicentral distance.
- **User Interface and Alerts:** Processed data is transmitted to a web-based dashboard where users can view real-time visualizations and receive prompt alerts. The interface is designed to be user-friendly and accessible, providing clear and concise information during emergencies.

This layer plays a vital role in ensuring that the earthquake detection system is not only technically robust but also practically useful for decision-makers and the general public.

5.4 Summary

The architecture of this project was designed with a strong focus on achieving high accuracy, minimal latency, and energy-efficient operation in remote environments.

On the **hardware side**, the system integrates a sensitive **MPU-6050 accelerometer**, a low-power **STM32 Bluepill microcontroller**, and an **ESP-01 Wi-Fi module** configured for UDP transmission. The use of a **1000 Hz sampling rate** and $\pm 2g$ sensitivity ensures precise detection of seismic events, while a **vibration-triggered power management** system allows the node to operate autonomously for extended durations using a rechargeable Li-ion battery. The measured communication latency was consistently under **4 milliseconds**, a substantial improvement over traditional early warning systems that exhibit latencies of several seconds.

On the **software side**, the STM32 firmware was developed in C using STM32CubeIDE and includes routines for real-time data acquisition, FFT-based signal processing, and event-triggered transmission. The server-side application, implemented in Python, performs advanced seismic analysis including STA/LTA detection, magnitude estimation using the Richter scale, and epicentral distance calculation based on P- and S-wave arrival times. All events are logged in a MySQL database and analyzed for both immediate alerts and long-term insights.

The **web-based dashboard**, built with React and Tailwind CSS, provides users with intuitive real-time visualizations, historical data access, and immediate alerts. This interface serves as the primary point of interaction between the end users and the backend system.

By integrating all these components into a cohesive, layered system, we have successfully implemented a scalable, responsive, and cost-effective earthquake monitoring solution. The system's ability to deliver real-time alerts with sub-second latency makes it highly suitable for early warning applications in both urban and remote deployments.

Chapter 6

Testing

6.1 Testing Setup

The testing phase focuses on verifying the accuracy, efficiency, and reliability of the system's hardware and software components. The setup includes:

- **Hardware Components:**Dayton audio BST-2 Bass Shaker, TPA3110 sound amplifier, STM32 microcontroller, MPU-6050 accelerometer, ESP-01 Wi-Fi module, and power management circuit.
- **Software Environment:** STM32CubeIDE for firmware development, Python-based backend, and a React web interface.
- **Testing Tools:** Debugging utilities for firmware analysis, and software simulators for testing data transmission.

6.2 Testing Separate H/W or S/W Components and Units

The testing is conducted at different levels to ensure proper functionality, the different testing levels are **unit testing, integration testing** and **system testing**:

6.2.1 Unit Testing

Unit testing focuses on evaluating each component of the system in isolation to verify that it functions as expected [51]. The following unit tests were conducted:

- **Microcontroller Testing:** Individual tests were performed on the STM32 microcontroller to ensure correct operation of GPIO, UART, SPI, and I2C communication interfaces. Debugging tools were used to monitor input/output signals and validate expected responses.
- **Sensor Testing:** The MPU-6050 accelerometer was tested for accurate motion detection, ensuring correct conversion of acceleration values and response to physical movements.

- **Wi-Fi Module Testing:** The ESP-01 module was tested for stable connectivity, verifying that it could establish a connection with the server and transmit data without delays.
- **Power Management Testing:** The system's low-power mode was tested to ensure proper activation and deactivation in response to vibrations, optimizing battery efficiency.

6.2.2 Integration Testing

Integration testing ensures that multiple system components work together seamlessly [51]. The following integration tests were conducted:

- **Sensor-to-Microcontroller Communication:** Verified that data from the MPU-6050 was correctly processed by the STM32 microcontroller and that sensor readings were within expected ranges.
- **Microcontroller-to-Wi-Fi Module Communication:** Ensured that the STM32 properly interfaced with the ESP-01 via UART, sending seismic data without transmission errors.
- **Data Transmission to Server:** Checked that the ESP-01 correctly transmitted data to the backend server using UDP, confirming data integrity and minimal latency.
- **Backend-to-Frontend Integration:** Validated that the server correctly processed seismic data and displayed real-time updates on the web interface without delays.

6.2.3 System Testing

System testing evaluates the entire system in a real-world setting to ensure it meets performance requirements [51]. To simulate realistic earthquake conditions during testing, a **bass shaker** was integrated into the setup, a bass shake is a device that produces vibrations based on low frequency signal from audio devices, so we can convert earthquake signals into sound signals to play through the bass shaker to produce an earthquake like vibrations. This component was connected to an audio amplifier and controlled via a python sound signal generator to produce low-frequency vibrations that mimic seismic activity. The bass shaker provided repeatable, measurable, and tunable vibrations, making it ideal for system validation. The setup for the bass shaker is shown in figure 6.1

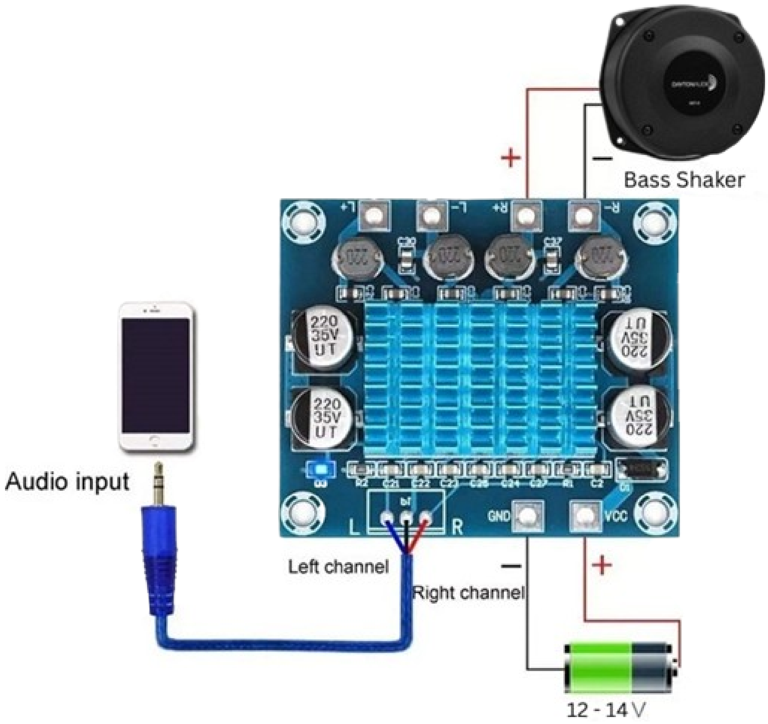

Figure 6.1: Earthquake Simulator Circuit Design with Bass Shaker

The following aspects were tested:

- **Seismic Event Detection (Simulated):** The system was placed on the bass shaker platform and subjected to earthquake like vibrations. The system reliably detected these artificial quakes, confirming the effectiveness of the accelerometer and the machine learning model in logging and classifying events.
- **Seismic Event Detection (Real):** In addition to simulated testing, the deployed system successfully detected and logged **two actual earthquake events** that were felt in Egypt during its runtime. These real-world detections validated both the sensitivity of the accelerometer and the reliability of the AI model under natural, unpredictable seismic conditions.
- **Real-Time Data Processing:** The acceleration data collected during both real and simulated tests was processed and classified by the server in real time. Results were transmitted and visualized on the website.
- **End-to-End Communication:** From vibration detection to alert display, the full communication pipeline was validated, including the triggering of the AI-based prediction, frequency check, and final decision output.
- **Repeatability and Control:** Using the bass shaker ensured consistent test inputs, allowing repeated trials with identical motion profiles. This reproducibility helped in fine-tuning threshold values and verifying system robustness across multiple sessions.

- **Reliability Under Network Constraints:** Various levels of network quality were emulated while running the bass shaker simulation to evaluate the system's tolerance to transmission delays or packet loss. The system continued to function correctly even under limited bandwidth.

6.2.4 Testing Integrated System

Integration tests validate the interaction between the station, server, and web interface. The main tests include:

- **Data Transmission Accuracy:** Ensuring UDP-based communication achieves low-latency and reliability.
- **Sensor Data Verification:** Checking accelerometer readings against expected seismic patterns.
- **Microcontroller Response Time:** Ensuring real-time activation upon seismic detection.

6.2.5 A/B Testing

- **ESP-01 Web Server vs. Direct UDP Transmission:** Initial testing involved hosting a web server on the ESP-01 module to serve data to clients upon request. However, this approach introduced significant latency due to the need for the ESP-01 to establish, process, and close connections for each data transmission cycle. Additionally, the server-based approach imposed limitations on the system's ability to handle concurrent requests efficiently. By contrast, switching to direct UDP transmission eliminated the need for connection management, allowing the system to send data instantaneously with minimal overhead. This change resulted in improved response times, reduced processing delays, and enhanced overall performance, making UDP the optimal choice for real-time seismic data transmission.
- **Machine Learning Models:** As seen in section 3.6 we tried out different machine learning models which were naive bayes, gradient boosting, lightGBM and random forest and we finally decided to go with lightGBM due to it being the fastest model while still providing good accuracy.
- **Earthquake Simulation:** We tried multiple ways to attempt to simulate earthquakes, first we attempted a vibration motor [52] however this fails due to the inability to control the way the motor vibrates, so then we tried stepper motors, where we would use two stepper motors to simulate an earthquake by moving the surface according to earthquake acceleration data and in theory this method would work, however practically it failed due to the stepper motors not being fast

enough to simulate an earthquake, therefore we have finally decided to go with bass shaker which is a device that produces vibrations based on low frequency signal from audio devices [53], its mainly used for people playing racing games to feel as if they were actually racing, however we converted earthquake signals into sound signals which goes through a sound amplifier which then sends it to the bass shaker to vibrate similar to an earthquake.

6.2.6 Test Cases

Table 6.1: Test case for seismic event detection.

Project Name:	IoT Wireless Earthquake Station
Test Case ID:	EQ_01
Test Designed by:	*Abdulrahman Sallam*
Test Priority:	High
Test Designed date:	*March 6th, 2025*
Module Name:	Seismic Event Detection
Test Executed by:	*Mohamed Abdelfattah*
Test Title:	Verify real-time earthquake detection
Test Execution date:	*March 7th 2025*
Description:	Ensure the system accurately detects and processes seismic activity in real-time
Preconditions:	The accelerometer is properly calibrated and connected
Dependencies:	Stable power supply and network connection

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Simulate seismic activity	Simulate Earthquake Shaking	System should detect movement	System detects movement	Pass
2	Process accelerometer data	MPU-6050 output	Data should be converted into displacement values	Data conversion is correct	Pass
3	Transmit data to server	UDP packet to backend	Data should be received within 1s	Data received within 0.5s	Pass
4	Display event on web interface	Frontend dashboard update	Seismic event should be logged and displayed	Event displayed correctly	Pass

Post-conditions:
The detected seismic event is successfully logged in the system database and displayed on the web interface in real-time. The system remains operational, ready for subsequent detections. Any transmitted data is stored for future analysis, and alerts are sent if the detected magnitude exceeds a predefined threshold.

Table 6.2: Test case for power efficiency.

Project Name:	IoT Wireless Earthquake Station
Test Case ID:	EQ_02
Test Designed by:	*Abdulrahman Sallam*
Test Priority:	Medium
Test Designed date:	*March 6th, 2025*
Module Name:	Power Management
Test Executed by:	*Mohamed Abdelfattah*
Test Title:	Verify system minimizes power consumption in idle state
Test Execution date:	March 10th, 2025
Description:	Ensure the microcontroller enters sleep mode when no seismic activity is detected
Preconditions:	System is fully charged and operational
Dependencies:	Battery module

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Let system remain idle	No seismic activity simulated	System should enter low-power mode	MCU enters sleep mode	Pass
2	Simulate earthquake	Apply threshold-level shaking	System should activate	System activated	Pass
3	Monitor power consumption	Measure with multimeter	Power usage should be minimal in idle mode	Power usage is less in sleep mode	Pass

Post-conditions: The system effectively reduces power consumption in sleep mode while maintaining responsiveness to seismic events.

Table 6.3: Test case for real-time web interface updates.

Project Name:	IoT Wireless Earthquake Station
Test Case ID:	EQ_03
Test Designed by:	*Abdulrahman Sallam*
Test Priority:	High
Test Designed date:	*March 6th, 2025*
Module Name:	Web Interface
Test Executed by:	*Mohamed Abdelfattah*
Test Title:	Verify real-time updates on the web dashboard
Test Execution date:	March 9th, 2025
Description:	Ensure seismic events detected by the system are immediately displayed on the monitoring dashboard.
Preconditions:	The system detects an earthquake.
Dependencies:	Stable internet connection and web server availability.

Step	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/Fail)**
1	Simulate seismic activity	Shake table with known vibration	System should detect movement	System detects movement	Pass
2	Process seismic data	MPU-6050 sensor output	Data should be processed into displacement values	Data processing is correct	Pass
3	Transmit processed data	UDP packet to backend server	Data should be received within 1s	Data received within 0.5s	Pass
4	Update web dashboard	Frontend receives server data	Dashboard should display seismic event	Event displayed correctly	Pass
5	Verify timestamp accuracy	Compare detection and display time	Delay should be < 1s	Delay is within limit	Pass

Post-conditions:
The detected seismic event is successfully logged in the system database and displayed on the web dashboard in real-time. The system remains operational and ready for subsequent detections. Users can access historical seismic event data for future analysis, and alerts are triggered if the detected magnitude exceeds a predefined threshold.

Table 6.4: Test case for false alarm detection with frequency threshold.

Project Name:	IoT Wireless Earthquake Station
Test Case ID:	EQ_04
Test Designed by:	*Abdulrahman Sallam*
Test Priority:	High
Test Designed date:	*April 3rd, 2025*
Module Name:	False Alarm Detection
Test Executed by:	*Abdulrahman Sallam*
Test Title:	Verify false alarm rejection using signal processing
Test Execution date:	*April 5th, 2025*
Description:	Ensure the system filters out non-seismic vibrations such as machinery noise, vehicle motion, or human activity.
Preconditions:	Background noise sources present near the sensor.
Dependencies:	Implementation of Fast Fourier Transform (FFT), frequency threshold and random forest model.

Step	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/Fail)**
1	Generate background noise (e.g., human steps, traffic)	Various environmental noise sources	No false seismic events detected	No false alarms detected	Pass
2	Introduce actual seismic activity	Simulated earthquake shaking	System should detect earthquake correctly	System correctly identifies earthquake	Pass

Post-conditions:
The system successfully differentiates between seismic events and background noise, minimizing false alarms. Non-seismic vibrations such as human activity and vehicle motion are effectively filtered out using frequency and acceleration thresholds. If an earthquake is identified, the system triggers an alert and logs the event for further analysis.

Table 6.5: Test case for Richter magnitude calculation.

Project Name:	IoT Wireless Earthquake Station
Test Case ID:	EQ_05
Test Designed by:	*Abdulrahman Sallam*
Test Priority:	High
Test Designed date:	*March 9th, 2025*
Module Name:	Seismic Data Processing
Test Executed by:	*Abdulrahman Sallam*
Test Title:	Verify Richter magnitude calculation based on reference seismic acceleration data
Test Execution date:	*March 12th, 2025*
Description:	Ensure the system calculates the Richter magnitude using ground acceleration and seismic wave amplitude.
Preconditions:	System is receiving acceleration data from an earthquake event.
Dependencies:	Implementation of signal processing algorithms, and Richter calculation formula.

Step	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/Fail)**
1	Capture real-time acceleration data	Seismic event acceleration	Data successfully recorded	Data recorded correctly	Pass
2	Convert acceleration to ground displacement	PGA-to-displacement formula applied	Correct displacement values computed	Displacement computed	Pass
3	Compute Richter magnitude using standard formula	Apply the richter equation using displacement	Computed magnitude matches reference data	Magnitude computed correctly	Pass
4	Compare system-calculated magnitude with strong motion data center [34] reference	Reference earthquake magnitude dataset	Difference within acceptable error range (< 0.2)	Results within tolerance	Pass

Post-conditions:
The system correctly calculates the Richter magnitude from acceleration data, ensuring accurate seismic event classification. The calculated values are validated against reference earthquake magnitudes, maintaining an acceptable error margin.

Table 6.6: Test case for sensor to STM32 communication.

Project Name:	IoT Wireless Earthquake Station
Test Case ID:	EQ_06
Test Designed by:	*Abdulrahman Sallam*
Test Priority:	High
Test Designed date:	*April 1st, 2025*
Module Name:	Sensor to STM32 Communication
Test Executed by:	*Mohamed Abdelfattah*
Test Title:	Verify real-time processing of acceleration data from MPU-6050
Test Execution date:	*April 2nd, 2025*
Description:	Ensure the STM32 processes acceleration data from the MPU-6050 and updates real-time motion data.
Preconditions:	MPU-6050 connected and properly calibrated.
Dependencies:	Stable power and microcontroller setup.

Step	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/Fail)**
1	Activate accelerometer	Simulate motion	System should detect motion and begin data processing	Motion detected, data processing begins	Pass
2	Process acceleration data	MPU-6050 output	Data should be transformed into displacement values	Data conversion is accurate	Pass
3	Monitor data processing	Processed data output	Data should be updated in real-time with minimal delay	Data updated in real time	Pass

Post-conditions: The system processes acceleration data accurately and continuously updates motion detection status.

Table 6.7: Test case for STM32 to ESP-01 communication.

Project Name:	IoT Wireless Earthquake Station
Test Case ID:	EQ_07
Test Designed by:	*Abdulrahman Sallam*
Test Priority:	High
Test Designed date:	*April 1st, 2025*
Module Name:	STM32 to ESP-01 UART Communication
Test Executed by:	*Mohamed Abdelfattah*
Test Title:	Verify reliable UART communication between STM32 and ESP-01
Test Execution date:	*April 3rd, 2025*
Description:	Verify that the STM32 microcontroller sends data to the ESP-01 via UART reliably, without loss or delay.
Preconditions:	STM32 and ESP-01 are connected via UART.
Dependencies:	Stable UART connection and data transmission setup.

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Send seismic data from STM32 to ESP-01	Simulated seismic data	Data should be transmitted without loss or corruption	Data received correctly without errors	Pass
2	Monitor transmission speed	Simulated data	Transmission should occur without noticeable delay	Transmission completed with minimal delay	Pass

Post-conditions:
Data is transmitted reliably from STM32 to ESP-01 via UART without errors, and the communication channel remains stable.

Table 6.8: Test case for multiple station synchronization.

Project Name:	IoT Wireless Earthquake Station
Test Case ID:	EQ_8
Test Designed by:	*Abdulrahman Sallam*
Test Priority:	High
Test Designed date:	*April 29th, 2025*
Module Name:	Server Multi-Source Integration
Test Executed by:	*Mohamed Abdelfattah*
Test Title:	Verify synchronization and correct logging of data from multiple detection stations
Test Execution date:	*May 1st, 2025*
Description:	Ensure the server correctly receives, differentiates, logs, and displays seismic data sent from multiple nodes (stations) without overwriting or timing errors.
Preconditions:	All stations (Node A, B, and C) are online and configured to send properly formatted seismic data.
Dependencies:	Reliable server backend, unique station IDs, working network infrastructure.

Step	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/Fail)**
1	Trigger seismic event on Node A	Simulated seismic signal	Server receives data tagged as Node A	Server logs and displays Node A data correctly	Pass
2	Trigger seismic event on Node B	Simulated seismic signal	Server receives data tagged as Node B	Server logs and displays Node B data correctly	Pass
3	Trigger seismic event on Node C	Simulated seismic signal	Server receives data tagged as Node C	Server logs and displays Node C data correctly	Pass
4	Trigger events simultaneously from all nodes	Simulated multi-node signals	Server processes all without overwriting or mislabeling	All events correctly labeled and displayed in sequence	Pass

Post-conditions: The server has logged all seismic events from the different nodes correctly, ensuring proper identification and temporal ordering. Frontend reflects this accurately.

Table 6.9: Test case for AI model integration for false alarm filtering.

Project Name:	IoT Wireless Earthquake Station
Test Case ID:	EQ_9
Test Designed by:	*Abdulrahman Sallam*
Test Priority:	High
Test Designed date:	*April 29th, 2025*
Module Name:	AI Model for Data Filtering
Test Executed by:	*Abdulrahman Sallam*
Test Title:	Verify AI model filters out false seismic triggers effectively
Test Execution date:	*May 2nd, 2025*
Description:	Validate that the AI model correctly classifies incoming data to distinguish between genuine seismic activity and false positives due to noise, vibrations, or anomalies.
Preconditions:	AI model is deployed and accessible by the server backend. Proper training data has been used to tune the model.
Dependencies:	Preprocessed and labeled datasets, trained AI model, consistent input data format.

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Feed genuine seismic event data to the model	Real seismic waveform	Model should classify as "earthquake"	Classified correctly as earthquake	Pass
2	Feed noise/vibration data (false positive)	Simulated mechanical noise	Model should classify as "non-earthquake"	Classified correctly as non-earthquake	Pass
3	Feed borderline/ambiguous input	Mixed anomaly data	Model gives a probability/-confidence score with low uncertainty	Model returns confidence score; still filters correctly	Pass
4	Perform real-time evaluation using streaming input	Live mixed seismic/noise data	System classifies and logs data in real time	System operates in real-time with no delay or misclassification	Pass

Post-conditions: The AI model successfully distinguishes between real earthquakes and false alarms, minimizing unnecessary alerts and improving reliability of the system.

Table 6.10: Test case for full path data flow from accelerometer to web interface.

Project Name:	IoT Wireless Earthquake Station
Test Case ID:	EQ_10
Test Designed by:	*Abdulrahman Sallam*
Test Priority:	Critical
Test Designed date:	*April 29th, 2025*
Module Name:	End-to-End Data Transmission
Test Executed by:	*Mohamed Abdelfattah*
Test Title:	Verify complete data transmission from sensor to web dashboard
Test Execution date:	*May 3rd, 2025*
Description:	Test the entire data flow pipeline: Accelerometer detects vibration, STM32 processes and formats it, ESP-01 transmits to server, server filters with AI model, and finally data is displayed on the website.
Preconditions:	System is powered on, accelerometer is active, STM32 and ESP-01 are connected, server and AI model are running, and web UI is accessible.
Dependencies:	Proper UART communication, stable Wi-Fi, server availability, AI model responsiveness, and frontend backend sync.

Step	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/Fail)**
1	Trigger vibration event on accelerometer	Simulated vibration signal	STM32 captures and formats data	Data correctly formatted and sent via UART	Pass
2	Transmit data via ESP-01 to server	Formatted UART packet	Server receives raw data packet	Data packet received and logged on server	Pass
3	Process data through AI model	Received seismic data	AI model classifies and filters event	Classification completed and result stored	Pass
4	Display on website dashboard	Classified seismic event	UI reflects new event with timestamp	Dashboard updates in real-time with correct info	Pass

Post-conditions: The entire data flow is verified from physical input to visual display. Seismic data successfully travels from accelerometer through STM32, ESP-01, and server to be filtered by the AI and finally shown on the web interface.

Table 6.11: Test case for bass shaker-based earthquake simulation.

Project Name:	IoT Wireless Earthquake Station
Test Case ID:	EQ_11
Test Designed by:	*Abdulrahman Sallam*
Test Priority:	High
Test Designed date:	*May 30th, 2025*
Module Name:	Earthquake Simulation with Bass Shaker
Test Executed by:	*Mohamed Abdelfattah*
Test Title:	Verify detection accuracy under simulated earthquake conditions
Test Execution date:	*June 1st, 2025*
Description:	Confirm the system can detect seismic motion generated by a bass shaker simulating an earthquake
Preconditions:	Bass shaker connected and configured to generate 1–20 Hz vibrations
Dependencies:	Station is mounted on the shaker, system is online, and network is active

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Activate shaker	Low-frequency earthquake signal	Station senses acceleration	Acceleration detected correctly	Pass
2	Verify detection	200-sample vibration	System identifies potential earthquake	Detected as potential earthquake	Pass
3	Process data	Acceleration array	Classifier gives earthquake status	AI prediction executed	Pass
4	Show output	Web dashboard update	Event visible with timestamp	Event displayed as expected	Pass

Post-conditions: The simulated earthquake was successfully detected, processed, and displayed on the system's dashboard. The system logged the event and remained responsive for subsequent testing.

Table 6.12: Test case for real-world earthquake detection.

Project Name:	IoT Wireless Earthquake Station
Test Case ID:	EQ_12
Test Designed by:	*Abdulrahman Sallam*
Test Priority:	Critical
Test Designed date:	*May 15th, 2025*
Module Name:	Live Earthquake Monitoring
Test Executed by:	*Mohamed Abdelfattah*
Test Title:	Validate system behavior during real earthquakes
Test Execution date:	*May 16th– June 3rd, 2025*
Description:	Ensure the system can detect and process actual earthquake activity from live sensor data
Preconditions:	System must be fully deployed and running live
Dependencies:	Real earthquake events must occur in range of the station

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Monitor sensor	Live MPU-6050 readings	Sudden vibration spike is captured	Spike recorded	Pass
2	Trigger classifier	Seismic pattern detected	AI classifies as earthquake	Detected as earthquake	Pass
3	Store and display	Event stored in database	Event shown on UI in real time	Displayed successfully	Pass
4	Log earthquake info	Magnitude and time saved	Data saved for user review	Logged with timestamp	Pass

Post-conditions:
Both real earthquakes were detected and confirmed by the system. Each event was logged, visualized on the website, and stored in the database with complete metadata (timestamp, magnitude, waveform characteristics). The system remained functional and alert-ready afterward.

Table 6.13: Test case for cross-platform system compatibility using Docker.

Project Name:	IoT Wireless Earthquake Station
Test Case ID:	EQ_13
Test Designed by:	*Abdulrahman Sallam*
Test Priority:	High
Test Designed date:	*June 25th, 2025*
Module Name:	Server-Side Processing and Deployment
Test Executed by:	*Mohamed Abdelfattah*
Test Title:	Validate multi-OS compatibility of backend using Docker
Test Execution date:	*June 25th, 2025*
Description:	Ensure the server-side system runs consistently on multiple operating systems (Windows, Linux) using Docker containers
Preconditions:	Docker must be installed and functional on both systems
Dependencies:	Proper Dockerfile and image build context must be available

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Build Docker image	Dockerfile on host OS	Image builds without error	Built successfully	Pass
2	Run container	Docker run command	System starts on both OS	UI and backend started	Pass
3	Send test data	Simulated UDP packet	Data processed and logged	Entry recorded	Pass
4	Access UI	Open dashboard URL	Interface loads correctly	Displayed without issue	Pass

Post-conditions:
The system functioned identically on both Windows and Linux hosts. The Docker container successfully built, ran, and processed seismic data. The frontend UI was accessible and functional across both platforms, validating the portability and reliability of the containerized implementation.

6.3 Problems Encountered

During testing, some challenges were identified and resolved, for example:

- **Web Server Inefficiency:** Hosting a web server on ESP-01 was extremely slow; switching to UDP has resolved this issue, as mentioned earlier in section 6.2.5 with more details.
- **Issues Power-Saving Techniques:** When we used vibration switch and transistor as seen in figure 6.2, it was supposed to make it so that it only works if it detects a vibration however it caused an issue with the Wi-Fi module where it needed time to start up causing a large amount of delay, therefore we have replaced it with the STM32 microntroller being in sleep mode when no vibration is detected.

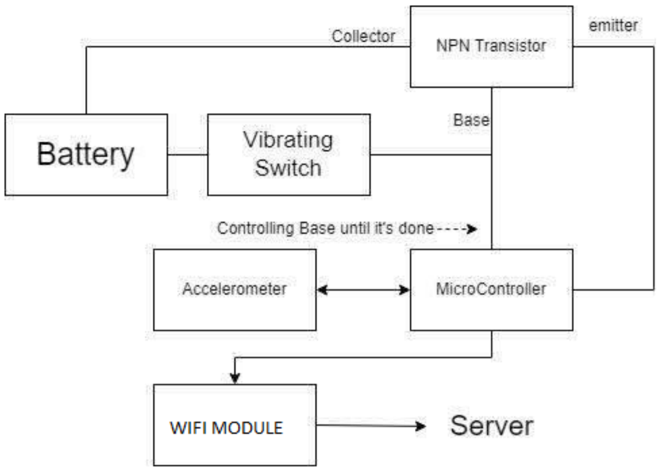

Figure 6.2: Vibration Switch and Transistor Power Saving Solution

- **Earthquake Simulation:** Although our system successfully detected two real earthquake events, we determined that this limited sample was insufficient to rigorously assess the system's reliability. Therefore, we explored methods to simulate seismic activity. Initially, we experimented with stepper motors; however, they lacked the speed required to mimic realistic earthquake conditions. We then tested vibration motors, but found that their output could not be precisely controlled. Our third option was to use a professional shake table, which is the standard equipment for earthquake simulation. Unfortunately, access to such equipment was not granted. As an alternative, we employed a bass shaker to generate controlled vibrations, as described in Section 6.2.3

6.4 Results and Discussions

The system has successfully achieved its design objectives, demonstrating robust performance across key operational metrics:

- **Precision in Seismic Event Detection:** The system exhibited high sensitivity in detecting seismic activity, effectively distinguishing between minor ground vibrations and significant tremors. The implementation of signal filtering techniques, including Fast Fourier Transform (FFT), reduced noise interference, ensuring a low false-positive rate.

- **Efficient and Low-Latency Data Transmission:** The adoption of the UDP protocol enabled real-time seismic data transmission with minimal overhead. Performance evaluation indicated that the system consistently achieved sub-second latency, enhancing its capability for early earthquake warning applications.

- **Optimized Power Management and Energy Efficiency:** The integration the STM32 microcontroller's low-power sleep mode, significantly reduced overall power consumption. Empirical testing confirmed that the system remained in low power standby mode during inactivity, maximizing battery life for long-term, autonomous deployment.

The results validate the system's effectiveness in providing **real-time, energy-efficient, and high-fidelity seismic monitoring**. These performance optimizations position it as a superior alternative to conventional earthquake detection solutions, particularly in resource-constrained or remote environments.

6.5 Summary

The testing phase successfully validated the accuracy, efficiency, and reliability of the earthquake detection system across multiple levels, including **unit testing, integration testing, and system testing**.

Hardware components such as the STM32 microcontroller, MPU-6050 accelerometer, and ESP-01 Wi-Fi module were individually tested to ensure proper functionality, stable communication, and precise motion detection. **Integration testing** confirmed seamless interactions between the sensor, microcontroller, and server, while **system testing** demonstrated the overall performance under real-world conditions.

A/B testing played a crucial role in optimizing the system, with results showing that **UDP-based data transmission significantly outperformed the ESP-01 web server approach** in terms of latency and efficiency. Power efficiency testing verified that the vibration-triggered activation mechanism effectively reduces energy consumption, allowing long-term deployment in remote environments.

Overall, the results indicate that the proposed earthquake detection system **achieves reliable real-time seismic event detection, rapid data transmission, and energy-efficient operation**. These optimizations make it a viable and superior alternative to traditional IoT-based earthquake monitoring solutions.

Chapter 7

Cost Analysis

7.1 One-Time Costs

- **ESP Wi-Fi module:** $5

  Used for communication in both the server and station.

- **STM32 Bluepill microcontroller:** $4.35

  Chosen for its superior performance compared to AVR and PIC microcontrollers, while being more cost-effective than high-end options like Raspberry Pi.

  Although Raspberry Pi is a viable alternative, it is considered overpowered for this project, and is more costly therefore it will only be used if necessary.

- **Li-ion Batteries:** $2.90

  Power supply for the system.

- **Battery case:** $1.04

  Simplifies battery connectivity.

- **IMU (MPU-6050) Accelerometer:** $2.80

  Measures and outputs magnitude to the microcontroller.

- **STM32 Programmer:** $6.01

  Used to burn code onto the microcontroller.

- **PCB:** $0.93

- **Case Design:** $0.4

Total: $23.43

Notice how the development of the application (backend,frontend,UI/UX,etc) costs are neglected? That is due to the application being an **open-source application.**

7.2 Recurring Cost

- **AWS Cloud server for the application:** $5–$20/month [54].
- **Domain registration:** $0.25–$1.25/month [55].

Total: $5.25 - $21.25 /month

7.3 Summary

The cost analysis for the proposed earthquake detection system demonstrates its affordability and practicality. The one-time hardware cost amounts to $23.43, which includes essential components like the STM32 Bluepill microcontroller for efficient performance and cost-effectiveness, along with the MPU-6050 accelerometer for seismic detection and the case design. Additional components such as the Li-ion batteries, battery case, STM32 programmer, and PCB further contribute to the system's functionality and prototyping. Notably, development costs for the application backend, frontend, UI/UX, and other associated software features are excluded, as the application will be open-source. Recurring costs primarily consist of AWS cloud hosting, ranging from $5 to $20 per month, and domain registration, which costs between $0.25 to $1.25 per month. This design ensures a low-cost, scalable, and accessible solution that is particularly suitable for resource-constrained environments.

Chapter 8

Time Plan

Here we'll showcase the timeline of our project and an estimate of how long each task will be using gantt chart which is an effecitive way to tell the details of how long a project will be done, if everything goes as planned then the gantt chart should be accurate [56].

Table 8.1: Gantt chart illustrating the main project milestones, duration, and task holder.

Task		Week in Semester												
Description	**Holder**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Define project scope and gather requirements	Both													
Research on earthquake detection methods and expenses	Abdulrahman													
Design hardware architecture	Mohamed													
Preparing the main plan	Abdulrahman													
Documenting Into Thesis	Both													
Develop the software code for STM32 microcontroller	Mohamed													
Integrate WIFI communication between nodes	Mohamed													
Developing a random forest model for false alarm detection	Abdulrahman													
Unit testing	Both													
Wiring the components	Both													
Set up and test physical server and WIFI connections	Mohamed													
Design web application for data visualization	Both													
Testing the entire system	Abdulrahman													

Chapter 9

Conclusions and Future Work

9.1 Conclusions

This project successfully demonstrates the development of a cost-effective, energy-efficient earthquake detection system utilizing an accelerometer and a Wi-Fi module. The system is designed to detect seismic activity in real-time and transmit relevant data to a centralized web-based platform, ensuring timely monitoring and analysis. The integration of an STM32 microcontroller allows for optimized signal processing and efficient data handling, contributing to the overall reliability and scalability of the system.

A key innovation of this work is its event-driven power management strategy, which employs a vibration-triggered activation mechanism. This significantly reduces power consumption by ensuring that the system remains idle during non-seismic periods while rapidly responding to detected vibrations. Such an approach enhances the device's longevity, making it well-suited for deployment in remote or off-grid environments where energy efficiency is critical.

The system's design also emphasizes modularity, enabling seamless integration with additional sensors, improved filtering techniques, or advanced data processing methods in future iterations. Rigorous testing and validation confirmed the system's ability to accurately distinguish seismic events from environmental noise, ensuring reliable performance under varying conditions.

Overall, this project contributes to the field of real-time earthquake detection by offering a compact, low-power, and easily deployable solution. Future enhancements may focus on refining detection algorithms, improving data transmission reliability, and expanding real-time visualization capabilities to further enhance early warning potential and disaster mitigation efforts.

9.2 Future Work

While the proposed system effectively detects and transmits earthquake data in real-time, several enhancements can be explored to further improve its performance and scalability.

1. **Improved Data Processing:** Enhance signal filtering techniques to better differentiate between seismic events and environmental noise.
2. **Real-Time Communication Enhancements:** Improve data transmission protocols to handle packet loss and ensure reliable earthquake data delivery.
3. **User Interface and Public Accessibility:** Expand the web-based monitoring platform to provide real-time visualization of seismic activity.
4. **Real-Time Communication Enhancements:** Improve data transmission protocols to handle packet loss, reduce latency further, and ensure robust earthquake data delivery over unreliable networks.

By addressing these areas, the system can evolve into a more **robust, scalable**, and **efficient** earthquake detection solution capable of providing real-time alerts and contributing to disaster preparedness.

References

[1] Hiroo Kanamori and Emily E. Brodsky. "The Physics of Earthquakes". In: *Physics Today* 54.6 (June 2001), pp. 34–40. ISSN: 0031-9228. DOI: 10.1063/1.1387590. eprint: https://pubs.aip.org/physicstoday/article-pdf/54/6/34/16693277/34_1_online.pdf. URL: https://doi.org/10.1063/1.1387590.

[2] H. HONDA. "Earthquake Mechanism and Seismic Waves". In: *Journal of Physics of the Earth* 10.2 (1962), pp. 1–97. DOI: 10.4294/jpe1952.10.2_1.

[3] SMS Tsunami Warning. *Earthquake | Seismology, magnitude and other units of measurement*. Accessed: December 1, 2024. n.d. URL: https://www.sms-tsunami-warning.com/pages/seismology-measurement.

[4] R. M. Allen and D. Melgar. "Earthquake early warning: Advances, scientific challenges, and societal needs". In: *Annual Review of Earth and Planetary Sciences* 47.1 (2019), pp. 361–388. DOI: 10.1146/annurev-earth-053018-060457. URL: https://doi.org/10.1146/annurev-earth-053018-060457.

[5] GeoNet. *GeoNet Earthquake Statistics*. Accessed: December 1, 2024. 2024. URL: https://www.geonet.org.nz/earthquake/statistics_long.

[6] British Geological Survey. *Is Earthquake Activity Increasing?* Accessed: December 1, 2024. n.d. URL: https://www.earthquakes.bgs.ac.uk/news/EQ_increase.html.

[7] *The Japan Tohoku Tsunami of March 11, 2011*. Accessed: December 1, 2024. 2011, pp. 5–12. URL: https://learningfromearthquakes.org/resources/the-japan-tohoku-tsunami-of-march-11-2011/.

[8] Earthquake Early Warning Starts Nationwide in Japan. "Earthquake Early Warning Starts Nationwide in Japan". In: *EOS, Transactions, American Geophysical Union* 89.8 (2008). Accessed: 2024-12-15, pp. 73–80. DOI: https://doi.org/10.1029/2008EO080001.

[9] J. Schmidt. *Early earthquake warning systems typically use seismometers to detect incoming fast P-waves*. Credit: NoPineapplesOnPizza (CC BY-SA 4.0) - Temblor.net. June 2020. URL: https://temblor.

net/earthquake-insights/challenges-of-earthquake-early-warning-11099/attachment/earthquake-early-warning-japan/.

[10] A. Karaci. "IoT Temelli Deprem Uyarı Sistemi Geliştirilmesi ve Değerlendirilmesi". In: *Mugla Journal of Science and Technology* 4.2 (2018), pp. 156–161. DOI: 10.22531/muglajsci.442492.

[11] N. Pandey et al. "Earthquake Alert System Based on IoT Technology". In: *JETIR* 8.5 (2021), pp. 1–6. ISSN: 2349-5162. URL: https://www.jetir.org/papers/JETIR2105384.pdf.

[12] Yusuf Perwej et al. "A System for Monitoring Seismic Activity Based on Internet of Things (IoT)". In: 11.7(1) (2022), pp. 6–12. ISSN: 2277-7881. URL: https://www.researchgate.net/publication/362540268_A_System_for_Monitoring_Seismic_Activity_Based_on_Internet_of_Things_IoT.

[13] U.S. Geological Survey. *Early Warning*. Accessed: December 1, 2024. Mar. 2022. URL: https://www.usgs.gov/programs/earthquake-hazards/science/early-warning.

[14] G. Ben, S. Thomas, and S. Philip. "Earthquake Detection Using GSM and Monitoring". In: *IJRTI* 2.4 (2017), pp. 16–17. ISSN: 2456-3315. URL: https://www.ijrti.org/papers/IJRTI1704005.pdf.

[15] Rahadian Irvan et al. "Early Warning System in Mobile-Based Impacted Areas". In: *International Journal of Engineering and Technology* 7 (Aug. 2018), pp. 118–121. DOI: 10.14419/ijet.v7i3.4.16758.

[16] Irshad Khan, Seonhwa Choi, and Young-Woo Kwon. "Earthquake Detection in a Static and Dynamic Environment Using Supervised Machine Learning and a Novel Feature Extraction Method". In: *Sensors* 20.3 (2020). ISSN: 1424-8220. DOI: 10.3390/s20030800. URL: https://www.mdpi.com/1424-8220/20/3/800.

[17] IndiaAI. *Seismic shift: AI-driven earthquake prediction for a safer future in 2025*. Accessed: 2025-05-03. 2025. URL: https://indiaai.gov.in/article/seismic-shift-ai-driven-earthquake-prediction-for-a-safer-future-in-2025.

[18] B. Palombo and N. Pino. "On the recovery and analysis of historical seismograms". In: *Annals of Geophysics* RV0326 (2013). DOI: 10.4401/ag-6270. URL: https://www.earth-prints.org/server/api/core/bitstreams/b210efd7-29fa-46f2-a071-9c128d468526/content.

[19] PCB Piezotronics. *Introduction to MEMS Accelerometers*. Accessed: December 1, 2024. n.d. URL: https://www.pcb.com/resources/technical-information/mems-accelerometers.

[20] Hervé Abdi. "Singular Value Decomposition (SVD) and Generalized Singular Value Decomposition (GSVD)". In: *Encyclopedia of Measurement and Statistics.* (Jan. 2007), pp. 1–14. URL: https://personal.utdallas.edu/~herve/Abdi-SVD2007-pretty.pdf.

[21] Enrico Bassetti and Emanuele Panizzi. "Earthquake Detection at the Edge: IoT Crowdsensing Network". In: *Information* 13.4 (2022), pp. 1–15. ISSN: 2078-2489. DOI: 10.3390/info13040195. URL: https://www.mdpi.com/2078-2489/13/4/195.

[22] STMicroelectronics. *STM32F103x8/xB Microcontroller Datasheet*. Technical specifications and features of STM32F103 microcontrollers. STMicroelectronics. Sept. 2023. URL: https://www.st.com/resource/en/datasheet/stm32f103c8.pdf.

[23] Shi-Tao Yeh et al. "Using trapezoidal rule for the area under a curve calculation". In: *Proceedings of the 27th Annual SAS® User Group International (SUGI'02)* (2002), pp. 1–5. URL: https://www.lexjansen.com/nesug/nesug02/ps/ps017.pdf.

[24] InvenSense, Inc. *MPU-6050 Datasheet*. Accessed: October 11, 2024. 2013. URL: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf.

[25] Peter Bormann. "Magnitude calibration formulas and tables, comments on their use and complementary data". In: *New Manual of Seismological Observatory Practice 2 (NMSOP-2)*. Ed. by Peter Bormann. Deutsches GeoForschungsZentrum GFZ, 2012, pp. 1–19. DOI: 10.2312/GFZ.NMSOP-2_DS_3.1.

[26] Calculator Academy. *Epicenter Distance Calculator*. Accessed: 2025-01-18. Aug. 2024. URL: https://calculator.academy/epicenter-distance-calculator/.

[27] Y. Wu. "An improved STA/LTA algorithm for P wave detection in the 2015 Nepal Earthquake". In: *Dean&Francis* 1.4 (2024). DOI: 10.61173/c2xxms51.

[28] Tong Shen et al. "A first arrival picking method of microseismic data based on single time window with window length independent". In: *Journal of Seismology* 22 (Nov. 2018). DOI: 10.1007/s10950-018-9789-y.

[29] Seismology Research Centre. *What is an Earthquake?* Accessed: 2024-12-15. June 2017. URL: https://www.src.com.au/earthquakes/seismology-101/what-is-an-earthquake/.

[30] T. Oleson. "Well-healed faults produce high-frequency earthquake waves". In: *Earth Magazine* (Feb. 2013). Accessed: 2024-12-18. URL: https://www.earthmagazine.org/article/well-healed-faults-produce-high-frequency-earthquake-waves/#:~:text=Earthquake%2Dinduced%20seismic%20wave%20frequencies,close%20to%20the%20earthquake%20source.

[31] Manabu Kobayashi, Shunsuke Takemura, and Kazuo Yoshimoto. "Frequency and distance changes in the apparent P-wave radiation pattern: effects of seismic wave scattering in the crust inferred from dense seismic observations and numerical simulations". In: *Geophysical Journal International* 202.3 (July 2015), pp. 1895–1907. ISSN: 0956-540X. DOI: 10.1093/gji/ggv263. eprint: https://academic.oup.com/gji/article-pdf/202/3/1895/1691139/ggv263.pdf. URL: https://doi.org/10.1093/gji/ggv263.

[32] Meinard Müller. *The Fourier Transform in a Nutshell*. Retrieved December 10, 2024. Springer, 2015. ISBN: 978-3-319-21944-8. URL: https://www.researchgate.net/publication/290440858_The_Fourier_Transform_in_a_Nutshell.

[33] James C. Schatzman. "Accuracy of the Discrete Fourier Transform and the Fast Fourier Transform". In: *SIAM Journal on Scientific Computing* 17.5 (1996), pp. 1150–1166. DOI: 10.1137/S1064827593247023. eprint: https://doi.org/10.1137/S1064827593247023. URL: https://doi.org/10.1137/S1064827593247023.

[34] COSMOS. *STRONG-MOTION VIRTUAL DATA CENTER (VDC)*. URL: https://www.strongmotioncenter.org/vdc/scripts/default.plx.

[35] Geoffrey Webb. "Naïve Bayes". In: Jan. 2016, pp. 1–2. ISBN: 9781489975027. DOI: 10.1007/978-1-4899-7502-7_581-1.

[36] Alexey Natekin and Alois Knoll. "Gradient boosting machines, a tutorial". In: *Frontiers in Neurorobotics* 7 (2013). ISSN: 1662-5218. DOI: 10.3389/fnbot.2013.00021. URL: https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2013.00021.

[37] Guolin Ke et al. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa Paper.pdf.

[38] Aakash Parmar, Rakesh Katariya, and Vatsal Patel. "A Review on Random Forest: An Ensemble Classifier". In: *International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018*. Ed. by Jude Hemanth et al. Cham: Springer International Publishing, 2019, pp. 758–763. ISBN: 978-3-030-03146-6. URL: https://link.springer.com/chapter/10.1007/978-3-030-03146-6_86.

[39] Rachel M. Adams et al. "ShakeAlert® and schools: Incorporating earthquake early warning in school districts in Alaska, California, Oregon, and Washington". In: *International Journal of Disaster Risk Reduction* 112 (2024), p. 104735. ISSN: 2212-4209. DOI: https://doi.org/10.1016/j.ijdrr.2024.104735. URL: https://www.sciencedirect.com/science/article/pii/S2212420924004977.

[40] Canadian Hazards Information Service Government of Canada Natural Resources Canada. *Canadian Earthquake Early Warning*. Accessed: 2024-12-15. Oct. 2024. URL: https://www.earthquakescanada.nrcan.gc.ca/eew-asp/system-en.php.

[41] Richard Allen and Hiroo Kanamori. "The Potential for Earthquake Early Warning in Southern California". In: *Science (New York, N.Y.)* 300 (June 2003), pp. 786–9. DOI: 10.1126/science.1080912.

[42] A. Pepe and F. Calò. “A Review of Interferometric Synthetic Aperture RADAR (InSAR) Multi-Track Approaches for the Retrieval of Earth’s Surface Displacements”. In: *Applied Sciences* 7.12 (2017), p. 1264. DOI: 10.3390/app7121264. URL: https://doi.org/10.3390/app7121264.

[43] Paola Pierleoni et al. “Internet of Things for Earthquake Early Warning Systems: A Performance Comparison Between Communication Protocols”. In: *IEEE Access* 11 (2023), pp. 43183–43194. DOI: 10.1109/ACCESS.2023.3271773.

[44] Jialin Song and Hui Meng*. “Research and Teaching Practice on Experiment Course of STM32-Embedded Microcontroller”. In: *Proceedings of the 2019 3rd International Conference on Education, Economics and Management Research (ICEEMR 2019)*. Atlantis Press, 2020, pp. 342–345. ISBN: 978-94-6252-871-0. DOI: 10.2991/assehr.k.191221.081. URL: https://doi.org/10.2991/assehr.k.191221.081.

[45] Ferran Adelantado et al. “Understanding the Limits of LoRaWAN”. In: *IEEE Communications Magazine* 55.9 (2017), pp. 34–40. ISSN: 0163-6804. DOI: 10.1109/mcom.2017.1600613. URL: https://doi.org/10.1109/mcom.2017.1600613.

[46] Jonathan Ott et al. “Multipath Delay Estimation in Complex Environments using Transformer”. In: Sept. 2023, pp. 1–6. DOI: 10.1109/IPIN57070.2023.10332470.

[47] National Telecommunications Regulatory Authority (NTRA). *Radio Spectrum Guidelines (Issue No. 1.0)*. Retrieved December 15, 2024. 2020. URL: https://www.tra.gov.eg/wp-content/uploads/2020/12/Guidelines-for-using-short-range-devices-srd-in-egypt.pdf.

[48] Circuits4you. *ESP8266 Absolute Maximum Ratings*. Retrieved December 15, 2024. 2019. URL: https://circuits4you.com/2019/01/13/esp8266-absolute-maximum-ratings/.

[49] A. Kraiem, S. Saka, and S. Kadirova. “Analysis of Electronic Systems for Control of Smart Greenhouses”. In: *University of Ruse* 61.3.1 (2022), pp. 99–105. URL: https://conf.uni-ruse.bg/bg/docs/cp22/3.1/3.1-14.pdf.

[50] David J Wald. “Practical limitations of earthquake early warning”. In: *Earthquake Spectra* 36.3 (Aug. 2020), pp. 1412–1447. ISSN: 8755-2930. DOI: 10.1177/8755293020911388. eprint: https://pubs.geoscienceworld.org/eeri/earthquake-spectra/article-pdf/36/3/1412/7185669/10.1177_8755293020911388.pdf. URL: https://doi.org/10.1177/8755293020911388.

[51] D. Dhivya and K. Nirmala. “Study on Integration Testing and System Testing”. In: *International Journal of Creative Research Thoughts (IJCRT)* 6.2 (Apr. 2018), pp. 794–798. ISSN: 2320-2882. URL: http://www.ijcrt.org/papers/IJCRT1812329.pdf.

[52] M. Naseet. *Arduino Earthquake Simulator - My Project Ideas*. https://myprojectideas.com/arduino-earthquake-simulator/. Accessed: 2025-05-19. Mar. 2023.

[53] Thijmen. *What are bass shakers and why are they so popular?* Accessed: 2025-05-19. June 24, 2024. URL: https://www.soundimports.eu/en/blogs/blog/what-are-bass-shakers-and-why-are-they-so-popular/.

[54] *Amazon EC2 On-Demand Pricing*. Accessed: 7 October 2024. 2024. URL: https://aws.amazon.com/ec2/pricing/on-demand/.

[55] Upwork. *How Much Does a Domain Name Really Cost in 2024? (A Complete Breakdown)*. Accessed: December 1, 2024. May 2021. URL: https://www.upwork.com/resources/how-much-does-a-domain-name-cost.

[56] K. Ramachandran and K. Karthick. "Gantt Chart: An Important Tool of Management". In: *International Journal of Innovative Technology and Exploring Engineering* 8.7C (2019), pp. 2278–3075. URL: https://www.researchgate.net/profile/Ramachandran-K-K-3/publication/358234055_Gantt_Chart_An_Important_Tool_of_Management/links/61f78cc7007fb5044727d77d/Gantt-Chart-An-Important-Tool-of-Management.pdf.

Appendix A

System code

0.1 translator.py

```python
import requests
import json
import time
import math
import socket
import numpy as np
import websocket
import matplotlib.pyplot as plt
from collections import deque
import time
import pickle

with open('random_forest_200.pkl', 'rb') as model_file:
    ml_model = pickle.load(model_file)

UDP_IP = "0.0.0.0"  # Listen on all interfaces
UDP_PORT = 5005     # Listening port

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((UDP_IP, UDP_PORT))
sock.settimeout(2.0)  # Set timeout to prevent blocking forever
last_richter=0
def predict_with_model(acceleration_data):
    # You may need to preprocess or reshape the data depending on your model
    # Example: flatten and wrap in list to simulate a 2D array for scikit-learn
    features = [acceleration_data]  # shape: (1, N) if needed
    prediction = ml_model.predict(features)
```

```
    return prediction[0] == 1

def compute_fft_peak_frequency(acceleration_data, fs):
    # Number of data points
    n = len(acceleration_data)

    if n == 0:
        raise ValueError("Error: The acceleration data array is empty.")

    # Compute FFT
    fft_result = np.fft.fft(acceleration_data)
    frequencies = np.fft.fftfreq(n, d=1/fs)

    # Take only positive frequencies
    pos_mask = frequencies > 0
    frequencies = frequencies[pos_mask]
    fft_magnitude = np.abs(fft_result[pos_mask])

    # Find peak frequency
    peak_index = np.argmax(fft_magnitude)
    peak_frequency = frequencies[peak_index]
    '''
    plt.figure(figsize=(8, 4))
    plt.plot(frequencies, fft_magnitude, label="FFT Magnitude")
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Magnitude")
    plt.title("FFT Spectrum of Acceleration Data")
    plt.grid()
    plt.legend()
    plt.show()'''
    #if (peak_frequency>0.001 and peak_frequency<0.1) or (peak_frequency>1 and peak_frequency<20):
    if (peak_frequency>0.001 and peak_frequency<0.1)  :
        return True
    return False

def send_message(message, target_ip, target_port):
    """Send a UDP message to the specified target IP and port."""
    try:
        sock.sendto(str(message).encode(), (target_ip, target_port))
        #print(f"Sent to {target_ip}:{target_port} -> {message}")
    except Exception as e:
        print(f"Error sending message: {e}")
```

```python
def movement_detection():
    try:
        data, addr = sock.recvfrom(1024)  # Receive message
        message = data.decode()
        value = float(message.split(":")[1])  # Extract acceleration value
        return value, addr
    except socket.timeout:
        return None, None
    except (ValueError, IndexError):
        return None, None

class EarthquakeProcessor:
    def __init__(self, station_id=1):
        self.station_id = station_id
        self.session = requests.Session()

        # Constants
        self.STA_WINDOW = 10
        self.LTA_WINDOW = 50
        #the more the diff between sta and lta the more the accuracy but slower to calc
            richter
        self.DETECTION_THRESHOLD = 2

        # Initial reset
        self.reset_state()

    def reset_state(self):
        """Full system reset with zero values sent"""
        # Clear all buffers and states
        self.event_start = None
        self.s_wave_detected = False
        self.acceleration_buffer = deque(maxlen=self.LTA_WINDOW)

        # Reset physical parameters to match original code
        self.initial_velocity = 0
        self.initial_displacement = 0
        self.total_displacement = 0
        self.max_displacement = 0
        self.max_richter = 0

        # Reset S-wave parameters
        self.s_wave_time = 0
        self.ed = 0
        self.a0_correction = 0
```

```python

        # Send zero values immediately
        self.send_zero_values()

    def send_zero_values(self):
        """Send zero-values using the persistent WebSocket connection"""

        if not hasattr(self, 'ws') or self.ws is None:
            import websocket  # Import inside to avoid unnecessary dependency if not
                used
            self.ws = websocket.WebSocket()
            self.ws.connect("ws://localhost:3001")  # Connect only once

        zero_data = {
            "type": "addEvent",
            "velocity": 0,
            "displacement": 0,
            "richter": 0,
            "acceleration": 0,
            "station_id": self.station_id
        }

        try:
            self.ws.send(json.dumps(zero_data))  # Send data over WebSocket
            print("Zero-values sent successfully via WebSocket")



        except Exception as e:
            print(f"Error sending zero-values via WebSocket: {e}")

    def fetch_data(self):

        """Fetch acceleration data from UDP, extract station_id and acceleration, and
            return as JSON."""

        try:
            data, addr = sock.recvfrom(1024)  # Receive message
            message = data.decode().strip()

            # Expecting format: "Station X: Y.YYYYYYYY"
            parts = message.split(":")

            station_part = parts[0].strip()  # "Station 1"
```

```python
            acceleration_part = parts[1].strip()  # "0.00000000"

            # Extract station number

            station_id = station_part.split()[1]  # Extract "1" from "Station 1"

            # Convert acceleration to float
            acceleration = float(acceleration_part)

            return {
                "station_id": station_id,
                "acceleration": acceleration
            }


        except socket.timeout:
                print("timeout")
                return None  # Timeout case
        # except (ValueError, IndexError):
        #     return None  #

    def send_data(self,result):
        """Send event data using a synchronous WebSocket connection"""
        ws = websocket.WebSocket()
        ws.connect("ws://localhost:3001")  # Connect to WebSocket server

        event_data = {
            "type": "addEvent",
            "velocity": result['velocity'],
            "displacement": result['displacement'],
            "richter": result['richter'],
            "acceleration": result['acceleration'],
            "station_id": self.station_id
        }
        #time.sleep(0.1)#delay to make it readable
        ws.send(json.dumps(event_data))  # Send event data







    def trapezoidal_integration(self, acceleration, elapsed_time):
```

```
        """EXACT replica of original integration method"""
        # Original calculation
        velocity = self.initial_velocity + 0.5 * elapsed_time * acceleration
        displacement = self.initial_displacement + 0.5 * (
            velocity + self.initial_velocity
        ) * elapsed_time

        # Original delta calculation
        delta_v = velocity - self.initial_velocity
        delta_d = displacement - self.initial_displacement
        return delta_v, delta_d,displacement

    def detect_S_wave(self):
        """Detect S-wave using STA/LTA ratio"""
        if len(self.acceleration_buffer) < self.LTA_WINDOW:
            return None

        sta = sum(abs(x) for x in list(self.acceleration_buffer)[-self.STA_WINDOW:]) / self.STA_WINDOW
        lta = sum(abs(x) for x in list(self.acceleration_buffer)[-self.LTA_WINDOW:]) / self.LTA_WINDOW

        return sta / lta if lta != 0 else 0

    def calculate_epicenter(self, s_wave_delay,current_time):
        """Calculate epicenter distance and A   correction"""
        if not s_wave_delay or s_wave_delay <= 0:
            return 0.0, 0.0

        ed = 8.4* (s_wave_delay)
        x = ed
        a0_correction = -1.60775e-10 * x**4 + 2.1429e-7 * x**3 - 0.000100878 * x**2 + 0.023678 * x + 1.45704
        print(x,a0_correction )
        return ed, a0_correction

    def estimate_richter(self, total_displacement):
        """Estimate Richter magnitude (updated)"""
        global last_richter
        if not self.s_wave_detected or self.a0_correction == 0:
            return 0  # Don't calculate before S-wave detection

        if total_displacement > 0:
            richter = math.log10(total_displacement)+self.a0_correction
```

```
            if richter<last_richter:
                return last_richter
            last_richter=richter
            return min(max(richter, 0), 9.5)
        return 0

    def main_loop(self):
        """Main processing loop matching original code flow"""
        while True:
            try:
                # Fetch new data
                data = self.fetch_data()
                if not data:  # If fetch_data() returns None, skip iteration
                    print("No valid data received. Skipping...")

                    self.reset_state()
                    break

                acceleration = abs(data['acceleration'])   # Convert to m/s



                current_time = time.time()

                # First-time event setup
                if not self.event_start:
                    self.event_start = current_time
                    self.last_integration_time = current_time
                    print("New event detected - starting measurements")
                    continue  # Skip first frame

                # Compute time elapsed
                elapsed_time = current_time - self.last_integration_time
                if elapsed_time <= 0:
                    continue

                # Perform integration
                delta_v, delta_d, temp = self.trapezoidal_integration(acceleration,
                    elapsed_time)
                self.total_displacement += delta_d
                self.max_displacement = max(self.max_displacement, delta_d)

                # Update initial values
                self.initial_velocity = delta_v
```

```
                self.initial_displacement = delta_d
                self.last_integration_time = current_time

                # S-wave detection
                self.acceleration_buffer.append(acceleration)
                ratio = self.detect_S_wave()
                if ratio and ratio > self.DETECTION_THRESHOLD:
                    self.s_wave_detected = True
                    self.s_wave_time = current_time - self.event_start
                    self.ed, self.a0_correction = self.calculate_epicenter(self.s_wave_time, current_time)
                    print(f"S-wave detected at {self.s_wave_time:.2f}s")

                # Prepare and send results
                result = {
                    'velocity': self.initial_velocity,
                    'displacement': self.initial_displacement,
                    'richter': self.estimate_richter(self.total_displacement),
                    'acceleration': acceleration,
                    'epicenter_distance': self.ed,
                    'a0_correction': self.a0_correction
                }

                #print("Sending result:", result)  # Debug output
                self.send_data(result)

            except KeyboardInterrupt:
                print("Shutting down...")
                break
            except Exception as e:
                print(f"Error: {e}")  # Print the actual error message
                self.reset_state()
                time.sleep(1)

if __name__ == "__main__":
    fs =200  # Sampling frequency (Hz)

    while True:
        station_id=0
        acceleration_data = []
        sender_address = None

        # Collect 200 acceleration samples
        while len(acceleration_data) < fs:
```

```python
            acc, addr = movement_detection()
            if acc is not None:
                acceleration_data.append(acc)
                sender_address = addr  # Store sender's address for response
            else:
                print("no possibility for earthquake")
                acceleration_data=[]

        if sender_address:
            if predict_with_model(acceleration_data): #ml model first
                is_earthquake = compute_fft_peak_frequency(acceleration_data, fs)
                if not is_earthquake:
                    print(acceleration_data[0])
                    send_message(0, sender_address[0], sender_address[1])  # Send response
                print(f"Is earthquake?: {is_earthquake}")
                if is_earthquake:
                    time.sleep(1)
                    last_richter=0
                    earthquake_processor = EarthquakeProcessor()
                    earthquake_processor.main_loop()
```

0.2 READINGS_ONLY.py

```python
import socket

UDP_IP = "0.0.0.0"  # Listen on all interfaces
UDP_PORT = 5005     # Listening port

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((UDP_IP, UDP_PORT))



counter=0
def receive_message():
    global counter
    data, addr = sock.recvfrom(1024)  # Receive message
    print(f"Received from {addr[0]}:{addr[1]} -> {data.decode()}")
    counter+=1
    print (counter)
    return data.decode(), addr

```

```
while True:
    msg, sender = receive_message()  # Receive a message
```

0.3 docker-compose.yml

```
version: '3.8'

networks:
  app-network:
    driver: bridge

services:
  mysql:
    image: mysql:5.7
    container_name: stationsql
    environment:
      - MYSQL_ROOT_PASSWORD=  # Empty password for MySQL
      - MYSQL_ALLOW_EMPTY_PASSWORD=true
      - MYSQL_DATABASE=station
    ports:
      - "3006:3306"
    volumes:
      - "./database_init:/docker-entrypoint-initdb.d"
      - "./savedata:/var/lib/mysql"
    networks:
      - app-network

  frontend:
    container_name: station_frontend
    build: ./frontend
    ports:
      - "3000:3000"
    networks:
      - app-network

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    container_name: phpmyadmin
    environment:
      - PMA_HOST=mysql  # Use the MySQL service name as host
      - PMA_PORT=3306
```

```
      - PMA_USER=root
      - PMA_PASSWORD=
    ports:
      - "3002:80"
    depends_on:
      - mysql
    networks:
      - app-network

  backend:
    container_name: station_backend
    build: ./backend
    ports:
      - "3001:3001"
    environment:
      - MYSQL_HOST=mysql  # Use the name of the MySQL service here
      - MYSQL_PORT=3306
      - MYSQL_USER=root
      - MYSQL_PASSWORD=  # Empty password as per your settings
      - MYSQL_DATABASE=station
    depends_on:
      - mysql
    networks:
      - app-network
    command: ["sh", "-c", "/usr/local/bin/wait-for-it mysql:3306 -- node index.js"]
    restart: always
```

0.4 Station Side Code

0.4.1 ESP01.h

```
#ifndef ESP01_H
#define ESP01_H

#include "main.h"  // Change this based on your STM32 series
#include <stdbool.h>

#include <ESP01_SECRET_KEYS.h>

#define UDP_TARGET_IP   "192.168.1.108"
#define UDP_TARGET_PORT 5005  // Define the UDP port number
#define RST_PORT    GPIOA
```

```
#define RST_PIN     GPIO_PIN_0
extern UART_HandleTypeDef huart1;  // Use UART1

// Function prototypes
void ESP_Init(void);
void ESP_Send_Data(float data, int Station_ID);
void ESP_HARD_RESET(void);
HAL_StatusTypeDef WaitForResponse(const char *expected, uint32_t timeout);

#endif // __ESP01_H
```

0.4.2 ESP_01_SECRET_KEYS.h

```
#define WIFI_SSID     "write_your_ssid_here"
#define WIFI_PASSWORD   "write_your_password_here"
```

0.4.3 MPU6050.h

```
#ifndef MPU6050_H
#define MPU6050_H

#include "stm32f1xx_hal.h"  // Adjust for your MCU family if needed
#include <stdint.h>

// MPU6050 I2C Address
#define MPU6050_ADDR 0x68 << 1  // Shifted for HAL (7-bit address is 0x68)

// MPU6050 Registers
#define WHO_AM_I_REG     0x75
#define PWR_MGMT_1       0x6B

#define ACCEL_CONFIG   0x1C
#define ACCEL_XOUT_H     0x3B


#define MPU6050_ACCEL_SENS_2G  16384.0  // LSB/g for  2g  full scale
#define MPU6050_ACCEL_SENS_4G  8192.0
#define MPU6050_ACCEL_SENS_8G  4096.0
#define MPU6050_ACCEL_SENS_16G  2048.0
// Struct to store acceleration values
typedef struct {
    float Ax;
```

```
    float Ay;
    float Az;
} MPU6050_Data_t;

// Function prototypes
HAL_StatusTypeDef MPU6050_Init(I2C_HandleTypeDef *hi2c);
float MPU6050_Read_Accel(I2C_HandleTypeDef *hi2c);

#endif
```

0.4.4 fonts.h

```
/* vim: set ai et ts=4 sw=4: */
#ifndef __FONTS_H__
#define __FONTS_H__

#include <stdint.h>

typedef struct {
    const uint8_t width;
    uint8_t height;
    const uint16_t *data;
} FontDef;

extern FontDef Font_5x8;

#endif // __FONTS_H__
```

0.4.5 main.h

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.h
  * @brief          : Header for main.c file.
  *                   This file contains the common defines of the application.
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.
  *
```

```c
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f1xx_hal.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */

/* USER CODE END ET */

/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */

/* USER CODE END EC */

/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */

/* USER CODE END EM */

/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */

```

```c
/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/

/* USER CODE BEGIN Private defines */

/* USER CODE END Private defines */

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */
```

0.4.6 st7735.h

```c
/* vim: set ai et ts=4 sw=4: */
#ifndef __ST7735_H__
#define __ST7735_H__

#include "fonts.h"
#include <stdbool.h>



#define STATION_Y 15
#define ESP_Y   55
#define ACCEL_Y   120

#define ST7735_MADCTL_MY  0x80
#define ST7735_MADCTL_MX  0x40
#define ST7735_MADCTL_MV  0x20
#define ST7735_MADCTL_ML  0x10
#define ST7735_MADCTL_RGB 0x00
#define ST7735_MADCTL_BGR 0x08
#define ST7735_MADCTL_MH  0x04

/*** Redefine if necessary ***/
#define ST7735_SPI_PORT hspi1
extern SPI_HandleTypeDef ST7735_SPI_PORT;

#define ST7735_RES_Pin       GPIO_PIN_1
#define ST7735_RES_GPIO_Port GPIOB
```

```c
#define ST7735_CS_Pin        GPIO_PIN_10
#define ST7735_CS_GPIO_Port  GPIOB
#define ST7735_DC_Pin        GPIO_PIN_0
#define ST7735_DC_GPIO_Port  GPIOB

// AliExpress/eBay 1.8" display, default orientation
/*
#define ST7735_IS_160X128 1
#define ST7735_WIDTH  128
#define ST7735_HEIGHT 160
#define ST7735_XSTART 0
#define ST7735_YSTART 0
#define ST7735_ROTATION (ST7735_MADCTL_MX | ST7735_MADCTL_MY)
*/

// AliExpress/eBay 1.8" display, rotate right
/*
#define ST7735_IS_160X128 1
#define ST7735_WIDTH  160
#define ST7735_HEIGHT 128
#define ST7735_XSTART 0
#define ST7735_YSTART 0
#define ST7735_ROTATION (ST7735_MADCTL_MY | ST7735_MADCTL_MV)
*/

// AliExpress/eBay 1.8" display, rotate left
/*
#define ST7735_IS_160X128 1
#define ST7735_WIDTH  160
#define ST7735_HEIGHT 128
#define ST7735_XSTART 0
#define ST7735_YSTART 0
#define ST7735_ROTATION (ST7735_MADCTL_MX | ST7735_MADCTL_MV)
*/

// AliExpress/eBay 1.8" display, upside down
/*
#define ST7735_IS_160X128 1
#define ST7735_WIDTH  128
#define ST7735_HEIGHT 160
#define ST7735_XSTART 0
#define ST7735_YSTART 0
#define ST7735_ROTATION (0)
*/
```

```

// WaveShare ST7735S-based 1.8" display, default orientation
/*
#define ST7735_IS_160X128 1
#define ST7735_WIDTH  128
#define ST7735_HEIGHT 160
#define ST7735_XSTART 2
#define ST7735_YSTART 1
#define ST7735_ROTATION (ST7735_MADCTL_MX | ST7735_MADCTL_MY | ST7735_MADCTL_RGB)
*/

// WaveShare ST7735S-based 1.8" display, rotate right
/*
#define ST7735_IS_160X128 1
#define ST7735_WIDTH  160
#define ST7735_HEIGHT 128
#define ST7735_XSTART 1
#define ST7735_YSTART 2
#define ST7735_ROTATION (ST7735_MADCTL_MY | ST7735_MADCTL_MV | ST7735_MADCTL_RGB)
*/

// WaveShare ST7735S-based 1.8" display, rotate left
/*
#define ST7735_IS_160X128 1
#define ST7735_WIDTH  160
#define ST7735_HEIGHT 128
#define ST7735_XSTART 1
#define ST7735_YSTART 2
#define ST7735_ROTATION (ST7735_MADCTL_MX | ST7735_MADCTL_MV | ST7735_MADCTL_RGB)
*/

// WaveShare ST7735S-based 1.8" display, upside down
/*
#define ST7735_IS_160X128 1
#define ST7735_WIDTH  128
#define ST7735_HEIGHT 160
#define ST7735_XSTART 2
#define ST7735_YSTART 1
#define ST7735_ROTATION (ST7735_MADCTL_RGB)
*/



//works fine ---->
```

```
 //1.44" display, default orientation

#define ST7735_IS_160X128 1
#define ST7735_WIDTH  128
#define ST7735_HEIGHT 160
#define ST7735_XSTART 0
#define ST7735_YSTART 0
#define ST7735_ROTATION (ST7735_MADCTL_MX | ST7735_MADCTL_MY )

// 1.44" display, rotate right
/*
#define ST7735_IS_128X128 1
#define ST7735_WIDTH  160
#define ST7735_HEIGHT 128
#define ST7735_XSTART 0
#define ST7735_YSTART 0
#define ST7735_ROTATION (ST7735_MADCTL_MY | ST7735_MADCTL_MV | ST7735_MADCTL_BGR)
*/

// 1.44" display, rotate left
/*
#define ST7735_IS_128X128 1
#define ST7735_WIDTH  128
#define ST7735_HEIGHT 128
#define ST7735_XSTART 1
#define ST7735_YSTART 2
#define ST7735_ROTATION (ST7735_MADCTL_MX | ST7735_MADCTL_MV | ST7735_MADCTL_BGR)
*/

// 1.44" display, upside down
/*
#define ST7735_IS_128X128 1
#define ST7735_WIDTH  128
#define ST7735_HEIGHT 128
#define ST7735_XSTART 2
#define ST7735_YSTART 1
#define ST7735_ROTATION (ST7735_MADCTL_BGR)
*/

// mini 160x80 display (it's unlikely you want the default orientation)
/*
#define ST7735_IS_160X80 1
#define ST7735_XSTART 26
#define ST7735_YSTART 1
```

```
#define ST7735_WIDTH  80
#define ST7735_HEIGHT 160
#define ST7735_ROTATION (ST7735_MADCTL_MX | ST7735_MADCTL_MY | ST7735_MADCTL_BGR)
*/

// mini 160x80, rotate left
/*
#define ST7735_IS_160X80 1
#define ST7735_XSTART 1
#define ST7735_YSTART 26
#define ST7735_WIDTH  160
#define ST7735_HEIGHT 80
#define ST7735_ROTATION (ST7735_MADCTL_MX | ST7735_MADCTL_MV | ST7735_MADCTL_BGR)
*/

// mini 160x80, rotate right
/*
#define ST7735_IS_160X80 1
#define ST7735_XSTART 1
#define ST7735_YSTART 26
#define ST7735_WIDTH  160
#define ST7735_HEIGHT 80
#define ST7735_ROTATION (ST7735_MADCTL_MY | ST7735_MADCTL_MV | ST7735_MADCTL_BGR)
*/

/****************************/

#define ST7735_NOP     0x00
#define ST7735_SWRESET 0x01
#define ST7735_RDDID   0x04
#define ST7735_RDDST   0x09

#define ST7735_SLPIN   0x10
#define ST7735_SLPOUT  0x11
#define ST7735_PTLON   0x12
#define ST7735_NORON   0x13

#define ST7735_INVOFF  0x20
#define ST7735_INVON   0x21
#define ST7735_GAMSET  0x26
#define ST7735_DISPOFF 0x28
#define ST7735_DISPON  0x29
#define ST7735_CASET   0x2A
#define ST7735_RASET   0x2B
```

```c
#define ST7735_RAMWR   0x2C
#define ST7735_RAMRD   0x2E

#define ST7735_PTLAR   0x30
#define ST7735_COLMOD  0x3A
#define ST7735_MADCTL  0x36

#define ST7735_FRMCTR1 0xB1
#define ST7735_FRMCTR2 0xB2
#define ST7735_FRMCTR3 0xB3
#define ST7735_INVCTR  0xB4
#define ST7735_DISSET5 0xB6

#define ST7735_PWCTR1  0xC0
#define ST7735_PWCTR2  0xC1
#define ST7735_PWCTR3  0xC2
#define ST7735_PWCTR4  0xC3
#define ST7735_PWCTR5  0xC4
#define ST7735_VMCTR1  0xC5

#define ST7735_RDID1   0xDA
#define ST7735_RDID2   0xDB
#define ST7735_RDID3   0xDC
#define ST7735_RDID4   0xDD

#define ST7735_PWCTR6  0xFC

#define ST7735_GMCTRP1 0xE0
#define ST7735_GMCTRN1 0xE1

// Color definitions
#define ST7735_BLACK   0x0000
#define ST7735_BLUE    0x001F
#define ST7735_RED     0xF800
#define ST7735_GREEN   0x07E0
#define ST7735_CYAN    0x07FF
#define ST7735_MAGENTA 0xF81F
#define ST7735_YELLOW  0xFFE0
#define ST7735_WHITE   0xFFFF
#define ST7735_COLOR565(r, g, b) (((r & 0xF8) << 8) | ((g & 0xFC) << 3) | ((b & 0xF8) >>
    3))

typedef enum {
  GAMMA_10 = 0x01,
```

```
  GAMMA_25 = 0x02,
  GAMMA_22 = 0x04,
  GAMMA_18 = 0x08
} GammaDef;

#ifdef __cplusplus
extern "C" {
#endif

// call before initializing any SPI devices
void ST7735_Unselect();

void ST7735_Init(void);
void ST7735_DrawPixel(uint16_t x, uint16_t y, uint16_t color);
void ST7735_WriteString(uint16_t x, uint16_t y, const char* str, FontDef font, uint16_t
    color, uint16_t bgcolor);
void ST7735_FillRectangle(uint16_t x, uint16_t y, uint16_t w, uint16_t h, uint16_t color
    );
void ST7735_FillRectangleFast(uint16_t x, uint16_t y, uint16_t w, uint16_t h, uint16_t
    color);
void ST7735_FillScreen(uint16_t color);
void ST7735_FillScreenFast(uint16_t color);
void ST7735_InvertColors(bool invert);
void ST7735_SetGamma(GammaDef gamma);

#ifdef __cplusplus
}
#endif

#endif // __ST7735_H__
```

0.4.7 ESP01.c

```
#include "ESP01.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "st7735.h"
#include "fonts.h"
#include "main.h"
extern IWDG_HandleTypeDef hiwdg;
#define RX_BUFFER_SIZE 128

```

```c
// Sends an AT command and waits for the expected response
static HAL_StatusTypeDef SendCommand(const char *cmd, const char *expected, uint32_t
    timeout)
{
    char cmdBuffer[128];
    snprintf(cmdBuffer, sizeof(cmdBuffer), "%s\r\n", cmd);

    // Transmit the command over UART1
    HAL_UART_Transmit(&huart1, (uint8_t*)cmdBuffer, strlen(cmdBuffer), 1000);

    // Wait for the expected response (non-blocking, no retries)
    return WaitForResponse(expected, timeout);
}

// Waits for a response within a given timeout (no retries, returns immediately if no
    response)
HAL_StatusTypeDef WaitForResponse(const char *expected, uint32_t timeout)
{

    uint32_t tickstart = HAL_GetTick();
    uint8_t rxByte;
    char buffer[RX_BUFFER_SIZE];
    uint16_t index = 0;
    memset(buffer, 0, sizeof(buffer));
    while ((HAL_GetTick() - tickstart) < timeout)
    {
        if (HAL_UART_Receive(&huart1, &rxByte, 1, 100) == HAL_OK)
        {
            if (index < RX_BUFFER_SIZE - 1)
            {
                buffer[index++] = rxByte;
                buffer[index] = '\0';
                if (strstr(buffer, expected) != NULL)
                {


                  HAL_IWDG_Refresh(&hiwdg);
                    return HAL_OK;
                }
            }
        }
    }
    //ST7735_FillScreenFast(ST7735_WHITE);
    ST7735_WriteString(1, ESP_Y,buffer, Font_5x8, ST7735_RED, ST7735_BLACK);
```

```
    return HAL_TIMEOUT; // No retries, just return the result
}

// Initializes ESP without retries or reset

void ESP_HARD_RESET(){


  HAL_GPIO_WritePin(RST_PORT, RST_PIN, 0);
  HAL_Delay(50);
  HAL_GPIO_WritePin(RST_PORT, RST_PIN, 1);



}
void ESP_Init(void)
{

      char cmd[128];
      // Check if already connected to Wi-Fi
      ST7735_WriteString(1, ESP_Y,"checking connection", Font_5x8, ST7735_GREEN,
          ST7735_BLACK);

      if (SendCommand("AT+CWJAP?", "No AP", 500)!= HAL_OK){

            snprintf(cmd, sizeof(cmd), "AT+CIPSTART=\"UDP\",\"%s\",%d", UDP_TARGET_IP,
                UDP_TARGET_PORT);

            SendCommand(cmd, "OK", 1500);
            if (SendCommand("AT", "OK", 100)==HAL_OK){
              ST7735_FillRectangleFast(0, ESP_Y, 128,45, ST7735_BLACK);
              ST7735_WriteString(20,ESP_Y,"ESP IS WORKING...", Font_5x8, ST7735_GREEN,
                  ST7735_BLACK);
              return;
            }
            else{
              ST7735_FillRectangleFast(0, ESP_Y, 128,45, ST7735_BLACK);
              ST7735_WriteString(1,ESP_Y,"ESP ERROR...", Font_5x8, ST7735_WHITE,
                  ST7735_BLACK);
              HAL_Delay(10000);
              NVIC_SystemReset();
            }

```

```c

      }

        // Set Wi-Fi mode to station mode
      ST7735_FillRectangleFast(0, ESP_Y, 128,45, ST7735_BLACK);
      ST7735_WriteString(1,ESP_Y,"CONNECTING...", Font_5x8, ST7735_WHITE, ST7735_BLACK);

        SendCommand("AT+CWMODE=1", "OK", 1500);

        // Attempt to connect
        snprintf(cmd, sizeof(cmd), "AT+CWJAP=\"%s\",\"%s\"", WIFI_SSID, WIFI_PASSWORD);

        SendCommand(cmd, "OK", 15000);

        // Attempt to start UDP connection (one-shot)
        snprintf(cmd, sizeof(cmd), "AT+CIPSTART=\"UDP\",\"%s\",%d", UDP_TARGET_IP,
            UDP_TARGET_PORT);

        SendCommand(cmd, "OK", 3000);

        if (SendCommand("AT", "OK", 100)==HAL_OK){
          ST7735_FillRectangleFast(0, ESP_Y, 128,45, ST7735_BLACK);
          ST7735_WriteString(20,ESP_Y,"ESP IS WORKING...", Font_5x8, ST7735_GREEN,
              ST7735_BLACK);
        }
        else{
          ST7735_FillRectangleFast(0, ESP_Y, 128,45, ST7735_BLACK);
          ST7735_WriteString(1,ESP_Y,"ESP ERROR...", Font_5x8, ST7735_WHITE,
              ST7735_BLACK);
          HAL_Delay(10000);
          NVIC_SystemReset();
        }
}

// Sends real-time data over UDP (no retry, no waiting)
void ESP_Send_Data(float data, int Station_ID)
{
    char payload[64];
    snprintf(payload, sizeof(payload), "Station %d: %.8f", Station_ID, data);
    uint16_t len = strlen(payload);

    // Send AT+CIPSEND command
    char cmd[32];
    snprintf(cmd, sizeof(cmd), "AT+CIPSEND=%d", len);
```

```c

    if (SendCommand(cmd, ">", 500) == HAL_OK)
    {
        // Send the payload immediately
      //ST7735_WriteString(10,ESP_Y+10,payload, Font_5x8, ST7735_GREEN, ST7735_BLACK);
        HAL_UART_Transmit(&huart1, (uint8_t*)payload, len, HAL_MAX_DELAY);
    }
    else{
      ESP_Init();
    }
}
```

0.4.8 MPU6050.c

```c
#include "MPU6050.h"
#include <string.h>
#include <math.h>
 // Function to initialize the MPU6050
HAL_StatusTypeDef MPU6050_Init(I2C_HandleTypeDef *hi2c) {
  volatile uint8_t check, data;

    // Check if MPU6050 is connected
    if (HAL_I2C_Mem_Read(hi2c, MPU6050_ADDR, WHO_AM_I_REG, 1, &check, 1, 1000) != HAL_OK
        ) {
        return HAL_ERROR;
    }

    if (check != 0x68) {
        return HAL_ERROR;  // MPU6050 not found
    }

    // Wake up the MPU6050 (clear sleep mode bit)
    data = 0x00;
    if (HAL_I2C_Mem_Write(hi2c, MPU6050_ADDR, PWR_MGMT_1, 1, &data, 1, 1000) != HAL_OK)
        {
        return HAL_ERROR;
    }

    // Set accelerometer range to  2g
    data = 0x00; // 2g :0x00 , 4g :0x08 , 8g :0x10 , 16g : 0x18
    if (HAL_I2C_Mem_Write(hi2c, MPU6050_ADDR, ACCEL_CONFIG, 1, &data, 1, 1000) != HAL_OK
        ) {
        return HAL_ERROR;
```

```c
    }

    return HAL_OK;
}

float MPU6050_Read_Accel(I2C_HandleTypeDef *hi2c)
{
    uint8_t rawData[6];
    int16_t rawAx, rawAy, rawAz;
    float Ax, Ay, Az;

    // Read accelerometer data
    if (HAL_I2C_Mem_Read(hi2c, MPU6050_ADDR, ACCEL_XOUT_H, 1, rawData, 6, 100) != HAL_OK
        )
    {
        MPU6050_Init(hi2c);
        return -1.0;  // Return error value
    }

    // Convert raw data to acceleration (m/s )
    rawAx = (int16_t)((rawData[0] << 8) | rawData[1]);
    rawAy = (int16_t)((rawData[2] << 8) | rawData[3]);
    rawAz = (int16_t)((rawData[4] << 8) | rawData[5]);

    Ax = ((float)rawAx / MPU6050_ACCEL_SENS_2G) * 9.81f;
    Ay = ((float)rawAy / MPU6050_ACCEL_SENS_2G) * 9.81f;
    Az = ((float)rawAz / MPU6050_ACCEL_SENS_2G) * 9.81f;

    // === Step 1: Compute Gravity Vector Magnitude ===
    float g_magnitude = sqrt(Ax * Ax + Ay * Ay + Az * Az);


    // Normalize gravity vector (unit vector)
    float gX = (Ax / g_magnitude) * 9.81f;
    float gY = (Ay / g_magnitude) * 9.81f;
    float gZ = (Az / g_magnitude) * 9.81f;

    // === Step 2: Remove Gravity from Acceleration ===
    float aX_noG = Ax - gX;
    float aY_noG = Ay - gY;
    float aZ_noG = Az - gZ;

    // Compute magnitude of true motion
    float motionMagnitude = sqrt(aX_noG * aX_noG + aY_noG * aY_noG + aZ_noG * aZ_noG);
```

```
    motionMagnitude -=0.55;
       if (motionMagnitude <0)return 0.0;
       // === Step 3: Apply Strict Noise Filtering ===


    return motionMagnitude;
}
```

0.4.9 fonts.c

```
/* vim: set ai et ts=4 sw=4: */
#include "fonts.h"

static const uint16_t Font5x8[] = {
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, /* SPACE */
    0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x0000, 0x2000, 0x0000, /* ! */
    0x5000, 0x5000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, /* " */
    0x5000, 0x7000, 0x5000, 0x5000, 0x7000, 0x5000, 0x0000, 0x0000, /* # */
    0x2000, 0x2000, 0x5000, 0x2000, 0x1000, 0x5000, 0x2000, 0x2000, /* $ */
    0x5000, 0x1000, 0x2000, 0x2000, 0x4000, 0x5000, 0x0000, 0x0000, /* % */
    0x2000, 0x5000, 0x2000, 0x5000, 0x5000, 0x6000, 0x3000, 0x0000, /* & */
    0x2000, 0x2000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, /* ' */
    0x1000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x1000, 0x0000, /* ( */
    0x4000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x4000, 0x0000, /* ) */
    0x0000, 0x5000, 0x2000, 0x7000, 0x2000, 0x5000, 0x0000, 0x0000, /* * */
    0x0000, 0x2000, 0x2000, 0x7000, 0x2000, 0x2000, 0x0000, 0x0000, /* + */
    0x0000, 0x0000, 0x0000, 0x0000, 0x2000, 0x4000, 0x0000, 0x0000, /* , */
    0x0000, 0x0000, 0x0000, 0x7000, 0x0000, 0x0000, 0x0000, 0x0000, /* - */
    0x0000, 0x0000, 0x0000, 0x0000, 0x2000, 0x2000, 0x0000, 0x0000, /* . */
    0x1000, 0x1000, 0x2000, 0x2000, 0x4000, 0x4000, 0x0000, 0x0000, /* / */
    0x2000, 0x5000, 0x7000, 0x5000, 0x5000, 0x2000, 0x0000, 0x0000, /* 0 */
    0x2000, 0x6000, 0x2000, 0x2000, 0x2000, 0x7000, 0x0000, 0x0000, /* 1 */
    0x2000, 0x5000, 0x1000, 0x2000, 0x4000, 0x7000, 0x0000, 0x0000, /* 2 */
    0x7000, 0x1000, 0x2000, 0x1000, 0x5000, 0x2000, 0x0000, 0x0000, /* 3 */
    0x1000, 0x3000, 0x5000, 0x7000, 0x1000, 0x1000, 0x0000, 0x0000, /* 4 */
    0x7000, 0x4000, 0x6000, 0x1000, 0x5000, 0x2000, 0x0000, 0x0000, /* 5 */
    0x3000, 0x4000, 0x6000, 0x5000, 0x5000, 0x2000, 0x0000, 0x0000, /* 6 */
    0x7000, 0x1000, 0x1000, 0x2000, 0x2000, 0x2000, 0x0000, 0x0000, /* 7 */
    0x2000, 0x5000, 0x2000, 0x5000, 0x5000, 0x2000, 0x0000, 0x0000, /* 8 */
    0x2000, 0x5000, 0x5000, 0x3000, 0x1000, 0x6000, 0x0000, 0x0000, /* 9 */
    0x0000, 0x2000, 0x0000, 0x0000, 0x2000, 0x0000, 0x0000, 0x0000, /* : */
    0x0000, 0x2000, 0x0000, 0x0000, 0x2000, 0x4000, 0x0000, 0x0000, /* ; */
    0x0000, 0x1000, 0x2000, 0x4000, 0x2000, 0x1000, 0x0000, 0x0000, /* < */
```

```
    0x0000, 0x0000, 0x7000, 0x0000, 0x7000, 0x0000, 0x0000, 0x0000, /* = */
    0x0000, 0x4000, 0x2000, 0x1000, 0x2000, 0x4000, 0x0000, 0x0000, /* > */
    0x2000, 0x5000, 0x1000, 0x2000, 0x0000, 0x2000, 0x0000, 0x0000, /* ? */
    0x2000, 0x5000, 0x5000, 0x4000, 0x4000, 0x3000, 0x0000, 0x0000, /* @ */
    0x2000, 0x5000, 0x5000, 0x7000, 0x5000, 0x5000, 0x0000, 0x0000, /* A */
    0x6000, 0x5000, 0x6000, 0x5000, 0x5000, 0x6000, 0x0000, 0x0000, /* B */
    0x2000, 0x5000, 0x4000, 0x4000, 0x5000, 0x2000, 0x0000, 0x0000, /* C */
    0x6000, 0x5000, 0x5000, 0x5000, 0x5000, 0x6000, 0x0000, 0x0000, /* D */
    0x7000, 0x4000, 0x7000, 0x4000, 0x4000, 0x7000, 0x0000, 0x0000, /* E */
    0x7000, 0x4000, 0x7000, 0x4000, 0x4000, 0x4000, 0x0000, 0x0000, /* F */
    0x2000, 0x5000, 0x4000, 0x5000, 0x5000, 0x2000, 0x0000, 0x0000, /* G */
    0x5000, 0x5000, 0x7000, 0x5000, 0x5000, 0x5000, 0x0000, 0x0000, /* H */
    0x7000, 0x2000, 0x2000, 0x2000, 0x2000, 0x7000, 0x0000, 0x0000, /* I */
    0x1000, 0x1000, 0x1000, 0x1000, 0x5000, 0x2000, 0x0000, 0x0000, /* J */
    0x5000, 0x5000, 0x6000, 0x5000, 0x5000, 0x5000, 0x0000, 0x0000, /* K */
    0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x7000, 0x0000, 0x0000, /* L */
    0x5000, 0x7000, 0x5000, 0x5000, 0x5000, 0x5000, 0x0000, 0x0000, /* M */
    0x1000, 0x5000, 0x7000, 0x7000, 0x5000, 0x4000, 0x0000, 0x0000, /* N */
    0x2000, 0x5000, 0x5000, 0x5000, 0x5000, 0x2000, 0x0000, 0x0000, /* O */
    0x6000, 0x5000, 0x5000, 0x6000, 0x4000, 0x4000, 0x0000, 0x0000, /* P */
    0x2000, 0x5000, 0x5000, 0x5000, 0x5000, 0x6000, 0x3000, 0x0000, /* Q */
    0x6000, 0x5000, 0x5000, 0x6000, 0x5000, 0x5000, 0x0000, 0x0000, /* R */
    0x2000, 0x5000, 0x2000, 0x1000, 0x5000, 0x2000, 0x0000, 0x0000, /* S */
    0x7000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x0000, 0x0000, /* T */
    0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x7000, 0x0000, 0x0000, /* U */
    0x5000, 0x5000, 0x5000, 0x5000, 0x5000, 0x2000, 0x0000, 0x0000, /* V */
    0x5000, 0x5000, 0x5000, 0x5000, 0x7000, 0x5000, 0x0000, 0x0000, /* W */
    0x5000, 0x5000, 0x2000, 0x5000, 0x5000, 0x5000, 0x0000, 0x0000, /* X */
    0x5000, 0x5000, 0x2000, 0x2000, 0x2000, 0x2000, 0x0000, 0x0000, /* Y */
    0x7000, 0x1000, 0x2000, 0x2000, 0x4000, 0x7000, 0x0000, 0x0000, /* Z */
    0x3000, 0x2000, 0x2000, 0x2000, 0x2000, 0x3000, 0x0000, 0x0000, /* ’ */
    0x4000, 0x4000, 0x2000, 0x2000, 0x1000, 0x1000, 0x0000, 0x0000, /* \ */
    0x6000, 0x2000, 0x2000, 0x2000, 0x2000, 0x6000, 0x0000, 0x0000, /* ’ */
    0x2000, 0x5000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, /* ^ */
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x7000, 0x0000, /* _ */
    0x2000, 0x1000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, /* ‘ */
    0x0000, 0x6000, 0x1000, 0x3000, 0x5000, 0x3000, 0x0000, 0x0000, /* a */
    0x4000, 0x4000, 0x6000, 0x5000, 0x5000, 0x6000, 0x0000, 0x0000, /* b */
    0x0000, 0x3000, 0x4000, 0x4000, 0x4000, 0x3000, 0x0000, 0x0000, /* c */
    0x1000, 0x1000, 0x3000, 0x5000, 0x5000, 0x3000, 0x0000, 0x0000, /* d */
    0x0000, 0x2000, 0x5000, 0x7000, 0x4000, 0x3000, 0x0000, 0x0000, /* e */
    0x1000, 0x2000, 0x7000, 0x2000, 0x2000, 0x2000, 0x0000, 0x0000, /* f */
    0x0000, 0x3000, 0x5000, 0x5000, 0x3000, 0x1000, 0x6000, 0x0000, /* g */
    0x4000, 0x4000, 0x6000, 0x5000, 0x5000, 0x5000, 0x0000, 0x0000, /* h */
```

```
    0x2000, 0x0000, 0x6000, 0x2000, 0x2000, 0x7000, 0x0000, 0x0000, /* i */
    0x1000, 0x0000, 0x3000, 0x1000, 0x1000, 0x1000, 0x6000, 0x0000, /* j */
    0x4000, 0x4000, 0x5000, 0x6000, 0x5000, 0x5000, 0x0000, 0x0000, /* k */
    0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x0000, 0x0000, /* l */
    0x0000, 0x5000, 0x7000, 0x5000, 0x5000, 0x5000, 0x0000, 0x0000, /* m */
    0x0000, 0x6000, 0x5000, 0x5000, 0x5000, 0x5000, 0x0000, 0x0000, /* n */
    0x0000, 0x2000, 0x5000, 0x5000, 0x5000, 0x2000, 0x0000, 0x0000, /* o */
    0x0000, 0x6000, 0x5000, 0x5000, 0x6000, 0x4000, 0x4000, 0x0000, /* p */
    0x0000, 0x3000, 0x5000, 0x5000, 0x3000, 0x1000, 0x1000, 0x0000, /* q */
    0x0000, 0x3000, 0x4000, 0x4000, 0x4000, 0x4000, 0x0000, 0x0000, /* r */
    0x0000, 0x3000, 0x4000, 0x2000, 0x1000, 0x6000, 0x0000, 0x0000, /* s */
    0x2000, 0x2000, 0x7000, 0x2000, 0x2000, 0x2000, 0x0000, 0x0000, /* t */
    0x0000, 0x5000, 0x5000, 0x5000, 0x5000, 0x7000, 0x0000, 0x0000, /* u */
    0x0000, 0x5000, 0x5000, 0x5000, 0x5000, 0x2000, 0x0000, 0x0000, /* v */
    0x0000, 0x5000, 0x5000, 0x5000, 0x7000, 0x5000, 0x0000, 0x0000, /* w */
    0x0000, 0x5000, 0x5000, 0x2000, 0x5000, 0x5000, 0x0000, 0x0000, /* x */
    0x0000, 0x5000, 0x5000, 0x5000, 0x3000, 0x1000, 0x6000, 0x0000, /* y */
    0x0000, 0x7000, 0x1000, 0x2000, 0x4000, 0x7000, 0x0000, 0x0000, /* z */
    0x1000, 0x2000, 0x2000, 0x4000, 0x2000, 0x2000, 0x1000, 0x0000, /* { */
    0x2000, 0x2000, 0x2000, 0x0000, 0x2000, 0x2000, 0x2000, 0x0000, /* | */
    0x4000, 0x2000, 0x2000, 0x1000, 0x2000, 0x2000, 0x4000, 0x0000, /* } */
    0x0000, 0x0000, 0x0000, 0x6000, 0x3000, 0x0000, 0x0000, 0x0000, /* ~ */
};
FontDef Font_5x8 = {5,8,Font5x8 };
```

0.4.10 main.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
```

```c
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "ESP01.h"
#include "MPU6050.h"
#include "st7735.h"
#include "fonts.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;

IWDG_HandleTypeDef hiwdg;

SPI_HandleTypeDef hspi1;

UART_HandleTypeDef huart1;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
```

```c
static void MX_USART1_UART_Init(void);
static void MX_I2C1_Init(void);
static void MX_IWDG_Init(void);
static void MX_SPI1_Init(void);
/* USER CODE BEGIN PFP */
volatile uint8_t uart_rx_data = 0;
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if (huart->Instance == USART1)  // Check if the interrupt is from USART1
    {
        if (uart_rx_data == '0')  // If received data is '0', restart the system
        {
            NVIC_SystemReset();
        }
        HAL_UART_Receive_IT(&huart1, &uart_rx_data, 1);  // Re-enable UART receive
            interrupt
    }
}
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
```

```c

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART1_UART_Init();
MX_I2C1_Init();
MX_IWDG_Init();
MX_SPI1_Init();
/* USER CODE BEGIN 2 */
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4,1);//enable accelrometer


ST7735_Init();

HAL_IWDG_Refresh(&hiwdg);

HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8,1);
ESP_Init();
MPU6050_Init(&hi2c1);

ST7735_WriteString(11, STATION_Y,"STATION IS WORKING..", Font_5x8, ST7735_GREEN,
    ST7735_BLACK);
HAL_IWDG_Refresh(&hiwdg);
float accelMagnitude=0;


/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
 //HAL_UART_Receive_IT(&huart1, &uart_rx_data, 1);
int x=0;
ST7735_FillRectangleFast(0, ACCEL_Y, 128, 50, ST7735_BLACK);
ST7735_WriteString(20, ACCEL_Y,"ACC IS WORKING..", Font_5x8, ST7735_GREEN,
    ST7735_BLACK);
while (1)
{

```

```
    accelMagnitude=MPU6050_Read_Accel(&hi2c1);
    if ( accelMagnitude==0 ){ //
      HAL_IWDG_Refresh(&hiwdg);

      x+=1;
      if (x==500){
       HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, 0);

      }

      continue;
    }
    else if ( accelMagnitude==-1 || isnanf(accelMagnitude) ){
      // HAL_IWDG_Refresh(&hiwdg);
      ST7735_FillRectangleFast(0, ACCEL_Y, 128, 50, ST7735_BLACK);
      ST7735_WriteString(20, ACCEL_Y,"ACC ERROR!!", Font_5x8, ST7735_WHITE, ST7735_BLACK
          );
      ST7735_WriteString(20, ACCEL_Y+10,"RESETTING..", Font_5x8, ST7735_WHITE,
          ST7735_BLACK);
      HAL_Delay(1000);
      NVIC_SystemReset();

    }
    //HAL_Delay(1);
    ESP_Send_Data( accelMagnitude, 1);
    HAL_IWDG_Refresh(&hiwdg);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8,1);

    x=0;




    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
```

```c
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_LSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.LSIState = RCC_LSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */
```

```c

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief IWDG Initialization Function
  * @param None
  * @retval None
  */
static void MX_IWDG_Init(void)
{

  /* USER CODE BEGIN IWDG_Init 0 */

  /* USER CODE END IWDG_Init 0 */

  /* USER CODE BEGIN IWDG_Init 1 */

  /* USER CODE END IWDG_Init 1 */
  hiwdg.Instance = IWDG;
  hiwdg.Init.Prescaler = IWDG_PRESCALER_256;
  hiwdg.Init.Reload = 4095;
  if (HAL_IWDG_Init(&hiwdg) != HAL_OK)
  {
    Error_Handler();
```

```c
  }
  /* USER CODE BEGIN IWDG_Init 2 */

  /* USER CODE END IWDG_Init 2 */

}

/**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI1_Init(void)
{

  /* USER CODE BEGIN SPI1_Init 0 */

  /* USER CODE END SPI1_Init 0 */

  /* USER CODE BEGIN SPI1_Init 1 */

  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI1_Init 2 */

  /* USER CODE END SPI1_Init 2 */

}
```

```
/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */

```

```c
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4|GPIO_PIN_8, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_10, GPIO_PIN_RESET);

  /*Configure GPIO pins : PA4 PA8 */
  GPIO_InitStruct.Pin = GPIO_PIN_4|GPIO_PIN_8;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : PB0 PB1 PB10 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_10;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
```

```
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

0.4.11 st7735.c

```
/* vim: set ai et ts=4 sw=4: */
#include "stm32f1xx_hal.h"
#include "st7735.h"
#include "malloc.h"
#include "string.h"

#define DELAY 0x80

// based on Adafruit ST7735 library for Arduino
static const uint8_t
  init_cmds1[] = {            // Init for 7735R, part 1 (red or green tab)
    15,                       // 15 commands in list:
    ST7735_SWRESET,   DELAY,  //  1: Software reset, 0 args, w/delay
      150,                    //     150 ms delay
    ST7735_SLPOUT ,   DELAY,  //  2: Out of sleep mode, 0 args, w/delay
      255,                    //     500 ms delay
    ST7735_FRMCTR1, 3      ,  //  3: Frame rate ctrl - normal mode, 3 args:
      0x01, 0x2C, 0x2D,       //     Rate = fosc/(1x2+40) * (LINE+2C+2D)
```

```
    ST7735_FRMCTR2, 3      , //  4: Frame rate control - idle mode, 3 args:
      0x01, 0x2C, 0x2D,      //     Rate = fosc/(1x2+40) * (LINE+2C+2D)
    ST7735_FRMCTR3, 6      , //  5: Frame rate ctrl - partial mode, 6 args:
      0x01, 0x2C, 0x2D,      //     Dot inversion mode
      0x01, 0x2C, 0x2D,      //     Line inversion mode
    ST7735_INVCTR , 1      , //  6: Display inversion ctrl, 1 arg, no delay:
      0x07,                  //     No inversion
    ST7735_PWCTR1 , 3      , //  7: Power control, 3 args, no delay:
      0xA2,
      0x02,                  //     -4.6V
      0x84,                  //     AUTO mode
    ST7735_PWCTR2 , 1      , //  8: Power control, 1 arg, no delay:
      0xC5,                  //     VGH25 = 2.4C VGSEL = -10 VGH = 3 * AVDD
    ST7735_PWCTR3 , 2      , //  9: Power control, 2 args, no delay:
      0x0A,                  //     Opamp current small
      0x00,                  //     Boost frequency
    ST7735_PWCTR4 , 2      , // 10: Power control, 2 args, no delay:
      0x8A,                  //     BCLK/2, Opamp current small & Medium low
      0x2A,
    ST7735_PWCTR5 , 2      , // 11: Power control, 2 args, no delay:
      0x8A, 0xEE,
    ST7735_VMCTR1 , 1      , // 12: Power control, 1 arg, no delay:
      0x0E,
    ST7735_INVOFF , 0      , // 13: Don't invert display, no args, no delay
    ST7735_MADCTL , 1      , // 14: Memory access control (directions), 1 arg:
      ST7735_ROTATION,       //     row addr/col addr, bottom to top refresh
    ST7735_COLMOD , 1      , // 15: set color mode, 1 arg, no delay:
      0x05 },                //     16-bit color

#if (defined(ST7735_IS_128X128) || defined(ST7735_IS_160X128))
  init_cmds2[] = {           // Init for 7735R, part 2 (1.44" display)
    2,                       //  2 commands in list:
    ST7735_CASET  , 4      , //  1: Column addr set, 4 args, no delay:
      0x00, 0x00,            //     XSTART = 0
      0x00, 0x7F,            //     XEND = 127
    ST7735_RASET  , 4      , //  2: Row addr set, 4 args, no delay:
      0x00, 0x00,            //     XSTART = 0
      0x00, 0x7F },          //     XEND = 127
#endif // ST7735_IS_128X128

#ifdef ST7735_IS_160X80
  init_cmds2[] = {           // Init for 7735S, part 2 (160x80 display)
    3,                       //  3 commands in list:
    ST7735_CASET  , 4      , //  1: Column addr set, 4 args, no delay:
```

```c
      0x00, 0x00,             //     XSTART = 0
      0x00, 0x4F,             //     XEND = 79
    ST7735_RASET  , 4      ,  //  2: Row addr set, 4 args, no delay:
      0x00, 0x00,             //     XSTART = 0
      0x00, 0x9F ,            //     XEND = 159
    ST7735_INVON, 0 },        //  3: Invert colors
#endif

  init_cmds3[] = {            // Init for 7735R, part 3 (red or green tab)
    4,                        //  4 commands in list:
    ST7735_GMCTRP1, 16      , //  1: Gamma Adjustments (pos. polarity), 16 args, no
        delay:
      0x02, 0x1c, 0x07, 0x12,
      0x37, 0x32, 0x29, 0x2d,
      0x29, 0x25, 0x2B, 0x39,
      0x00, 0x01, 0x03, 0x10,
    ST7735_GMCTRN1, 16      , //  2: Gamma Adjustments (neg. polarity), 16 args, no
        delay:
      0x03, 0x1d, 0x07, 0x06,
      0x2E, 0x2C, 0x29, 0x2D,
      0x2E, 0x2E, 0x37, 0x3F,
      0x00, 0x00, 0x02, 0x10,
    ST7735_NORON  ,    DELAY, //  3: Normal display on, no args, w/delay
      10,                     //     10 ms delay
    ST7735_DISPON ,    DELAY, //  4: Main screen turn on, no args w/delay
      100 };                  //     100 ms delay

static void ST7735_Select() {
    HAL_GPIO_WritePin(ST7735_CS_GPIO_Port, ST7735_CS_Pin, GPIO_PIN_RESET);
}

void ST7735_Unselect() {
    HAL_GPIO_WritePin(ST7735_CS_GPIO_Port, ST7735_CS_Pin, GPIO_PIN_SET);
}

static void ST7735_Reset() {
    HAL_GPIO_WritePin(ST7735_RES_GPIO_Port, ST7735_RES_Pin, GPIO_PIN_RESET);
    HAL_Delay(5);
    HAL_GPIO_WritePin(ST7735_RES_GPIO_Port, ST7735_RES_Pin, GPIO_PIN_SET);
}

static void ST7735_WriteCommand(uint8_t cmd) {
    HAL_GPIO_WritePin(ST7735_DC_GPIO_Port, ST7735_DC_Pin, GPIO_PIN_RESET);
    HAL_SPI_Transmit(&ST7735_SPI_PORT, &cmd, sizeof(cmd), HAL_MAX_DELAY);
```

```c
}

static void ST7735_WriteData(uint8_t* buff, size_t buff_size) {
    HAL_GPIO_WritePin(ST7735_DC_GPIO_Port, ST7735_DC_Pin, GPIO_PIN_SET);
    HAL_SPI_Transmit(&ST7735_SPI_PORT, buff, buff_size, HAL_MAX_DELAY);
}

static void ST7735_ExecuteCommandList(const uint8_t *addr) {
    uint8_t numCommands, numArgs;
    uint16_t ms;

    numCommands = *addr++;
    while(numCommands--) {
        uint8_t cmd = *addr++;
        ST7735_WriteCommand(cmd);

        numArgs = *addr++;
        // If high bit set, delay follows args
        ms = numArgs & DELAY;
        numArgs &= ~DELAY;
        if(numArgs) {
            ST7735_WriteData((uint8_t*)addr, numArgs);
            addr += numArgs;
        }

        if(ms) {
            ms = *addr++;
            if(ms == 255) ms = 500;
            HAL_Delay(ms);
        }
    }
}

static void ST7735_SetAddressWindow(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1) {
    // column address set
    ST7735_WriteCommand(ST7735_CASET);
    uint8_t data[] = { 0x00, x0 + ST7735_XSTART, 0x00, x1 + ST7735_XSTART };
    ST7735_WriteData(data, sizeof(data));

    // row address set
    ST7735_WriteCommand(ST7735_RASET);
    data[1] = y0 + ST7735_YSTART;
    data[3] = y1 + ST7735_YSTART;
    ST7735_WriteData(data, sizeof(data));
```

```c

    // write to RAM
    ST7735_WriteCommand(ST7735_RAMWR);
}

void ST7735_Init() {
    ST7735_Select();
    ST7735_Reset();
    ST7735_ExecuteCommandList(init_cmds1);
    ST7735_ExecuteCommandList(init_cmds2);
    ST7735_ExecuteCommandList(init_cmds3);
    ST7735_InvertColors(1);
    ST7735_Unselect();
    ST7735_FillScreenFast(ST7735_BLACK);
    ST7735_FillRectangleFast(0, 0, 128, 10, ST7735_WHITE);
    ST7735_WriteString(20, 2,"EARTHQUAKE STATION", Font_5x8, ST7735_BLACK, ST7735_WHITE)
        ;
    ST7735_FillRectangleFast(0, 40, 128, 10, ST7735_WHITE);
    ST7735_WriteString(30, 42,"ESP01 STATUS", Font_5x8, ST7735_BLACK, ST7735_WHITE);
    ST7735_FillRectangleFast(0, 105, 128, 10, ST7735_WHITE);
    ST7735_WriteString(15, 107,"ACCELEROMETER STATUS", Font_5x8, ST7735_BLACK,
        ST7735_WHITE);
}

void ST7735_DrawPixel(uint16_t x, uint16_t y, uint16_t color) {
    if((x >= ST7735_WIDTH) || (y >= ST7735_HEIGHT))
        return;

    ST7735_Select();

    ST7735_SetAddressWindow(x, y, x+1, y+1);
    uint8_t data[] = { color >> 8, color & 0xFF };
    ST7735_WriteData(data, sizeof(data));

    ST7735_Unselect();
}

static void ST7735_WriteChar(uint16_t x, uint16_t y, char ch, FontDef font, uint16_t
    color, uint16_t bgcolor) {
    uint32_t i, b, j;
    if (ch == '\r' || ch == '\n' ) {
        ch = ' ';
    }
    ST7735_SetAddressWindow(x, y, x+font.width-1, y+font.height-1);
```

```c

    for(i = 0; i < font.height; i++) {
        b = font.data[(ch - 32) * font.height + i];
        for(j = 0; j < font.width; j++) {
            if((b << j) & 0x8000)  {
                uint8_t data[] = { color >> 8, color & 0xFF };
                ST7735_WriteData(data, sizeof(data));
            } else {
                uint8_t data[] = { bgcolor >> 8, bgcolor & 0xFF };
                ST7735_WriteData(data, sizeof(data));
            }
        }
    }
}

/*
Simpler (and probably slower) implementation:

static void ST7735_WriteChar(uint16_t x, uint16_t y, char ch, FontDef font, uint16_t
    color) {
    uint32_t i, b, j;

    for(i = 0; i < font.height; i++) {
        b = font.data[(ch - 32) * font.height + i];
        for(j = 0; j < font.width; j++) {
            if((b << j) & 0x8000)  {
                ST7735_DrawPixel(x + j, y + i, color);
            }
        }
    }
}
*/

void ST7735_WriteString(uint16_t x, uint16_t y, const char* str, FontDef font, uint16_t
    color, uint16_t bgcolor) {
    ST7735_Select();

    while(*str) {
        if(x + font.width >= ST7735_WIDTH) {
            x = 0;
            y += font.height;
            if(y + font.height >= ST7735_HEIGHT) {
              ST7735_FillScreenFast(ST7735_BLACK);
              break;
```

```
            }

            if(*str == ' ' ) {
                // skip spaces in the beginning of the new line
                str++;
                continue;
            }
        }

        ST7735_WriteChar(x, y, *str, font, color, bgcolor);
        x += font.width;

        str++;
    }

    ST7735_Unselect();
}

void ST7735_FillRectangle(uint16_t x, uint16_t y, uint16_t w, uint16_t h, uint16_t color
    ) {
    // clipping
    if((x >= ST7735_WIDTH) || (y >= ST7735_HEIGHT)) return;
    if((x + w - 1) >= ST7735_WIDTH) w = ST7735_WIDTH - x;
    if((y + h - 1) >= ST7735_HEIGHT) h = ST7735_HEIGHT - y;

    ST7735_Select();
    ST7735_SetAddressWindow(x, y, x+w-1, y+h-1);

    uint8_t data[] = { color >> 8, color & 0xFF };
    HAL_GPIO_WritePin(ST7735_DC_GPIO_Port, ST7735_DC_Pin, GPIO_PIN_SET);
    for(y = h; y > 0; y--) {
        for(x = w; x > 0; x--) {
            HAL_SPI_Transmit(&ST7735_SPI_PORT, data, sizeof(data), HAL_MAX_DELAY);
        }
    }

    ST7735_Unselect();
}

void ST7735_FillRectangleFast(uint16_t x, uint16_t y, uint16_t w, uint16_t h, uint16_t
    color) {
    // clipping
    if((x >= ST7735_WIDTH) || (y >= ST7735_HEIGHT)) return;
    if((x + w - 1) >= ST7735_WIDTH) w = ST7735_WIDTH - x;
```

```c
    if((y + h - 1) >= ST7735_HEIGHT) h = ST7735_HEIGHT - y;

    ST7735_Select();
    ST7735_SetAddressWindow(x, y, x+w-1, y+h-1);

    // Prepare whole line in a single buffer
    uint8_t pixel[] = { color >> 8, color & 0xFF };
    uint8_t *line = malloc(w * sizeof(pixel));
    for(x = 0; x < w; ++x)
      memcpy(line + x * sizeof(pixel), pixel, sizeof(pixel));

    HAL_GPIO_WritePin(ST7735_DC_GPIO_Port, ST7735_DC_Pin, GPIO_PIN_SET);
    for(y = h; y > 0; y--)
        HAL_SPI_Transmit(&ST7735_SPI_PORT, line, w * sizeof(pixel), HAL_MAX_DELAY);

    free(line);
    ST7735_Unselect();
}

void ST7735_FillScreen(uint16_t color) {
    ST7735_FillRectangle(0, 0, ST7735_WIDTH, ST7735_HEIGHT, color);
}

void ST7735_FillScreenFast(uint16_t color) {
    ST7735_FillRectangleFast(0, 0, ST7735_WIDTH, ST7735_HEIGHT, color);
}


void ST7735_InvertColors(bool invert) {
    ST7735_Select();
    ST7735_WriteCommand(invert ? ST7735_INVON : ST7735_INVOFF);
    ST7735_Unselect();
}

void ST7735_SetGamma(GammaDef gamma)
{
  ST7735_Select();
  ST7735_WriteCommand(ST7735_GAMSET);
  ST7735_WriteData((uint8_t *) &gamma, sizeof(gamma));
  ST7735_Unselect();
}
```

0.5 Database Code

0.5.1 station.sql

```sql
-- phpMyAdmin SQL Dump
-- version 5.2.1
-- https://www.phpmyadmin.net/
--
-- Host: 127.0.0.1
-- Generation Time: Jan 18, 2025 at 08:08 AM
-- Server version: 10.4.32-MariaDB
-- PHP Version: 8.1.25

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `station`
--

-- --------------------------------------------------------

--
-- Table structure for table `events`
--

CREATE TABLE `events` (
  `station_id` int(11) NOT NULL,
  `acceleration` float NOT NULL,
  `velocity` float NOT NULL,
  `displacement` float NOT NULL,
  `richter` float NOT NULL,
  `date` datetime NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;

-- --------------------------------------------------------

```

```
--
-- Table structure for table `station_table`
--

CREATE TABLE `station_table` (
  `station_id` int(11) NOT NULL,
  `location` text NOT NULL,
  `coordinates` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;

--
-- Indexes for dumped tables
--

--
-- Indexes for table `events`
--
ALTER TABLE `events`
  ADD KEY `events_ibfk_1` (`station_id`);

--
-- Indexes for table `station_table`
--
ALTER TABLE `station_table`
  ADD PRIMARY KEY (`station_id`);

--
-- AUTO_INCREMENT for dumped tables
--

--
-- AUTO_INCREMENT for table `station_table`
--
ALTER TABLE `station_table`
  MODIFY `station_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=2;

--
-- Constraints for dumped tables
--

--
-- Constraints for table `events`
--
ALTER TABLE `events`
```

```
  ADD CONSTRAINT `events_ibfk_1` FOREIGN KEY (`station_id`) REFERENCES `station_table`
      (`station_id`);
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

0.5.2 database.js

```
import mysql from 'mysql2'

export const connect=mysql.createConnection({
        host:'mysql',//mysql
        password:'',
        database:'station',
        user:'root',
        port:3306,
        connectTimeout:10000
    }
)
```

0.6 Frontend

0.6.1 index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
```

```
      user's mobile device or desktop. See https://developers.google.com/web/
          fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>
```

0.6.2 index.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
```

```
}
code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
.active{
  font-weight: bold;
}
```

0.6.3 index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { BrowserRouter } from 'react-router-dom'
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

0.6.4 App.test.js

```
import { render, screen } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

0.6.5 reportWebVitals.js

```
const reportWebVitals = onPerfEntry => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};
export default reportWebVitals;
```

0.6.6 setupTests.js

```
// jest-dom adds custom jest matchers for asserting on DOM nodes.
// allows you to do things like:
// expect(element).toHaveTextContent(/react/i)
// learn more: https://github.com/testing-library/jest-dom
import '@testing-library/jest-dom';
```

0.6.7 Introduction.js

```
import React, { useState } from "react";
export default function Introduction() {
  const [colapse, set_colapse] = useState(false);

  function handle_colapse() {
    set_colapse(!colapse);
  }

  return (
    <div className="flex justify-center w-full">
      <button
        onClick={handle_colapse}
        className="text-center italic m-2 p-2 w-full bg-gradient-to-l from-gray-800 via-
            blue-500 to-gray-800 text-white text-sm font-mono border border-black flex
            flex-col items-center">
        <h2 className="font-bold not-italic">Welcome to the Earthquake Station Dashboard
            </h2>
```

```
        {colapse && (
          <div>
            <p>
              This platform provides real-time data from our IoT-based earthquake detection system. Our station continuously monitors seismic activity, detecting even the smallest vibrations. The data is displayed live, helping you stay informed about any significant events in the area.</p>
            <p>
              Below, y o u ll find key insights including earthquake magnitude, displacement readings, and real-time alerts. You can also explore historical data and adjust station settings to tailor the monitoring to your needs.</p>
            <br />
            <h1>Know more about earthquakes:</h1>
            <iframe
              className="w-full h-64"
              src="https://www.youtube.com/embed/vEgLjgnv_3c"
            ></iframe>
          </div>
        )}
      </button>
    </div>
  );
}
```

0.6.8 about.js

```
import React from 'react'

export default function About() {
  return (
    <div className='flex flex-col text-white m10 items-center justify-center bg-sky-900 h-[calc(100vh-40px)]'>
      <h1 className='text-5xl'>Abstract</h1>
      <p className='m-10 text-2xl'>
This project presents a cost-effective IoT-based Earthquake Station designed to detect seismic activity using an accelerometer and Wi-Fi for real-time data transmission. The system measures ground motion and operates only during seismic events,
ensuring optimal energy efficiency, an essential feature for remote or off-grid locations. The system is only activated when needed. The project emphasizes reliable
earthquake detection, efficient communication, and energy-saving techniques, adhering to a streamlined design approach. To address potential challenges, the agile
```

```
    development methodology is adopted, enabling flexibility and continuous refinement
throughout the project lifecycle.</p>
  </div>
  )
}
```

0.6.9 history.js

```
import React, { useState } from 'react';

export default function History(props) {
  const info = props.info;
  const [currentPage, setCurrentPage] = useState(0); // Track the current page
  const recordsPerPage = 5; // Number of records to display per page

  // Calculate the index of the first and last record on the current page
  const startIndex = currentPage * recordsPerPage;
  const endIndex = startIndex + recordsPerPage;

  // Get the records to display on the current page
  const currentRecords = info.slice(startIndex, endIndex);

  // Function to handle page change
  const goToNextPage = () => {
    if (endIndex < info.length) {
      setCurrentPage(currentPage + 1);
    }
  };

  const goToPreviousPage = () => {
    if (startIndex > 0) {
      setCurrentPage(currentPage - 1);
    }
  };

  return (
    <div>
      {info && info.length > 0 ? (
        <div>
          <table className="w-full border-collapse border border-gray-400">
            <thead>
              <tr>
                <th className="border border-gray-400 px-4 py-2">Date</th>
```

```
              <th className="border border-gray-400 px-4 py-2">Acceleration</th>
              <th className="border border-gray-400 px-4 py-2">Velocity</th>
              <th className="border border-gray-400 px-4 py-2">Displacement</th>
              <th className="border border-gray-400 px-4 py-2">Richter</th>
            </tr>
          </thead>
          <tbody>
            {currentRecords.map((item, index) => (
              <tr key={index}>
                <td className="border border-gray-400 px-4 py-2">{String(item.date).
                    slice(0,10)}</td>
                <td className="border border-gray-400 px-4 py-2">{item.acceleration}</
                    td>
                <td className="border border-gray-400 px-4 py-2">{item.velocity}</td>
                <td className="border border-gray-400 px-4 py-2">{item.displacement}</
                    td>
                <td className="border border-gray-400 px-4 py-2">{String(item.richter)
                    .slice(0,5)}</td>
              </tr>
            ))}
          </tbody>
        </table>

        {/* Pagination Controls */}
        <div className="flex justify-between mt-4">
          <button
            onClick={goToPreviousPage}
            disabled={startIndex === 0}
            className="bg-sky-900 text-white px-4 py-2 disabled:bg-gray-300 disabled:
                text-black">
            Previous
          </button>
          <button
            onClick={goToNextPage}
            disabled={endIndex >= info.length}
            className="bg-sky-900 text-white px-4 py-2 disabled:bg-gray-300 disabled:
                text-black">
            Next
          </button>
        </div>
      </div>
    ) : (
      <p>No data available</p>
    )}
```

```
    </div>
  );
}
```

0.6.10 home.js

```
import React, { useState, useEffect } from 'react';
import Station_info from './station_info';
import Introduction from './Introduction';

export default function Home() {
  const [searchQuery, setSearchQuery] = useState('');
  const [colapse_all, colapse_all_setter] = useState(false);
  const [currentPage, setCurrentPage] = useState(1);
  const [data, setData] = useState([]);
  const stationsPerPage = 5;

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await fetch('http://localhost:3001/stations/');
        const result = await response.json();
        console.log("Fetched data:", result.data); // Log fetched data for debugging
        setData(result.data);
      } catch (error) {
        console.error('Error fetching station data:', error);
      }
    };

    fetchData(); // Call the fetch function
  }, []);

  // Filter the data based on the search query
  const filteredData = data.filter((item) => {
    return item.location.toLowerCase().includes(searchQuery.toLowerCase());
  });

  // Calculate the index of the first and last item on the current page
  const indexOfLastStation = currentPage * stationsPerPage;
  const indexOfFirstStation = indexOfLastStation - stationsPerPage;

  // Get the current stations to display based on the page
  const currentStations = filteredData.slice(indexOfFirstStation, indexOfLastStation);
```

```
// Handle next and previous page
const nextPage = () => {
  if (currentPage < Math.ceil(filteredData.length / stationsPerPage)) {
    setCurrentPage(currentPage + 1);
  }
};

const prevPage = () => {
  if (currentPage > 1) {
    setCurrentPage(currentPage - 1);
  }
};

return (
  <div>
    <Introduction></Introduction>
    <div className="m-2 bg-gray-300 pb-2 h-full">
      <div className="m-2 text-2xl text-white text-center flex justify-between py-2
          items-center gap-16 search_bar">
        <p className="text-sky-900">Available Stations</p>
        <input
          type="text"
          placeholder="Search by location"
          className="text-xl h-10 w-1/2 text-black rounded-full"
          onChange={(e) => setSearchQuery(e.target.value)}
        />
        <button
          className="flex items-center bg-sky-900 p-2 text-base hover:bg-gray-800"
          onClick={() => {
            colapse_all_setter(!colapse_all);
          }}
        >
          Collapse All
        </button>
      </div>

      <div className="m-2 flex flex-col gap-1">
        {currentStations.map((station) => (
          <Station_info
            key={station.station_id} // Ensure each station has a unique key
            station_id={station.station_id}
            location={station.location}
            data={station} // Ensure data is passed properly
```

```
                colapse={colapse_all}
                coordinates={station.coordinates}
              />

          ))}
        </div>

        {/* Pagination Controls */}
        <div className="flex justify-between mx-2">
          <button
            onClick={prevPage}
            className="bg-sky-900 text-white p-2 hover:bg-gray-800"
            disabled={currentPage === 1}
          >
            Previous
          </button>
          <button
            onClick={nextPage}
            className="bg-sky-900 text-white p-2 hover:bg-gray-800"
            disabled={currentPage === Math.ceil(filteredData.length / stationsPerPage)}
          >
            Next
          </button>
        </div>
      </div>
    </div>
  );
}
```

0.6.11 navbar.js

```
import React from 'react'
import { NavLink} from 'react-router-dom'

export default function Navbar() {

  return (
    <div className='bg-gray-800 text-white text-2xl flex justify-between items-center h
        -10 border-b border-black overflow-hidden navbar'>
        <NavLink to="/home" className=' px-16 h-10 flex items-center hover:bg-sky-900
            hover:text-white'>Earthquake Stations</NavLink>
        <div className='flex justify-around items-center '>
```

```
            <NavLink className='px-16 h-10 flex items-center hover:bg-sky-900 hover:text
                -white' to='/about'>About</NavLink>
        </div>
    </div>
  )
}
```

0.6.12 station_info.js

```
import React, { useEffect, useState } from "react";
import History from "./history";

export default function Station_info(props) {
  let id = props.station_id;
  let location=props.location;
  let coordinates=props.coordinates
  let [arrow, arrow_setter] = useState(
    <div className="rounded-full ml-2 bg-black h-4 w-4"></div>
  );
  let [iscollapsed, iscollapsed_setter] = useState(false);
  let [stationinfo, setstationinfo] = useState([]);
  let [history, setHistory] = useState([]);
  let [loading, setLoading] = useState(true); // For station info
  let [loadingHistory, setLoadingHistory] = useState(true); // For history

  // Function to fetch history data using REST remains unchanged
  const fetchHistoryData = async () => {
    console.log(coordinates)
    try {
      const response = await fetch(`http://localhost:3001/history/${id}`);
      const result = await response.json();
      setHistory(result.data);
      setLoadingHistory(false);
    } catch (error) {
      console.error("Error fetching history data:", error);
      setLoadingHistory(false);
    }
  };

  // Client-side WebSocket connection for real-time station data
  useEffect(() => {
    const ws = new WebSocket("ws://localhost:3001");

```

```
    ws.onopen = () => {
      console.log("WebSocket connected for station info");
      ws.send(JSON.stringify({ type: "getStationLastRead", station_id: id }));
    };

    ws.onmessage = (event) => {

      try {
        const data = JSON.parse(event.data);
        if (data.type === "stationLastRead" && data.data.station_id === id) {
          setstationinfo([data.data]);
          setLoading(false);

        }
      } catch (error) {
        console.error("Error parsing WebSocket message:", error);
        setLoading(false);
      }
    };

    ws.onerror = (error) => {
      console.error("WebSocket error:", error);
    };

    ws.onclose = () => {
      console.log("WebSocket connection closed for station info");
    };

  }, [id]);

  // Keep REST polling for history data every 5 seconds
  useEffect(() => {
    fetchHistoryData(); // Initial fetch for history data

    const interval = setInterval(() => {
      fetchHistoryData();
    }, 5000);

    return () => clearInterval(interval);
  }, [id]);

  useEffect(() => {
    iscollapsed_setter(false);
    arrow_setter(
```

```
      <div className="rounded-full ml-2 block bg-black h-4 w-4"></div>
    );
  }, [props.colapse]);

  function toggle_collapse() {
    iscollapsed_setter(!iscollapsed);
    arrow_setter(
      <div
        className={`rounded-full ml-2 block h-4 w-4 ${
          iscollapsed ? "bg-black" : "bg-sky-900"
        }`}
      ></div>
    );
  }

  return (
    <div className="flex flex-col justify-center items-center border border-black">
      <button
        className="flex w-full bg-gray-300 h-full items-center gap-2 text-lg hover:bg-
            gray-500"
        onClick={toggle_collapse}
      >
        {arrow} Station ({id})
      </button>

      <div
        className="bg-gray-600 text-white px-10 w-full overflow-hidden"
        style={{
          maxHeight: !iscollapsed ? "0" : "500px",
          transition: "max-height 0.15s ease-in-out",
        }}
      >
        <table className="w-full">
          <thead className="text-center">
            <tr>
              <th>Location</th>
              <th>Acceleration</th>
              <th>Velocity</th>
              <th>Displacement</th>
              <th>Richter's Magnitude</th>
              <th>Date</th>
            </tr>
          </thead>
          <tbody className="text-center">
```

```
            {loading ? (
              <tr>
                <td colSpan="6">No movement detected</td>
              </tr>
            ) : stationinfo.length > 0 ? (
              <tr>
                <td>{location}</td>
                <td>{stationinfo[0]?.acceleration}</td>
                <td>{stationinfo[0]?.velocity}</td>
                <td>{stationinfo[0]?.displacement}</td>
                <td>{stationinfo[0]?.richter}</td>
                <td>{stationinfo[0]?.date}</td>
              </tr>
            ) : (
              <tr>
                <td colSpan="6">No Data Available</td>
              </tr>
            )}
          </tbody>
        </table>

        <div>
          <p className="bg-gray-300 text-black text-center">History</p>
          <History info={history} />
        </div>
          <div className="mt-2 p-2">
        <iframe
          title={`map-${id}`}
          width="100%"
          height="250"
          style={{ border: 0 }}
          loading="lazy"
          allowFullScreen
          referrerPolicy="no-referrer-when-downgrade"
          src={`https://www.google.com/maps?q=${coordinates}&z=14&output=embed`}
        />
      </div>
      </div>
    </div>
  );
}
```

0.7 Backend Code

0.7.1 checkStationExists.js

```
import { connect } from "../database/database.js";

export const checkStationExist=(req,res,next)=>{
    const {id}=req.params
    const connection=connect
    connection.execute(`SELECT station_id FROM station_table WHERE station_id='${id}'`,(err,data)=>{
        if (data.length>0){
            next()
        }
        else {
            return res.status(404).json({message:"No Station with this ID found"})
        }
    })
}
```

0.7.2 controller.js

```
import { connect } from "../database/database.js";
import moment from "moment";
import { WebSocketServer } from 'ws';

const connection = connect;
export const getAllStations = (req,res)=>{
    connection.execute(`SELECT * FROM station_table`,(err,data)=>{
        return res.status(200).json({data})
    })
}

function getCurrentDateTime() {
    const now = new Date();
    const year = now.getFullYear();
    const month = String(now.getMonth() + 1).padStart(2, '0');
    const day = String(now.getDate()).padStart(2, '0');
    const hours = String(now.getHours()).padStart(2, '0');
    const minutes = String(now.getMinutes()).padStart(2, '0');
    const seconds = String(now.getSeconds()).padStart(2, '0');
    return `${year}-${month}-${day} ${hours}:${minutes}:${seconds}`;
}
```

```
export const addEvent = async (ws, wsServer, message) => {
    try {
        const data = JSON.parse(message);
        const { station_id, acceleration, velocity, displacement, richter } = data;
        const date = getCurrentDateTime();

        // Insert the new event into the database using safe values
        await connection.execute(
            `INSERT INTO events (station_id, acceleration, velocity, displacement,
                richter, date) VALUES (?, ?, ?, ?, ?, ?)`,
            [station_id, acceleration, velocity, displacement, richter, date]
        );

        // Format the data for broadcasting
        const formattedData = {
            station_id,
            acceleration: parseFloat(acceleration).toFixed(8),
            velocity: parseFloat(velocity).toFixed(8),
            displacement: parseFloat(displacement).toFixed(8),
            richter: parseFloat(richter).toFixed(8),
            date: moment.utc(date).utcOffset(2).format("hh:mm:ss A"),
        };

        // Broadcast directly to all WebSocket clients
        wsServer.clients.forEach(client => {
            if (client.readyState === 1) { // 1 means WebSocket.OPEN
                client.send(JSON.stringify({ type: "stationLastRead", data:
                    formattedData }));
            }
        });

    } catch (error) {

    }
};

export const getStationHistory = (req, res) => {
    const { id } = req.params;

    const lastReadQuery = `
        SELECT acceleration, velocity, displacement, richter, date
        FROM events
        WHERE station_id = ?
```

```
        ORDER BY date DESC
        LIMIT 1;
    `;

    const maxRichterQuery = `
SELECT
  acceleration,
  velocity,
  displacement,
  richter,
  date
FROM (
  SELECT
    e.*,
    @row_number := IF(
      @current_date = DATE(date),
      @row_number + 1,
      1
    ) AS rn,
    @current_date := DATE(date) AS event_date
  FROM
    events e,
    (SELECT @row_number := 0, @current_date := NULL) AS vars
  WHERE
    e.station_id = ?
    AND e.richter = (
      SELECT MAX(richter)
      FROM events
      WHERE station_id = e.station_id
        AND DATE(date) = DATE(e.date)
    )
  ORDER BY
    DATE(date) DESC,  -- Group by date
    richter DESC,      -- Prioritize max Richter
    date DESC          -- Break ties with latest timestamp
) AS ranked
WHERE rn = 1  -- Keep only the first row per day
ORDER BY date DESC;
    `;

    connection.execute(lastReadQuery, [id], (err1, lastReadData) => {
        if (err1) return res.status(500).json({ error: "Failed to fetch last read." });

        connection.execute(maxRichterQuery, [id], (err2, richterData) => {
```

```
            if (err2) return res.status(500).json({ error: "Failed to fetch station
                history." });

            const formattedLastRead = lastReadData[0]
                ? {
                      ...lastReadData[0],
                      acceleration: parseFloat(lastReadData[0].acceleration).toFixed(8),
                      velocity: parseFloat(lastReadData[0].velocity).toFixed(8),
                      displacement: parseFloat(lastReadData[0].displacement).toFixed(8),
                      richter: parseFloat(lastReadData[0].richter).toFixed(8),
                      date: lastReadData[0].date, // Preserving full date and time
                  }
                : null;

            const formattedRichterData = richterData.map((row) => ({
                ...row,
                acceleration: row.acceleration ? parseFloat(row.acceleration).toFixed(8)
                     : row.acceleration,
                velocity: row.velocity ? parseFloat(row.velocity).toFixed(8) : row.
                    velocity,
                displacement: row.displacement ? parseFloat(row.displacement).toFixed(8)
                     : row.displacement,
                richter: row.richter ? parseFloat(row.richter).toFixed(8) : row.richter,
                date: row.date, // Preserving full date and time
            }));

            const combinedData = formattedLastRead
                ? [formattedLastRead, ...formattedRichterData]
                : [...formattedRichterData];

            // Save all the data (including last read) to the database
            connection.execute(`DELETE FROM events WHERE station_id = ?`, [id]);

            const insertQuery = `
                INSERT INTO events (station_id, acceleration, velocity, displacement,
                    richter, date)
                VALUES (?, ?, ?, ?, ?, ?)
            `;

            for (const row of combinedData) {
                connection.execute(insertQuery, [
                    id,
                    row.acceleration,
                    row.velocity,
```

```
                    row.displacement,
                    row.richter,
                    row.date,
                ]);
            }

            // Exclude the last read entry from the response
            const responseData = formattedRichterData; // Remove formattedLastRead from
                the response

            // Send the response
            return res.status(200).json({ data: responseData });
        });
    });
};
```

0.7.3 routes.js

```
import { Router } from "express";
import { checkStationExist } from "../middlewares/checkStationExist.js";
import {  getAllStations, getStationHistory } from "./controller.js";

const stationRouter=Router()

stationRouter.get('/history/:id',checkStationExist,getStationHistory)
stationRouter.get('/stations',getAllStations)
//stationRouter.post('/addevent',addEvent) //commented for websocket
//stationRouter.get('/stations/:id',checkStationExist,getStationLastRead)

export default stationRouter
```

0.7.4 Dockerfile

```
# Dockerfile for backend
FROM node:16

# Set the working directory
WORKDIR /app

# Install dependencies
COPY package*.json ./
RUN npm install
```

```
# Download wait-for-it script
ADD https://raw.githubusercontent.com/vishnubob/wait-for-it/master/wait-for-it.sh /usr/
    local/bin/wait-for-it
RUN chmod +x /usr/local/bin/wait-for-it

# Copy the rest of your application code
COPY . .

# Expose the necessary port
EXPOSE 3001

# Command to start the app
CMD ["sh", "-c", "/usr/local/bin/wait-for-it mysql:3306 -- node index.js"]
```

0.7.5 index.js

```
import express from 'express';
import cors from 'cors';
import { WebSocketServer } from 'ws'; // Missing WebSocket import
import { addEvent } from './stationsModule/controller.js';
import stationRouter from './stationsModule/routes.js';

const app = express();
const port = 3001;

app.use(cors());
app.use(express.json());
app.use(stationRouter);

// Start HTTP Server
const server = app.listen(port, () => console.log(`Server running on port ${port}`));

// Create WebSocket Server
const wss = new WebSocketServer({ server });

wss.on("connection", (ws) => {
    ws.on("message", (message) => {
        try {
            addEvent(ws,wss, message); // Directly call addEvent on message receive
        } catch (error) {
            console.error("Error processing WebSocket message:", error);
            ws.send(JSON.stringify({ error: "Invalid WebSocket message format" }));
```

```
        }
    });
});
```

0.8 Machine Learning Models

```python
# -*- coding: utf-8 -*-
"""Untitled40.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1PlmA5pHopP6-sMyTKGu8cLArU45Ivobo
"""

import numpy as np
import pandas as pd
import os
from google.colab import files

# Upload 1 to 3 V2c files manually
uploaded = files.upload()

# Function to parse .V2c files
def parse_v2c_data(contents):
    acceleration = []
    for line in contents:
        try:
            values = [float(val) for val in line.strip().split()]
            acceleration.extend(values)
        except ValueError:
            continue
    return np.array(acceleration)

# Load and process uploaded files
acc_arrays = []
for filename in uploaded:
    with open(filename, "r") as f:
        lines = f.readlines()
        acc = parse_v2c_data(lines)
        acc_arrays.append(acc)

```

```
# Make sure all arrays are the same length
min_length = min(len(arr) for arr in acc_arrays)
acc_arrays = [arr[:min_length] for arr in acc_arrays]

# Dynamically calculate magnitude
acc_stack = np.vstack(acc_arrays)  # shape: (n_axes, n_points)
magnitude = np.sqrt(np.sum(acc_stack**2, axis=0))  # shape: (n_points,)

# Create a single row DataFrame with 'acc1', ..., 'accN', and 'earthquake?' = 1
def create_row_df(magnitude, count):
    segment = magnitude[:count]
    data = {f'acc{i+1}': val for i, val in enumerate(segment)}
    data['earthquake?'] = 1
    return pd.DataFrame([data])

# Append or create Excel files
def append_to_excel(filename, new_row_df):
    if os.path.exists(filename):
        existing_df = pd.read_excel(filename)
        combined_df = pd.concat([existing_df, new_row_df], ignore_index=True)
    else:
        combined_df = new_row_df
    combined_df.to_excel(filename, index=False)
    return combined_df

# Process all sizes
df_200 = create_row_df(magnitude, 200)
df_500 = create_row_df(magnitude, 500)
df_1000 = create_row_df(magnitude, 1000)

# Save to Excel
df_200_full = append_to_excel("earthquake_200.xlsx", df_200)
df_500_full = append_to_excel("earthquake_500.xlsx", df_500)
df_1000_full = append_to_excel("earthquake_1000.xlsx", df_1000)

# Show results
print("    Updated earthquake_200.xlsx:")
display(df_200_full.tail())

print("    Updated earthquake_500.xlsx:")
display(df_500_full.tail())

print("    Updated earthquake_1000.xlsx:")
display(df_1000_full.tail())
```

```python

import numpy as np
import pandas as pd
import os

# Sampling setup
fs = 1000  # Hz
duration = 1  # seconds (enough for 1000 samples)
samples = fs * duration
t = np.linspace(0, duration, samples)

# Signal generators
def generate_human_walk():
    freq = np.random.uniform(1.5, 3.0)
    amp = np.random.uniform(0.1, 0.4)
    noise = np.random.normal(0, 0.05, samples)
    return amp * np.sin(2 * np.pi * freq * t) + noise

def generate_truck_driveby():
    freq = np.random.uniform(2.5, 5.0)
    amp = np.random.uniform(0.5, 1.5)
    decay = np.exp(-np.linspace(0, 3, samples))
    noise = np.random.normal(0, 0.1, samples)
    return amp * np.sin(2 * np.pi * freq * t) * decay + noise

def generate_random_noise():
    spikes = np.random.normal(0, 1, samples)
    smooth = np.convolve(spikes, np.ones(20)/20, mode='same')
    return smooth * np.random.uniform(0.1, 0.3)

# Combined generator
def generate_non_eq_signal():
    kind = np.random.choice(['human', 'truck', 'noise'])
    if kind == 'human':
        return generate_human_walk()
    elif kind == 'truck':
        return generate_truck_driveby()
    else:
        return generate_random_noise()

# Append to Excel
def append_to_excel(filename, signals, sample_size):
    rows = []
    for signal in signals:
```

```
        trimmed = signal[:sample_size]
        row = {f'acc{i+1}': val for i, val in enumerate(trimmed)}
        row['earthquake?'] = 0
        rows.append(row)

    df_new = pd.DataFrame(rows)

    if os.path.exists(filename):
        df_existing = pd.read_excel(filename)
        df_combined = pd.concat([df_existing, df_new], ignore_index=True)
    else:
        df_combined = df_new

    df_combined.to_excel(filename, index=False)
    print(f"    Added {len(signals)} non-earthquake signals to {filename}")

# Generate and distribute to all Excel files
non_eq_signals = [generate_non_eq_signal() for _ in range(3)]
append_to_excel('earthquake_200.xlsx', non_eq_signals, 200)
append_to_excel('earthquake_500.xlsx', non_eq_signals, 500)
append_to_excel('earthquake_1000.xlsx', non_eq_signals, 1000)

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import (
    accuracy_score, confusion_matrix, precision_score, recall_score,
    f1_score, roc_curve, auc, roc_auc_score
)

def evaluate_naive_bayes(file_path, label):
    # Load the dataset
    df = pd.read_excel(file_path)
    X = df.drop(columns=['earthquake?'])
    y = df['earthquake?']

    # Split into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35,
        random_state=42)

    # Initialize and train the model
    model = GaussianNB()
    model.fit(X_train, y_train)
```

```python

    # Predictions and probabilities
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]

    # ROC curve and AUC
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    auc_score = roc_auc_score(y_test, y_proba)

    # Plot ROC
    plt.plot(fpr, tpr, label=f'{label} (AUC = {auc_score:.2f})')

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

    return {
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall (True Positive Rate)": recall,
        "F1 Score": f1,
        "True Positive Rate": tp / (tp + fn) if (tp + fn) > 0 else 0,
        "False Positive Rate": fp / (fp + tn) if (fp + tn) > 0 else 0,
        "True Negative Rate": tn / (tn + fp) if (tn + fp) > 0 else 0,
        "False Negative Rate": fn / (fn + tp) if (fn + tp) > 0 else 0,
        "ROC AUC": auc_score
    }

# Evaluate each file
results_200 = evaluate_naive_bayes("earthquake_200.xlsx", "200 Accels")
results_500 = evaluate_naive_bayes("earthquake_500.xlsx", "500 Accels")
results_1000 = evaluate_naive_bayes("earthquake_1000.xlsx", "1000 Accels")

# Display metrics
print("Results for 200 accelerations:")
print(results_200)
print("\nResults for 500 accelerations:")
print(results_500)
print("\nResults for 1000 accelerations:")
print(results_1000)

```

```
# Finalize and show ROC plot
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Naive Bayes ROC Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import (
    accuracy_score, confusion_matrix, precision_score, recall_score,
    f1_score, fbeta_score, roc_curve, auc
)

def evaluate_gradient_boosting(file_path, label):
    df = pd.read_excel(file_path)
    X = df.drop(columns=['earthquake?'])
    y = df['earthquake?']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
        random_state=42)
    model = GradientBoostingClassifier(n_estimators=50, random_state=42)
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]  # Probabilities for ROC

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    f2 = fbeta_score(y_test, y_pred, beta=2)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

    fpr, tpr, _ = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)

    # Plot ROC curve
    plt.plot(fpr, tpr, label=f'{label} (AUC = {roc_auc:.2f})')
```

```python

    return {
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall": recall,
        "F1 Score": f1,
        "F2 Score": f2,
        "True Positive Rate": tp / (tp + fn) if (tp + fn) > 0 else 0,
        "False Positive Rate": fp / (fp + tn) if (fp + tn) > 0 else 0,
        "True Negative Rate": tn / (tn + fp) if (tn + fp) > 0 else 0,
        "False Negative Rate": fn / (fn + tp) if (fn + tp) > 0 else 0,
        "Confusion Matrix": confusion_matrix(y_test, y_pred),
        "ROC AUC": roc_auc
    }

# Run for each file and collect results
results_200 = evaluate_gradient_boosting("earthquake_200.xlsx", "200 Accelerations")
results_500 = evaluate_gradient_boosting("earthquake_500.xlsx", "500 Accelerations")
results_1000 = evaluate_gradient_boosting("earthquake_1000.xlsx", "1000 Accelerations")

# Finalize ROC plot
plt.plot([0, 1], [0, 1], 'k--', label='Random Classifier')
plt.title("ROC Curve - Gradient Boosting")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.grid()
plt.tight_layout()
plt.show()

# Print results
print("Results for 200 accelerations:")
print(results_200)
print("\nResults for 500 accelerations:")
print(results_500)
print("\nResults for 1000 accelerations:")
print(results_1000)

import pandas as pd
for file in ["earthquake_200.xlsx", "earthquake_500.xlsx", "earthquake_1000.xlsx"]:
    df = pd.read_excel(file)
    counts = df['earthquake?'].value_counts()
    print(f"\n{file}:")
    print(f" - No earthquake (0): {counts.get(0, 0)}")
```

```python
    print(f" - Earthquake (1): {counts.get(1, 0)}")

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    precision_score,
    recall_score,
    f1_score,
    fbeta_score,
    roc_auc_score,
    roc_curve
)
import matplotlib.pyplot as plt
from lightgbm import LGBMClassifier

# Store curves for plotting later
roc_data = []

def evaluate_lightgbm(file_path, label):
    df = pd.read_excel(file_path)
    X = df.drop(columns=['earthquake?'])
    y = df['earthquake?']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
        random_state=42)

    model = LGBMClassifier(n_estimators=50, random_state=42)
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    f2 = fbeta_score(y_test, y_pred, beta=2)
    auc = roc_auc_score(y_test, y_proba)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

    # Save ROC data
    fpr, tpr, _ = roc_curve(y_test, y_proba)
```

```
    roc_data.append((fpr, tpr, auc, label))

    return {
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall": recall,
        "F1 Score": f1,
        "F2 Score": f2,
        "ROC AUC": auc,
        "True Positive Rate": tp / (tp + fn) if (tp + fn) > 0 else 0,
        "False Positive Rate": fp / (fp + tn) if (fp + tn) > 0 else 0,
        "True Negative Rate": tn / (tn + fp) if (tn + fp) > 0 else 0,
        "False Negative Rate": fn / (fn + tp) if (fn + tp) > 0 else 0,
        "Confusion Matrix": confusion_matrix(y_test, y_pred),
    }

# Run evaluations
results_200 = evaluate_lightgbm("earthquake_200.xlsx", label="200 Accelerations")
results_500 = evaluate_lightgbm("earthquake_500.xlsx", label="500 Accelerations")
results_1000 = evaluate_lightgbm("earthquake_1000.xlsx", label="1000 Accelerations")

# Plot all ROC curves together
plt.figure(figsize=(8, 6))
for fpr, tpr, auc, label in roc_data:
    plt.plot(fpr, tpr, label=f'{label} (AUC = {auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', linewidth=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - LightGBM Models')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()

# Print results
print("\nResults for 200 accelerations:")
print(results_200)
print("\nResults for 500 accelerations:")
print(results_500)
print("\nResults for 1000 accelerations:")
print(results_1000)

import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score, confusion_matrix, precision_score, recall_score,
    f1_score, roc_auc_score, roc_curve
)
import matplotlib.pyplot as plt

def evaluate_random_forest(file_path, label):
    # Load the dataset
    df = pd.read_excel(file_path)
    X = df.drop(columns=['earthquake?'])
    y = df['earthquake?']

    # Split into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35,
        random_state=42)

    # Initialize the Random Forest model and fit it to the data
    model = RandomForestClassifier(random_state=42)
    model.fit(X_train, y_train)

    # Predictions and probabilities for ROC curve
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]

    # Metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_proba)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

    # ROC curve data
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    plt.plot(fpr, tpr, label=f"{label} (AUC = {auc:.2f})")

    return {
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall (True Positive Rate)": recall,
        "F1 Score": f1,
        "ROC AUC": auc,
        "True Positive Rate": tp / (tp + fn) if (tp + fn) > 0 else 0,
```

```python
        "False Positive Rate": fp / (fp + tn) if (fp + tn) > 0 else 0,
        "True Negative Rate": tn / (tn + fp) if (tn + fp) > 0 else 0,
        "False Negative Rate": fn / (fn + tp) if (fn + tp) > 0 else 0,
    }

# Evaluate and collect results
plt.figure(figsize=(8, 6))

results_200 = evaluate_random_forest("earthquake_200.xlsx", label="200 Accelerations")
results_500 = evaluate_random_forest("earthquake_500.xlsx", label="500 Accelerations")
results_1000 = evaluate_random_forest("earthquake_1000.xlsx", label="1000 Accelerations"
    )

# Plot settings
plt.plot([0, 1], [0, 1], 'k--', linewidth=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()

# Print results
print("\nResults for 200 accelerations:")
print(results_200)

print("\nResults for 500 accelerations:")
print(results_500)

print("\nResults for 1000 accelerations:")
print(results_1000)

import pandas as pd

# Input and output file mappings
files = {
    "earthquake_200.xlsx": "earthquake_200.xlsx",
    "earthquake_500.xlsx": "earthquake_500.xlsx",
    "earthquake_1000.xlsx": "earthquake_1000.xlsx"
}

for input_file, output_file in files.items():
    # Load the Excel file with headers
```

```python
    df = pd.read_excel(input_file)

    # Separate features and label
    feature_columns = df.columns[:-1]         # All columns except the last one
    label_column = df.columns[-1]             # The 'earthquake?' column

    # Take absolute value of only the acceleration features
    df[feature_columns] = df[feature_columns].abs()

    # Save the result to a new Excel file
    df.to_excel(output_file, index=False)
    print(f"Saved: {output_file}")

#This is to ensure all the results are positive
import pandas as pd
import joblib
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

def save_random_forest_model(file_path, model_save_path):
    # Load dataset
    df = pd.read_excel(file_path)
    X = df.drop(columns=['earthquake?'])
    y = df['earthquake?']

    # Train/test split
    X_train, _, y_train, _ = train_test_split(X, y, test_size=0.35, random_state=42)

    # Train model
    model = RandomForestClassifier(random_state=42)
    model.fit(X_train, y_train)

    # Save trained model
    joblib.dump(model, model_save_path)

# Save the model trained on 200 acceleration readings
save_random_forest_model("earthquake_200.xlsx", "random_forest_200.pkl")

import pandas as pd
import joblib

# Load the trained Random Forest model
model = joblib.load("random_forest_200.pkl")

```

```python
# Load the 200-acceleration dataset (used to train this model)
data = pd.read_excel("earthquake_200.xlsx")

# Filter to get only false alarms (earthquake? == 0)
false_alarms = data[data["earthquake?"] == 0]

# Check if there are any false alarms
if false_alarms.empty:
    print("No false alarm samples found in the dataset.")
else:
    # Select the first false alarm sample (drop the 'earthquake?' column)
    sample = false_alarms.iloc[0, :-1].values

    # Convert to DataFrame (model expects 2D input)
    new_data = pd.DataFrame([sample])

    # Make prediction
    prediction = model.predict(new_data)[0]

    print(f"Prediction: {prediction} (0 = False Alarm, 1 = Earthquake)")
```

ملخص المشروع باللغة العربية

محطة زلازل لاسلكية

يهدف هذا المشروع إلى تطوير محطة زلازل ذكية منخفضة التكلفة تعتمد على تقنيات إنترنت الأشياء، للكشف عن النشاط الزلزالي في الوقت الفعلي. تستخدم المحطة مستشعر تسارع لقياس الاهتزازات الأرضية، إلى جانب وحدة اتصال لاسلكي (واي-فاي) لنقل البيانات مباشرة إلى الخادم. يتميز النظام بكفاءة عالية في استهلاك الطاقة، حيث يتم تنشيطه فقط عند استشعار اهتزازات تتجاوز عتبة معينة، مما يجعله مناسباً للتطبيقات في المواقع النائية أو التي يصعب الوصول إليها. يركز المشروع على دقة الكشف الزلزالي، وسرعة الاستجابة في نقل البيانات، وتحقيق كفاءة الطاقة، وذلك من خلال تصميم مبسط قائم على استخدام أقل قدر ممكن من الموارد لتحقيق نتائج دقيقة وموثوقة.

جامعة أكتوبر للعلوم الحديثة والآداب

كلية الهندسة

قسم هندسة نظم الحاسبات

محطة زلازل لاسلكية

مشروع تخرج مقدم ضمن متطلبات درجة البكالوريوس

في

هندسة نظم الحاسبات

الجزء الثاني

إعداد

محمد وليد محمد عبدالفتاح ٢٠٣٦٥٩

عبدالرحمن السيد عبدالمحسن سلام ٢١٢٤٢١

إشراف

دكتور إيهاب صلاح الدين عوض

٢٠٢٤-٢٠٢٥