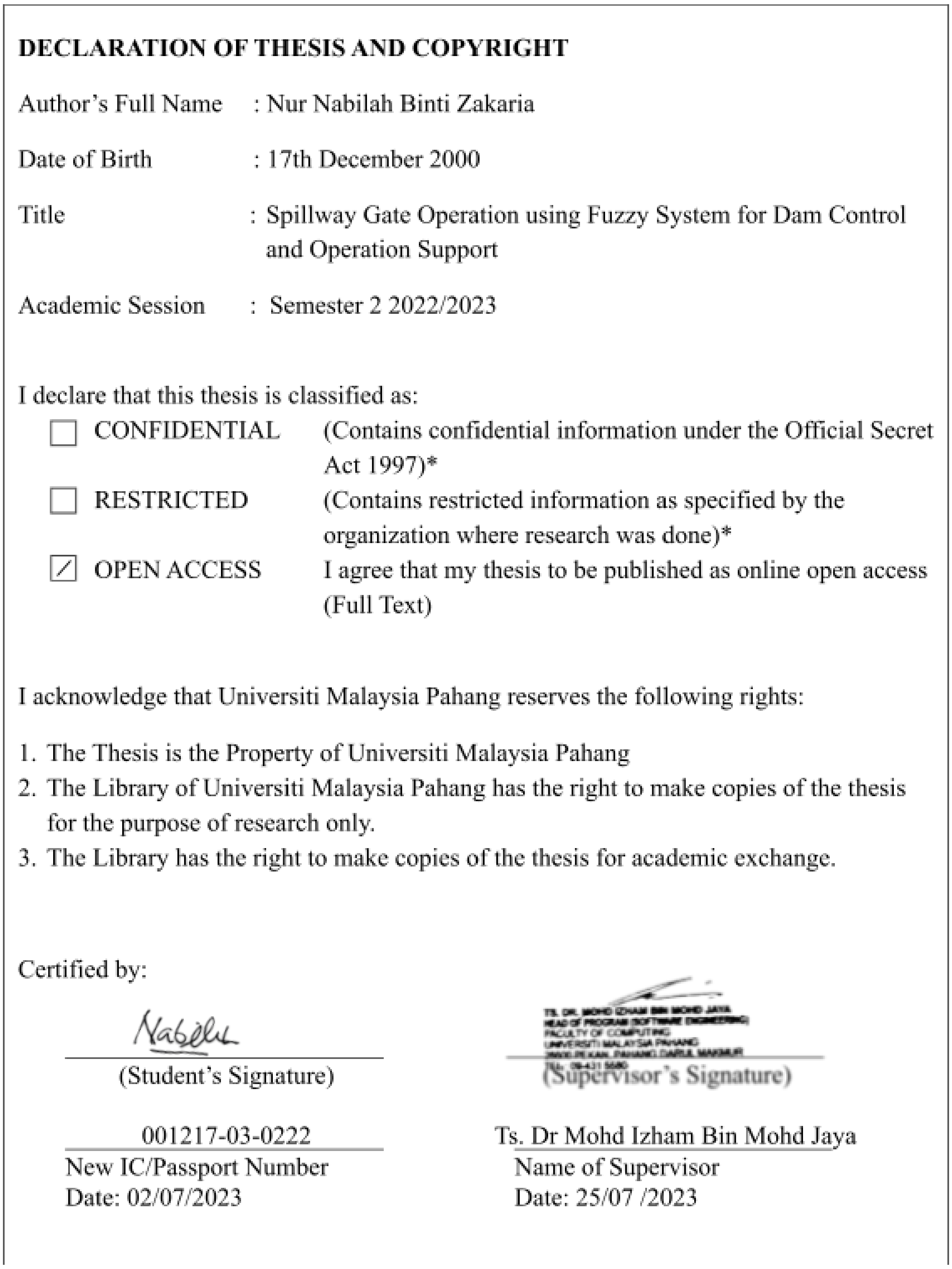
SPILLWAY GATE OPERATION USING FUZZY SYSTEM FOR DAM CONTROL AND OPERATION SUPPORT

NUR NABILAH BINTI ZAKARIA

BACHELOR OF COMPUTER SCIENCE (SOFTWARE ENGINEERING) WITH HONORS

UNIVERSITI MALAYSIA PAHANG

UNIVERSITI MALAYSIA PAHANG



DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name	: Nur Nabilah Binti Zakaria
Date of Birth	: 17th December 2000
Title	: Spillway Gate Operation using Fuzzy System for Dam Control and Operation Support
Academic Session	: Semester 2 2022/2023

I declare that this thesis is classified as:

☐	CONFIDENTIAL	(Contains confidential information under the Official Secret Act 1997)*
☐	RESTRICTED	(Contains restricted information as specified by the organization where research was done)*
✓	OPEN ACCESS	I agree that my thesis to be published as online open access (Full Text)

I acknowledge that Universiti Malaysia Pahang reserves the following rights:

1. The Thesis is the Property of Universiti Malaysia Pahang
2. The Library of Universiti Malaysia Pahang has the right to make copies of the thesis for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by:

[illegible]	[illegible] TS. DR. MOHD IZHAM BIN MOHD JAYA HEAD OF PROGRAM (SOFTWARE ENGINEERING) FACULTY OF COMPUTING UNIVERSITI MALAYSIA PAHANG 26600 PEKAN, PAHANG DARUL MAKMUR TEL: 09-431 5680
(Student's Signature)	(Supervisor's Signature)
001217-03-0222	Ts. Dr Mohd Izham Bin Mohd Jaya
New IC/Passport Number	Name of Supervisor
Date: 02/07/2023	Date: 25/07 /2023

NOTE : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.

THESIS DECLARATION LETTER (OPTIONAL)

Librarian,
Perpustakaan Universiti Malaysia Pahang,
Universiti Malaysia Pahang,
Lebuhraya Tun Razak,
26300, Gambang, Kuantan.

Dear Sir,

CLASSIFICATION OF THESIS AS RESTRICTED

Please be informed that the following thesis is classified as RESTRICTED for a period of three (3) years from the date of this letter. The reasons for this classification are as listed below.

Author's Name	
Thesis Title	
Reasons	(i)
	(ii)
	(iii)

Thank you.

Yours faithfully,

[illegible]

TS. DR. MOHD IZHAM BIN MOHD JAYA
HEAD OF PROGRAM (SOFTWARE ENGINEERING)
FACULTY OF COMPUTING
UNIVERSITI MALAYSIA PAHANG
26600 PEKAN, PAHANG DARUL MAKMUR
TEL : 09-431 5680

(Supervisor's Signature)

Date: 27th July 2023

Stamp:

Note: This letter should be written by the supervisor, addressed to the Librarian, *Perpustakaan Universiti Malaysia Pahang* with its copy attached to the thesis.

Universiti Malaysia PAHANG

Engineering • Technology • Creativity

SUPERVISOR's DECLARATION

I/We* hereby declare that I/We* have checked this thesis/project* and in my/our* opinion, this thesis/project* is adequate in terms of scope and quality for the award of the degree of Bachelor of Computer Science (Software Engineering) with Honors.

[illegible]

TS. DR. MOHD IZHAM BIN MOHD JAYA
HEAD OF PROGRAM (SOFTWARE ENGINEERING)
FACULTY OF COMPUTING
UNIVERSITI MALAYSIA PAHANG
26600 PEKAN, PAHANG DARUL MAKMUR
TEL : 09-431 5580

(Supervisor's Signature)

Full Name	: Ts. Dr Mohd Izham Bin Mohd Jaya
Position	: Head of Program (Software Engineering)
Date	: 25th July 2023

(Co-supervisor's Signature)

Full Name	:
Position	:
Date	:

STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Universiti Malaysia Pahang or any other institutions.

[illegible]

(Student's Signature)

Full Name	: NUR NABILAH BINTI ZAKARIA
ID Number	: CB19044
Date	: 02nd JULY 2023

SPILLWAY GATE OPERATION USING FUZZY SYSTEM FOR DAM CONTROL AND OPERATION SUPPORT

NUR NABILAH BINTI ZAKARIA

Thesis submitted in fulfillment of the requirements
for the award of the degree of
Bachelor of Computer Science (Software Engineering) with Honors

Faculty of Computing

UNIVERSITI MALAYSIA PAHANG

JULY 2023

ACKNOWLEDGEMENTS

All forms of praise and thanks are due to Allah, the creator of mankind, the most merciful and gracious for his blessings, protections, courage and guidance.

I would like to express my heartfelt gratitude and appreciation to the following individuals and organisations who have contributed to the completion of this thesis. First, I extend my sincere thanks to my supervisor, Ts. Dr Mohd Izham Bin Mohd Jaya, for their invaluable guidance, expertise, and continuous encouragement. His insightful feedback and unwavering support have been invaluable in shaping this research work.

I would like to express my gratitude to my family, especially my mother for their unwavering love, encouragement, and support throughout this journey. Their belief in me and their sacrifices have been a constant source of motivation and inspiration. I am thankful to my friends for their companionship, stimulating discussions, and moral support. Their friendship has made this academic pursuit more enjoyable and rewarding.

Last but not least, I acknowledge the contributions of all the authors, researchers, and scholars whose works I have referenced in this thesis. Their contributions have shaped my understanding and provided a solid foundation for this research.

ABSTRAK

Operasi pintu limpahan (spillway) adalah aspek penting dalam mengawal dan menyokong operasi empangan, memastikan pengurusan aliran air yang cekap dan mencegah potensi risiko seperti banjir. Dalam kajian ini, pendekatan sistem kabur (fuzzy) dicadangkan untuk meningkatkan kawalan dan pengendalian pintu air limpah di empangan. Objektif penyelidikan ini adalah untuk membangunkan sistem berasaskan logik kabur yang dapat menentukan dengan tepat pembukaan dan penutupan pintu tumpahan yang optimum berdasarkan pelbagai parameter input. Logik kabur membolehkan perwakilan dan pengendalian maklumat yang tidak tepat dan tidak pasti, menjadikannya sesuai untuk menangani kerumitan operasi empangan. Sistem kabur yang dicadangkan menggunakan input seperti paras air takungan, kadar aliran air dan kelebatan hujan untuk mengira tindakan pembukaan atau penutupan pintu yang sesuai. Keputusan akan dinilai berdasarkan kriteria seperti kecekapan pintu limpahan dan kawalan paras air.

ABSTRACT

Spillway gate operation is a critical aspect of dam control and operation support, ensuring the efficient management of water flow and preventing potential risks such as flooding. In this study, a fuzzy system approach is proposed to enhance the control and operation of spillway gates in dams. The objective of this research is to develop a fuzzy logic-based system that can accurately determine the optimal opening and closing of spillway gates based on various input parameters. Fuzzy logic enables the representation and handling of imprecise and uncertain information, making it suitable for dealing with the complexities of dam operations. The proposed fuzzy system utilizes inputs such as reservoir water level, flow rate, and rainfall to calculate the appropriate gate opening or closing actions. The results will be evaluated based on criteria such as spillway gate efficiency and water level control.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
AI	Artificial Intelligence
MATLAB	Matrix Laboratory
MySQL	My Structured Query Language
LED	Light-Emitting Diode
LAN	Local Area Network
FLC	Fuzzy logic Controller
API	Application Programming Interface
FTP	File Transfer Protocol

CHAPTER 1

INTRODUCTION

1.1 Introduction

A greater emphasis has been placed in recent years on the creation of effective and trustworthy monitoring systems for dam infrastructure. Dams are essential for managing water resources, preventing flooding, and producing hydropower. However, they are also susceptible to natural calamities like heavy rain, which can cause a water overflow and endanger the structural integrity of the entire building. Smart monitoring system deployment has drawn a lot of interest in order to reduce these dangers (V. Ramani Bai & Mohamad Rom Tamjis, 2007; Imran et al., 2014; Mijovic et al., 2016).

Three sensors will be used by this system: a rain gauge, a water level sensor, and a flow sensor. This system can continually gather real-time data, giving important insights into the current condition of the dam, by combining rain gauges, water float sensors, and flow sensors. The use of rain gauges makes it possible to quantify rainfall intensity accurately, providing crucial data for determining the potential influence on dam operations. With the use of water float sensors, the water level inside the dam may be precisely monitored, allowing for the early identification of any unusual variations or dangers (M. Abbas et al., 2011; Dhandre et al., 2016; V. Sathya et al., 2019). Furthermore, flow sensors are essential for detecting the amount and rate of water flowing through the dam, which helps with efficient management and control.

MQTT and WebSocket are included in the suggested smart dam monitoring system to increase functionality (Mijovic et al., 2016). MQTT offers lightweight real-time data transmission by enabling efficient and dependable communication between sensors, microcontrollers, and the MQTT broker (Eridani & Eko Didik Widianto, 2018). Instant updates and real-time visualization are made possible via

WebSocket, which create a bidirectional link between the web server and user interface (Mijovic et al., 2016). Users may now monitor the dam's state, manage actuators, and quickly get notifications thanks to this integration (Dong et al., 2017). MQTT and WebSocket working together improve data transmission, seamless integration, and user experience, eventually enhancing dam performance and safety.

The spillway gate of the dam will operate as the system's actuator and can be controlled using data gathered from sensors and fuzzy logic-based classification. The spillway gate actuator optimizes the dam's operations and ensures optimal management of water resources by enabling efficient regulation of water outflow. This smart dam monitoring system will use a classification strategy based on the gathered data. The system can manage ambiguous and imprecise information thanks to fuzzy logic, which makes predictions and classifications more exact. The system can offer preventative actions and prompt notifications by introducing fuzzy logic into the decision-making process, which contributes to improved dam safety and performance.

1.2 Problem Statement

There is a serious issue with the monitoring and management of dams in the area of dam infrastructure. Data gathering process without sensors in dam monitoring causes inefficiencies, delays, and mistakes when obtaining crucial data concerning elements like rainfall intensity, water levels, and water movement inside the dam (V. Ramani Bai & Mohamad Rom Tamjis, 2007); Imran et al., 2014). This issue presents a significant obstacle because it makes it difficult to evaluate the dam's existing condition, recognise possible threats, and put preventive measures in place in a timely manner.

The lack of effective and reliable dam monitoring systems is a critical problem that needs to be solved. Dam safety is crucial for safeguarding people, property, and the environment. Dam operators and other related parties have a tough time getting the timely, accurate, and thorough data needed for proactive decision-making and efficient risk management without a reliable monitoring system in place (Karaboga et al., 2008). In regions where there are frequent extreme weather events, this situation is made worse because there is a higher risk of dam failures and their related effects.

1.3 Objective

The objectives of this project are:

I. To collect the requirement for the development of a spillway gate operation using a fuzzy system for dam control and operation support based on real-time dam data.

II. To develop spillway gate operation using a fuzzy system for dam control and operation support that utilizes all components in the IoT architecture and fuzzy logic-based classification methods that are capable of producing precise classifications pertaining to the security and possible dangers of the dam.

III. To evaluate the efficacy of the developed spillway gate operation using a fuzzy system for dam control and operation support.

1.4 Scope

The scope of the project consists of user, system, and development. The scopes are specified below:

User Scope:

I. The operator of the dam in Kuantan, Pahang.

System Scope:

I. Use sensors to collect the data which are the volume and intensity of the rain, the level of the dam water, and the speed of the dam flow.
II. Provide alerts when the level of the dam water reaches a certain level.
III. Provide precise classification skills, improving the performance and safety of dams using fuzzy logic.

IV. Utilize bidirectional connectivity and efficient real-time data transmission through the use of communication protocols like MQTT and WebSocket.

Development Scope:

I. The system will be developed using Visual Studio Code, Laravel framework, and MySQL as the database server.
II. The system incorporates servers for MQTT, WebSocket, Web, and AI, enabling efficient data transmission, real-time communication, user interaction, and intelligent analysis.

1.5 Thesis Organization

This thesis consists of five chapters. Chapter 1 discusses the introduction to the project which are the introduction, problem statement, objectives, scope, and thesis organization.

Chapter 2 provides a quick overview of the literature study on the existing dam monitoring systems.

Chapter 3 covers all the designs which are the architecture design, dashboard design, and analytic feature design of the system.

Chapter 4 explains the result and the discussion of the system. The chapter covers the implementation of the dashboard and the fuzzy logic.

Lastly, Chapter 5 concludes all the chapters about the system.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter provides a thorough analysis of the body of research on the smart dam monitoring system. It provides a thorough review of earlier research, methods, hardware, and technologies that are pertinent to the suggested solution. This chapter tries to evaluate the present state of knowledge, emphasize the problem or solutions, and identify the gaps that the proposed project hopes to fill by addressing the "What, Why, and How" of the research. In addition, the literature review will assess the relevant hardware, software, and tools, compare and contrast at least three existing systems, and examine the merits and drawbacks of the project-based methodology.

2.2 Existing Systems/Works

This section will analyze various past systems and techniques for smart dam monitoring systems. Understanding the design, implementation, and performance of these systems will be the main emphasis of the review. Data collection methods, monitoring strategies, data processing algorithms, and decision-making processes will all be taken into account during the analysis. There will be significant insights acquired to enhance the suggested solution by analyzing the advantages and disadvantages of various systems**.**

2.2.1 Controlling Spillway Gates of Dams by using Fuzzy Logic Controller with Optimum Rule Number

The control of the reservoir functioning in a dam is the primary objective of this system (Karaboga et al., 2008). The system operates according to standard operating principles that are controlled by people, which might result in abrupt changes in the released water and undesirable variations in the level of the reservoir. It is challenging to build a precise reservoir operating model based on inflow/outflow hydrographs due to the complexity and nonlinearity of the hydrological circumstances. Numerous approaches and models, such as deterministic operating procedures, linear programming, goal-programming models, dynamic programming, prediction tools based on neural networks, and the fusion of genetic algorithms and fuzzy inference systems, have been put forth in the literature to address this problem. These traditional control methods, however, might not be sufficient to offer the best solutions for complicated and uncertain systems. Expert systems and fuzzy logic-based control are two artificial intelligence techniques that could be used to increase the adaptability and flexibility of control systems. Figure 2.1 shows the structure of the fuzzy logic-based control system.

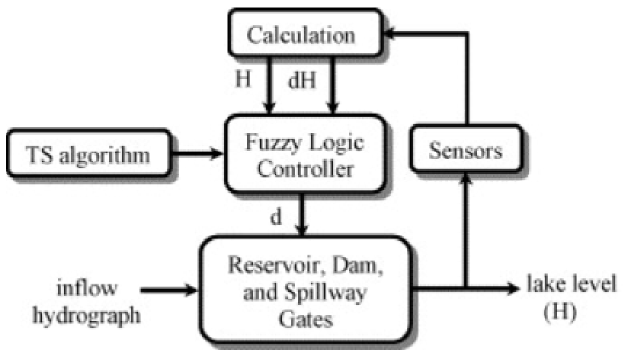


Figure 2.1: The structure of the fuzzy logic-based control system

2.2.2 Fuzzy Logic Based Hydro-Electric Power Dam Control System

This existing system demonstrates how fuzzy sets and fuzzy logic are used in a variety of fields, such as information storage and retrieval, web search, image processing, control, pattern recognition, bioinformatics, e-markets, autonomous navigation, and guidance. Fuzzy sets are being included in computing systems in an effort to make them more intelligent, autonomous, and flexible (M. Abbas et al., 2011). Fuzzy logic has been used in the context of dam control to create control systems based on measurements of the water level and flow rate. To control the operation of drain valves and release valves in a hydroelectric power dam, the suggested system makes use of fuzzy rules. The arrangement of the proposed hydroelectric power system for the journal is shown in Figure 2.2 below.

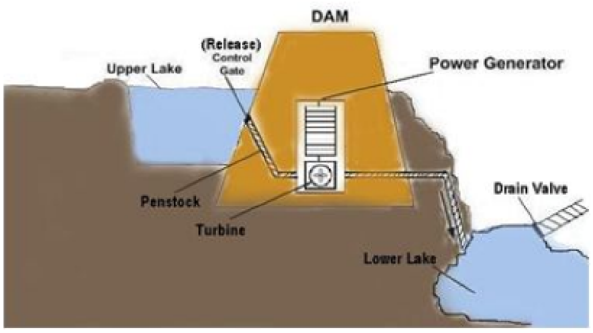


Figure 2.2: Arrangement of proposed hydro-electric power system

2.2.3 Fuzzy Logic Model on Operation and Control of Hydro-Power Dams in Malaysia

Using fuzzy logic-based modeling, this existing system focuses on ensuring reservoirs run as efficiently as possible. Fuzzy logic offers a straightforward method for converting verbal descriptions into numerical values, enabling the creation of reservoir operation rules. Applications involving water resources have used fuzzy logic, frequently as an improvement over traditional optimisation methods. This system's goal is to generate the macro-level optimal reservoir operating rules for better performance and control (V. Ramani Bai & Mohamad Rom Tamjis, 2007). Numerous research studies that take into account stochastic models, storage non-specificity, and competing goals including flood management, irrigation, and power generation have used fuzzy logic in reservoir operation (V. Ramani Bai & Mohamad Rom Tamjis, 2007). For dam control, the use of fuzzy dynamic programming and neural network systems has also been investigated. Figure 2.3 below shows the fuzzy inference system of this system.

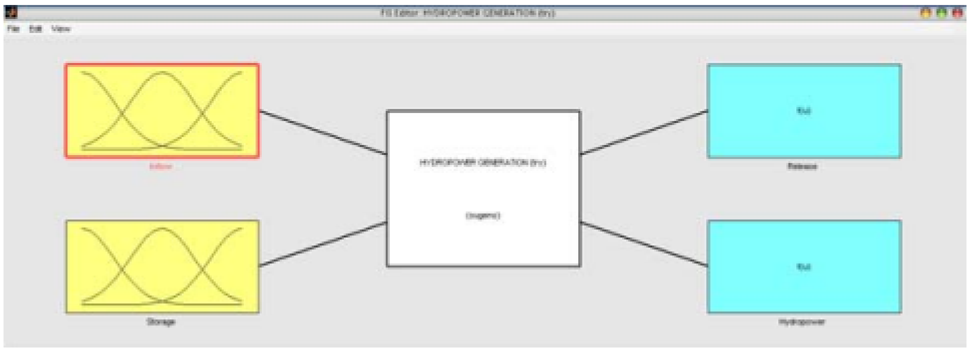

FIS Editor: HYDROPOWER GENERATION (try)
File
Edit
View
Inflow
Storage
HYDROPOWER GENERATION (try)
(sugeno)
f(u)
Release
f(u)
Hydropower

Figure 2.3: Fuzzy Inference System

2.3 Analysis/ Comparison of Existing System

Analysis and comparison of current smart dam monitoring systems require evaluating a number of factors, including techniques/methods, tools/technology, as well as the positive and negative aspects of each approach. This review helps in determining the most successful methods and tools for monitoring and offers insightful information about the various approaches used in the field of dam monitoring. This analysis intends to enhance knowledge of methodologies and technologies available, enabling the development of more reliable and effective smart dam monitoring systems. It does this by analyzing the strengths and shortcomings of existing systems. This review also aids in identifying potential issues and problems that must be resolved to improve the overall effectiveness and dependability of dam monitoring systems.

2.3.1 Analysis of comparison on existing systems

The comparison of existing systems for smart dam monitoring, as presented in Table 2.1, evaluates different techniques/methods and tools/technology used.

Table 2.1 : Table of comparison on existing systems

Title/ Criteria	**Controlling Spillway Gates Of Dams By Using Fuzzy Logic Controller With Optimum Rule Number**	**Fuzzy Logic Based Hydro-Electric Power Dam Control System**	**Fuzzy Logic Model On Operation And Control Of Hydro-Power Dams In Malaysia**
System Features	● Spillway gate control ● Optimum rule number	● Water flow regulation ● Power generation optimization ● Turbine control ● Coordination of multiple units/generators	● Adaptive control algorithms ● Real-time monitoring ● Fault detection ● System optimization
Technique/ Method	Fuzzy logic	Fuzzy logic	Fuzzy logic
Tools/ Technology	Fuzzy Logic Toolbox in MATLAB	Fuzzy Logic Toolbox in MATLAB	Fuzzy Logic Toolbox in MATLAB, real-time data monitoring systems

2.3.2 Relevance of comparison with project title

Table 2.2 : Table of strengths and weaknesses on existing systems

System/ Comparison	Strengths	Weaknesses
Controlling Spillway Gates Of Dams By Using Fuzzy Logic Controller With Optimum Rule Number	● Application of fuzzy logic controller (FCL) for complex dam control systems.	● Limited scope of analysis. ● Lack of thorough validation. ● May require a technical background.
Fuzzy Logic Based Hydro-Electric Power Dam Control System	● FCL's effectiveness in handling uncertainties in hydro-electric dam control.	● Limited generalizability. ● Insufficient analysis of robustness and stability. ● Technical assumptions.
Fuzzy Logic Model On Operation And Control Of Hydro-Power Dams In Malaysia	● Focus on the specific context of hydro-power dams in Malaysia. ● Insights into regional challenges and opportunities.	● Limited applicability to other regions. ● Potential lack of extensive validation. ● Assumes familiarity with Malaysian dams.

2.4 Summary

The literature review provides broad knowledge about the important material necessary to comprehend the field of the suggested research. The literature review can validate the strategy of choice and identifies best practices from the current systems. By utilizing its strengths and resolving its faults, this analysis improves the system's performance. Overall, the comparison analysis advances the understanding of the subject.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter presents the overall approach and framework adopted in this thesis, which encompasses four main sections. The system's structure and components are described in the first section, which concentrates on architectural design. It goes over the frameworks, integrations, and technologies used to enable effective communication across various modules or layers. The layout, visual components, and user interaction aspects of the dashboard interface are highlighted in the second segment to create a more user-friendly and intuitive experience. The third section describes the design of the analytical features and explains how to gather, process, and present data to produce insightful information. There has been research into various analytical techniques, algorithms, and visualizations to aid with data analysis and decision-making. (Mijovic et al., 2016; Mijovic et al., 2016). Lastly, the chapter addresses the potential use of the proposed solution, highlighting its advantages and potential impact in real-world scenarios. Overall, this chapter offers a comprehensive overview of the thesis framework, encompassing architecture design, dashboard design, analytic feature design, and the potential use of the proposed solution.

3.2 Flowchart

Figure 3.1 below shows the flowchart of the smart dam monitoring system:

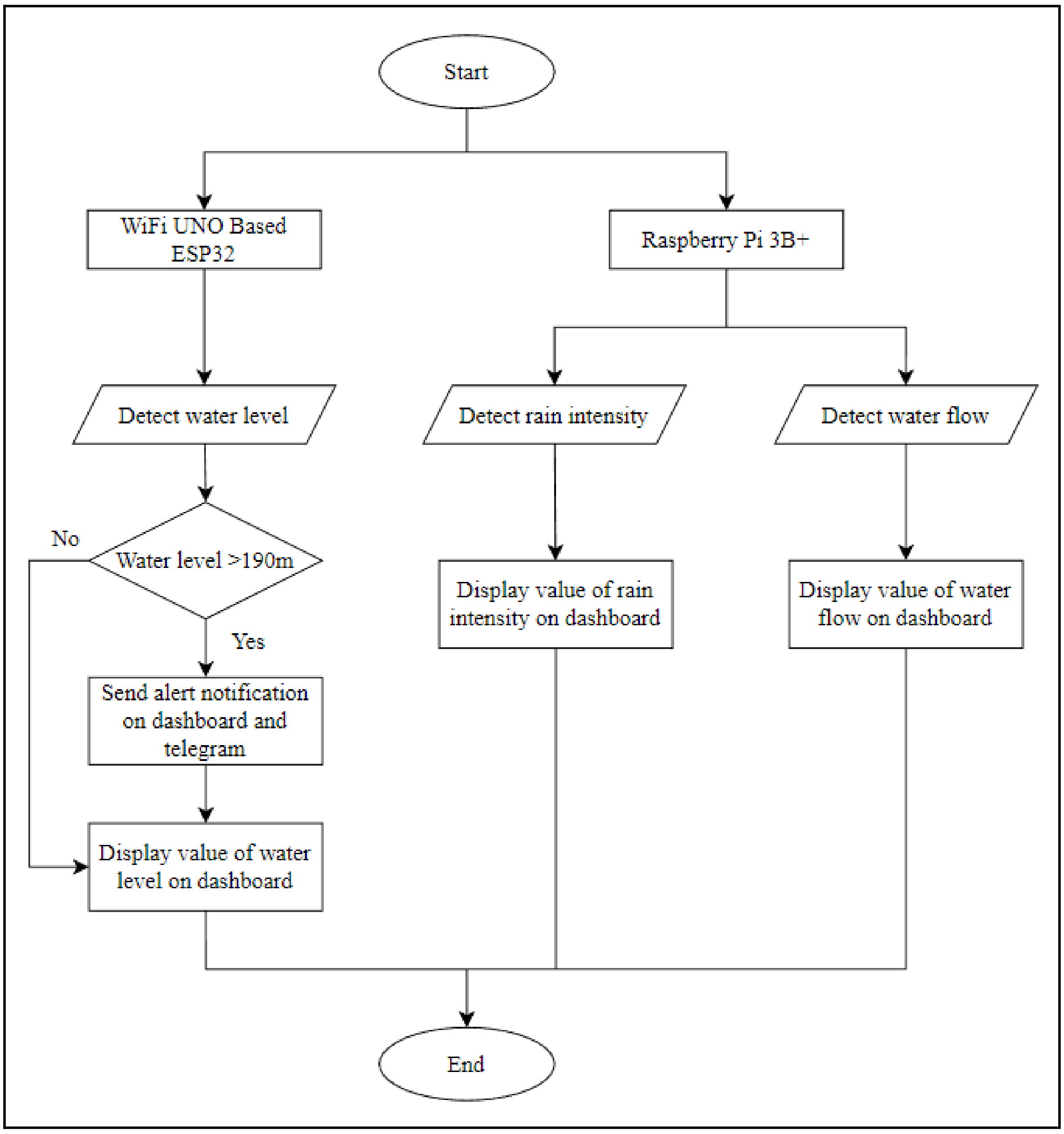

Start
WiFi UNO Based ESP32
Raspberry Pi 3B+
Detect water level
Detect rain intensity
Detect water flow
No
Water level >190m
Yes
Display value of rain intensity on dashboard
Display value of water flow on dashboard
Send alert notification on dashboard and telegram
Display value of water level on dashboard
End

Figure 3.1: Flowchart of Smart Dam Monitoring System

The system uses two microcontrollers: a Raspberry Pi model 3B+ and a WiFi Uno-based ESP32. The ESP32, which is coupled to a water level sensor, is where the flow starts. The amount of water in a dam is continuously monitored by this sensor. An alert signal is delivered to both the dashboard and Telegram when the water level surpasses 190m. The value of the water level is additionally shown on the dashboard. If the water level is lower than 190m, it is only shown on the dashboard and no notifications are sent.

Two sensors are included with the Raspberry Pi 3B+: a water flow sensor and an ultrasonic sensor for determining the intensity of the rain. These sensors continuously keep track of the water flow and rainfall intensity, respectively. When rain happens, the dashboard displays the values of rain intensity and water flow, giving real-time information.

The system offers efficient monitoring and management of water-related parameters by integrating these microcontrollers and sensors, enabling users to be informed about crucial water levels and environmental conditions.

3.3 IoT Architecture

The Internet of Things (IoT), which connects a wide range of devices and allows for seamless data sharing and intelligent decision-making, has emerged as a paradigm-shifting technology. In the world of IoT, the architecture is crucial to the efficient operation of networked systems. The sensor layer, network layer, data processing layer, and application layer are the four basic IoT architecture layers that will be discussed in this section. The overall functioning and communication of IoT systems are improved by each layer. The choice of protocols, databases, and software frameworks used in each tier also has a big impact on the success of IoT implementations. All the explanations on the IoT architecture layers in Figure 3.1 are explained in 3.3.1 for the sensing layer, 3.3.2 for the network layer, 3.3.3 for the data processing layer, and 3.3.4 for the application layer.

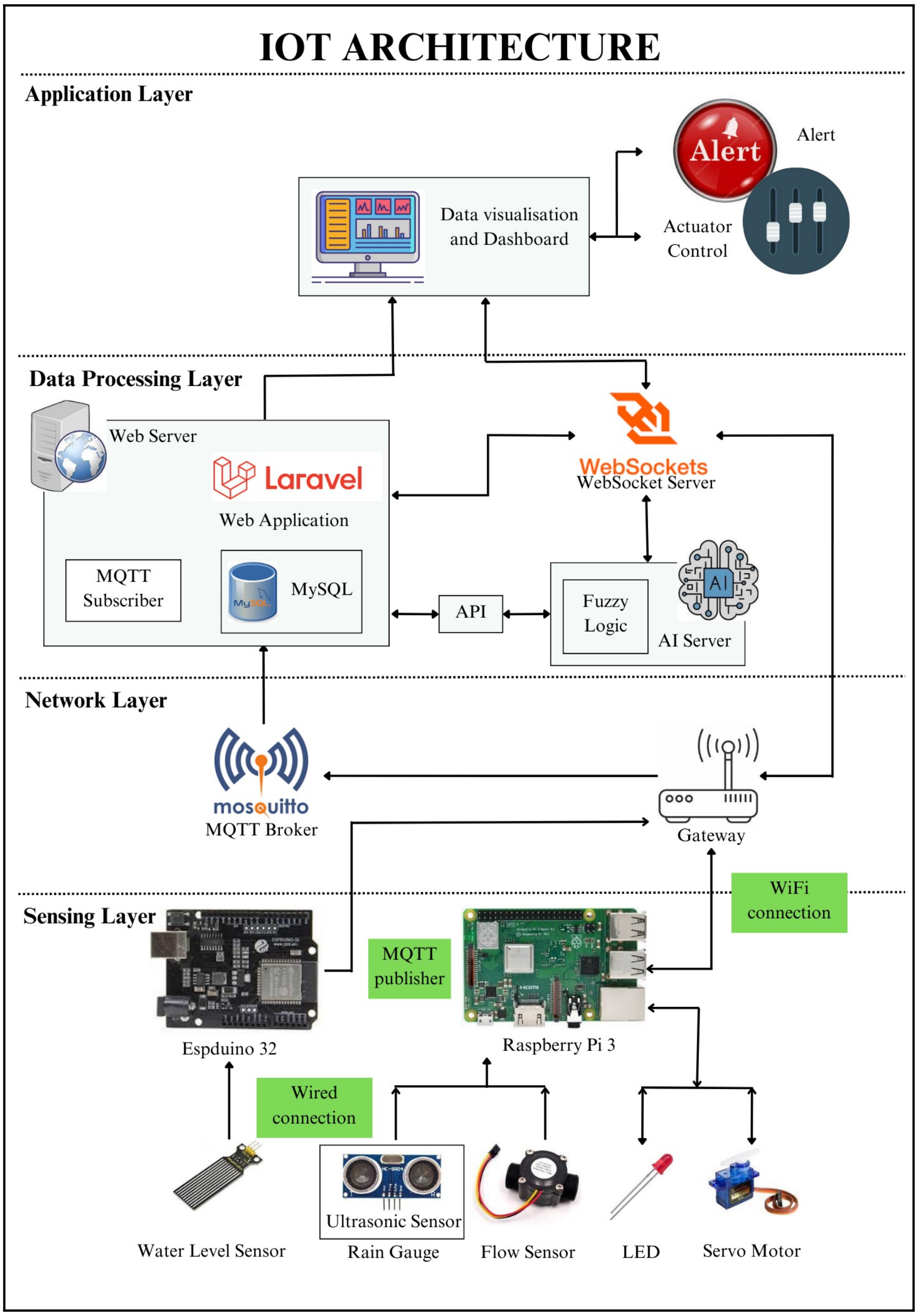


Figure 3.1: IoT Architecture for Smart Dam Monitoring System

3.3.1 Sensing Layer

The sensing layer is the foundational layer in the IoT architecture, housing the hardware components such as sensors and microcontrollers. Three sensors are used in the smart dam monitoring system project: a flow sensor, an ultrasonic sensor, and a water level sensor. The ultrasonic sensor is built into the rain gauge to measure the water level and calculate the amount of rainfall, while the water level sensor is in charge of detecting the dam's water level. Insights on rainfall intensity are provided by the rain gauge based on the volume of water that is collected. On the other hand, the flow sensor quantifies the speed and amount of the dam water.

In terms of connectivity, the water level sensor is connected to anEspduino 32 via a wired connection. A serial connection is established between the Raspberry Pi and the Espduino 32 (Oh Am Suk, 2018). On the other hand, the ultrasonic sensor and flow sensor directly connect to the Raspberry Pi since they are digital sensors. The Raspberry Pi takes on the role of an MQTT publisher, sending data to the MQTT broker through a gateway.

The sensor layer also includes output devices and actuators that are activated in response to certain circumstances. An LED acts as an alert output device in the smart dam monitoring system, illuminating in the event of an alert condition. In addition, the actuator used has a motor that opens and closes the dam gate to ensure proper management of the water flow.

3.3.2 Network Layer

The gateway and MQTT broker are two essential elements of the IoT architecture's network layer. The gateway serves as a bridge between the MQTT broker and the MQTT publisher, such as the Raspberry Pi. In this project, the gateway is connected via a wired connection using a LAN port to create connectivity. Data transmission from the microcontroller to the MQTT broker is handled by the gateway.

As the central hub for data transfer, the MQTT broker is essential to the network layer. When a subscriber subscribes to a certain user and topic, it receives the data from the gateway and preserves it until the subscriber wants it. The MQTT broker holds the data and ensures its delivery to the appropriate subscribers (Eridani & Eko Didik Widianto, 2018).

The gateway and MQTT broker together with the network layer provide effective and dependable data transmission and communication in the IoT system. The MQTT broker stores and distributes data to subscribers per their subscriptions to particular users and topics, while the gateway permits communication between the MQTT publisher and the broker.

3.3.3 Data Processing Layer

All of the data gathered from the sensors is processed in the data processing layer of the IoT architecture. The web server plays a crucial part in this project by using the same topic to subscribe to the MQTT broker and retrieve the data. The obtained information is then kept in a local database, specifically MySQL. After being stored in MySQL, the data is processed using WebSocket and an AI server. To ensure data security, the web server exposes an API that allows the AI server to retrieve the data. The retrieved data is then processed using an AI method which is fuzzy logic. Fuzzy logic plays a vital role in analyzing and interpreting the collected data, enabling intelligent decision-making based on specific circumstances or occurrences. Real-time communication between the data processing layer and the actuators and output devices is facilitated by the use of WebSocket. Based on particular circumstances or occurrences, the processed data is used to activate processes, control the actuators and output devices, or both.

Overall, processing and analyzing the data gathered from sensors rely heavily on the data processing layer. To retrieve the data, the web server subscribes to the MQTT broker. The data is then stored in a MySQL database. WebSocket is used to trigger actions once the data has been further analyzed using AI methods. This enables efficient control and communication with the actuators and output devices in the IoT system.

3.3.4 Application Layer

The IoT architecture's last layer is the application layer. The Laravel framework is used in this layer to create a dashboard that displays all the gathered data. Users can view and manage the actuators and output devices of the system using the dashboard's user-friendly interface. The dashboard also shows alerts to make sure users are immediately alerted of any urgent occurrences or circumstances.

The dashboard provides a detailed view of the data from the IoT system by utilizing the capabilities of the Laravel framework. Users can easily understand charts, graphs, and other visual components that provide sensor data, historical trends, and real-time updates. The dashboard facilitates efficient monitoring and control of the IoT system by empowering users to make well-informed decisions based on the information provided.

The dashboard additionally enables direct user interaction with the actuators and output devices. Through the dashboard's interface, they can initiate actions like opening or closing the dam gate or managing the LED alerts. This interactive functionality improves user interaction and offers a practical way to control the IoT device.

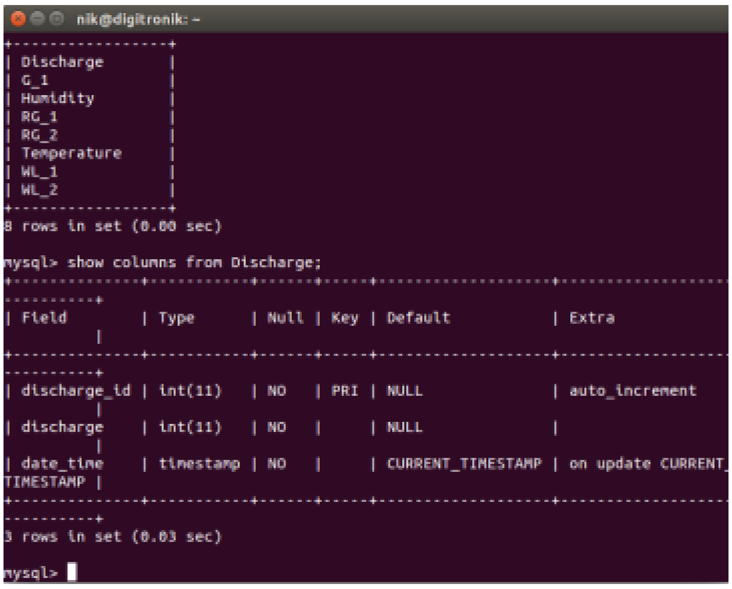
In conclusion, the Laravel framework is included in the application layer of the IoT architecture to provide a feature-rich dashboard. Data is visualized on the dashboard, which also offers real-time warnings and actuator and output device management. It enables people to successfully manage and make decisions about the IoT system by monitoring, analyzing, and interacting with it.

3.4 Data Collection

In the context of the existing system, data collection involves the retrieval of information from sensors installed at the dam site. These sensors produce both analog and digital outputs, providing measurements for various parameters that are relevant to the dam's operation and the surrounding environment (Dhandre et al., 2016; V. Sathya et al., 2019). To facilitate the interaction with these sensors and gather the necessary data, a functional model will be developed. This model serves the purpose of autonomously collecting the data, removing the need for external interventions.

Continuous data collection ensures the acquisition of a comprehensive dataset that accurately represents the dynamic behavior of the dam's variables over time. Specifically, the data being collected in this thesis pertains to water level, flow rate, and rainfall, which serve as input or membership functions for the fuzzy logic analysis of the dam's operations. Continuous data collection enables the analysis of trends, recognition of patterns, and informed decision-making about to the operation, maintenance, and performance of the dam. By leveraging this dataset, valuable insights can be derived to optimize the dam's functionality and ensure its effective management.

Figure 3.3 below provides an example of the database structure from the current system for collecting dam data (Dong et al., 2017). This figure illustrates an example of the database structure, showcasing the organization of data within the smart dam monitoring system. The design of the database for the system being developed in this thesis will follow a similar structure as depicted in the figure. The database for the smart dam monitoring system will include fields to store the essential inputs for the system, which consist of water level, flow rate, and rain intensity data.



```
+------------------+
| Discharge        |
| G_1              |
| Humidity         |
| RG_1             |
| RG_2             |
| Temperature      |
| WL_1             |
| WL_2             |
+------------------+
8 rows in set (0.00 sec)

mysql> show columns from Discharge;
+--------------+-----------+------+-----+-------------------+------------------
----------+
| Field        | Type      | Null | Key | Default           | Extra
          |
+--------------+-----------+------+-----+-------------------+------------------
----------+
| discharge_id | int(11)   | NO   | PRI | NULL              | auto_increment
          |
| discharge    | int(11)   | NO   |     | NULL              |
          |
| date_time    | timestamp | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_
TIMESTAMP |
+--------------+-----------+------+-----+-------------------+------------------
----------+
3 rows in set (0.03 sec)

mysql> 
```

Figure 3.3: Database Structure from Existing System

3.5 Dashboard Design

As the main interface through which users interact with and monitor the system's data and operations, the dashboard design is an important factor to take into account while creating a smart dam monitoring system. The overall user experience is substantially impacted by the dashboard's aesthetic appeal and usability. This section will cover the suggested dashboard layout for the smart dam monitoring system, with an emphasis on developing an interface that is both aesthetically pleasing and simple to use. The design will have a simple, contemporary user interface and a color palette that reflects the branding of the system. The design will be clean and simple, with an emphasis on the presentation of important parameters relating to the status of the dam. It will also provide real-time updates and notifications. To make data analysis and decision-making easier, interactive data visualization tools will be incorporated. The suggested dashboard intends to improve the management and monitoring of the smart dam monitoring system through effective information presentation and user engagement provided by an intuitive design. Figure 3.4 below shows the wireframe dashboard of the system.

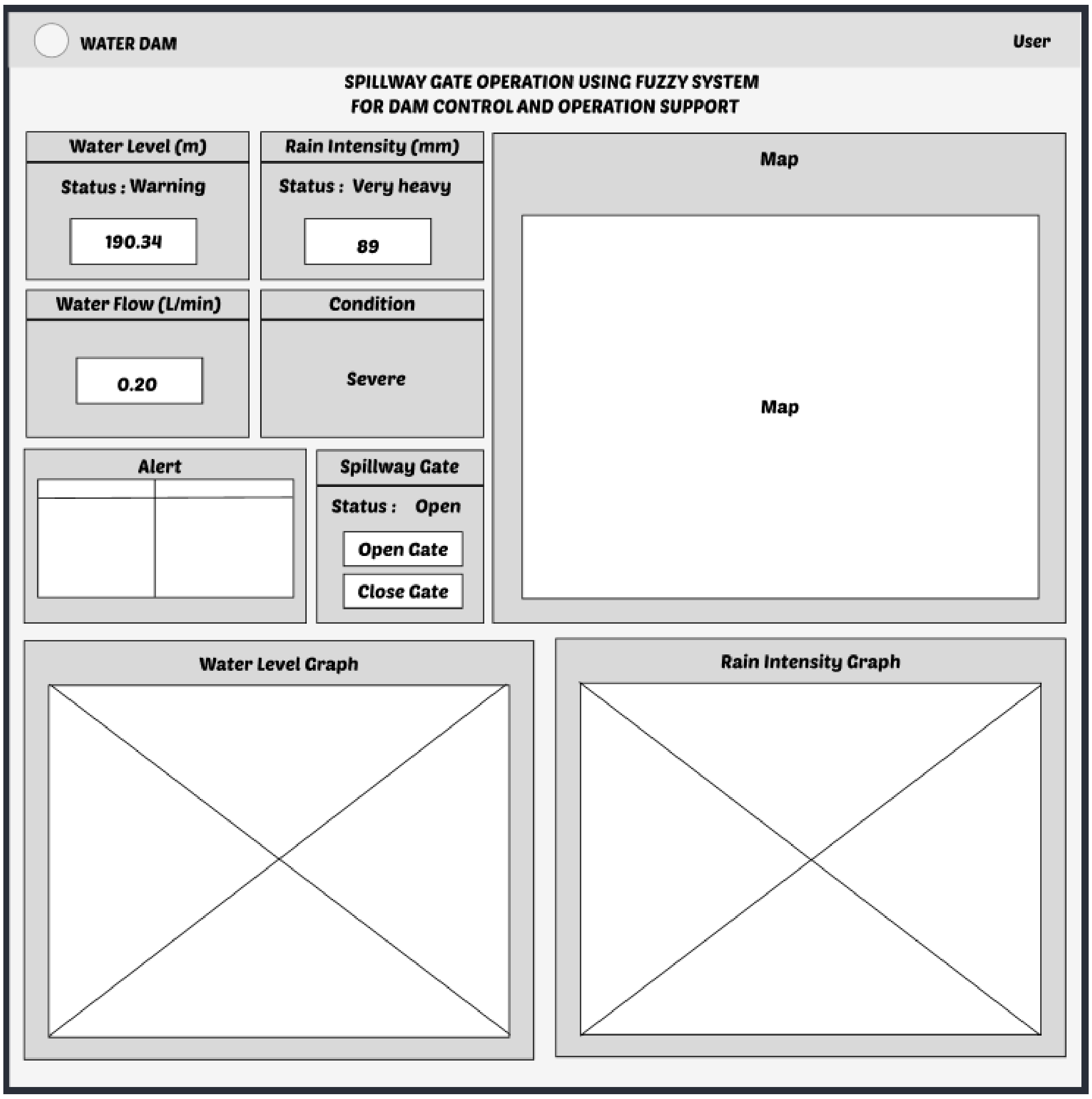


Figure 3.4: Wireframe Dashboard of Smart Dam Monitoring System

The dashboard acts as the system's main interface and gives users a thorough overview of the water management procedure. The title of the system is prominently displayed, providing background information for the data being displayed. The dashboard's numerous sections each display current values gleaned from various sensors. The water level (measured in meters), the amount of rain (measured in millimeters), and the water flow (measured in liters per minute) are all displayed in the first section. These values enable users to keep an eye on the situation and make wise decisions.

A quick evaluation of the situation is made possible by the visible display of the water level status and rain intensity. The dashboard's usage of a map element to determine the dam's precise location improves users' spatial awareness of the system. The dashboard uses fuzzy logic principles to evaluate the situation in order to further assist users in analyzing the facts. Based on the results of the fuzzy rules, the current situation is classified as normal, moderate, or severe. A succinct description of the whole situation is provided by this classification.

An alert is generated and clearly displayed on the dashboard if the water level rises to the warning level. The spillway gate can also be operated manually by users, giving them the choice of opening or closing it by pressing specific buttons. The dashboard includes graphs showing the water level and rain intensity over time to give a visual depiction of the historical trends. Users may examine patterns, spot trends, and make educated predictions thanks to these graphs.

3.5 Analytic Feature Design

The system uses fuzzy logic for analytics and membership functions for rainfall, flow rate, and water level (Dhandre et al., 2016; V. Sathya et al., 2019). The system's fuzzy rules are based on the two key membership functions of water level and rainfall intensity. These activities are crucial in figuring out how the dam is doing overall. The approach divides the dam's condition into three groups—normal, moderate, and severe—by analyzing these variables. On the other hand, the water flow value has a distinct function; it primarily tracks the water flow via the spillway gate of the dam when it is open. This knowledge aids in evaluating the flow of water into and out of the dam. The water flow value does not, however, directly affect the outcomes of the fuzzy rules used to determine the stability of the dam. The fuzzy system's main goal is to ascertain whether or not the dam is in a condition that makes it impossible for it to breach. The safety and structural integrity of the dam structure depend on this classification. The system uses language phrases related to the inputs for water level and rain intensity to make this analysis easier. The system can assess the combined impact of these variables and precisely calculate the right state for the dam thanks to the terms' meticulous definition and mapping to particular ranges of values. Table 3.1 lists the input and output variables for the fuzzy system along with the associated conditions.

Table 3.1: The conditions for input and output variables of the fuzzy logic

	Variables	**Conditions**				**Units**
Input	**Water Level**	**Normal**	**Alert**	**Warning**	**Danger**	**m**
		180 - 185	185 - 190	191 - 200	201 - 210	
	Rain Intensity	**Light**	**Moderate**	**Heavy**	**Very Heavy**	**mm**
		1 - 10	11 - 30	31 - 60	61 - 100	
Output	**Condition**	**Normal**	**Moderate**	**Severe**		**%**
		0 - 40	41 - 75	76 - 100		

3.6 Potential Use of Proposed Solution

There are numerous applications for the suggested real-time smart dam monitoring system. It enables real-time, continuous monitoring of rainfall, flow rate, and water level, allowing dam operators to act quickly. The system may issue alerts when concerns are found and analyze data to provide early warnings. Additionally, it makes remote management of dam operations possible. Real-time data visualization aids in resource optimization and trend identification. Overall, the technology improves early warning capabilities, makes dam management simpler, and allows for remote accessibility for timely decision-making.

CHAPTER 4

IMPLEMENTATION, RESULT AND DISCUSSION

4.1 Introduction

This chapter discusses the Smart Dam Monitoring System's implementation process, which includes hardware and web application development as well as the incorporation of AI techniques. It talks about the system's performance and adherence to the project's objectives as well as the outcomes of the testing and final development assessments. While the web application acts as the user interface, the hardware implementation consists of sensors, microcontrollers, and communication modules. The system's capabilities are enhanced using AI approaches. The testing stage guarantees usability, performance, and function. The effectiveness of the system is assessed using the results, and any problems are dealt with and improved as necessary.

4.2 Implementation

This subchapter offers a thorough explanation of each step required in creating the Smart Dam Monitoring System. These procedures involve the installation of the hardware, the creation of web applications, and the application of AI techniques. To ensure a complete grasp of their application within the broader development lifecycle, each subchapter will focus on the exact stages and procedures carried out during these processes. These specifics will be examined to get a clear and thorough understanding of how the Smart Dam Monitoring System is constructed, from the original hardware setup to the development of the web application and the incorporation of AI techniques.

4.2.1 Hardware Setup

The hardware configuration serves as the Smart Dam Monitoring System's fundamental building block since it includes the sensors required for data retrieval and the actuators required for system control. The succeeding development steps cannot function effectively without a properly configured hardware setup. The accompanying figure shows the finished hardware configuration.

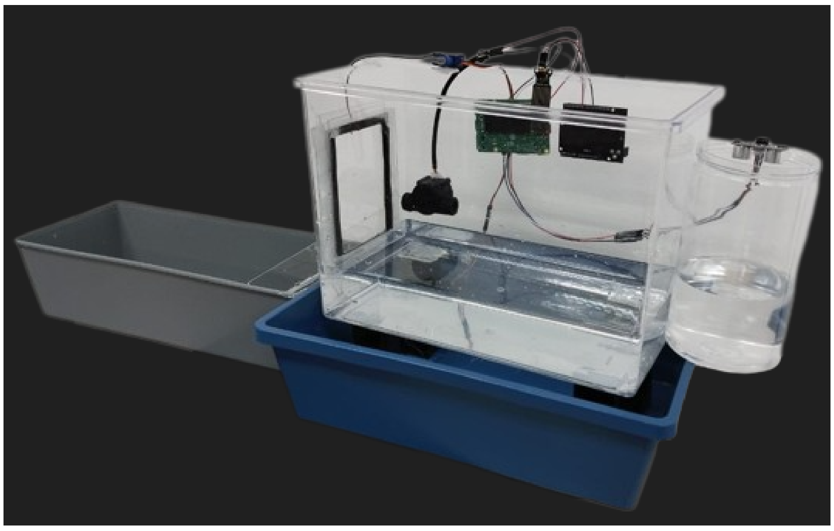


Figure 4.1: The prototype of the Smart Dam Monitoring System

When everything is set up, data transmission from the sensing layer to the data processing layer is implemented. This entails sending data to the database via the MQTT network layer protocol. In addition, actuator control is provided, allowing the application layer to initiate actions that are then communicated back to the sensing layer via the WebSocket network layer protocol. Chapters 3.3.1 through 3.3.3 go into great detail about this technique.

Python programs are developed to carry out these functions. These include the "DAMsensors.py" program for publishing data, the "controlbutton.py" program for monitoring WebSocket events and updating actuator control accordingly, and the "autogateDam.py" program for automating actuators based on fuzzy logic outputs. The Raspberry Pi environment (in VNC Viewer) consistently executes these Python programs. The Appendix contains the complete codes for these programs.

4.2.2 Web Application Development

The development of the web application can start after the hardware setup is complete. The Laravel framework configurations, which are essential to creating the Smart Dam Monitoring System, will be the primary emphasis of this subchapter. The development procedures for each page of the web application as well as the creation of the database and API controllers, routing setups, and event channels will all be covered. The setup of these components within the Laravel framework will be thoroughly covered and explained in the subchapter, allowing for the development of a dependable and useful web application for the Smart Dam Monitoring System.

4.2.2.1 Controller Setup (dbcontroller, alarmcontroller & apicontroller)

Following the development of the Laravel Framework project file, the environment (.env) file must be configured in order to enable connections to the MySQL database and WebSocket server at the data processing layer. Controllers must then be created in order to retrieve data from the database. The Smart Dam Monitoring System relies heavily on three controllers in various ways. The web application's data visualisation charts require data to be retrieved, which is done by the 'dbcontroller'. The "alarmcontroller" accesses particular information about alarms to enable alarm features in the web application. Last but not least, the 'apicontroller' enables realtime data access from the database via API routing for the fuzzy logic AI server, facilitating data processing for model development. The Appendix contains the codes for these controllers, which may be used as a guide for how to implement and use them with the Laravel framework project.

4.2.2.2 Routes

The routing mechanism in the system should also be taken into account. Routes can be thought of as the request paths the system employs to reach files, including web pages, APIs, and controllers, all of which are essential to the system's operation. To establish meaningful links between the various parts of the Smart Dam Monitoring System, certain routes must be defined within the Laravel Framework. The system can enable efficient communication and interaction between various elements by defining the proper routes. It is easier to comprehend how data flows and connections are made among the Smart Dam Monitoring System's core components thanks to the additional information and visual representations of the routing structure provided in the following table.

Table 4.1: Route details and descriptions

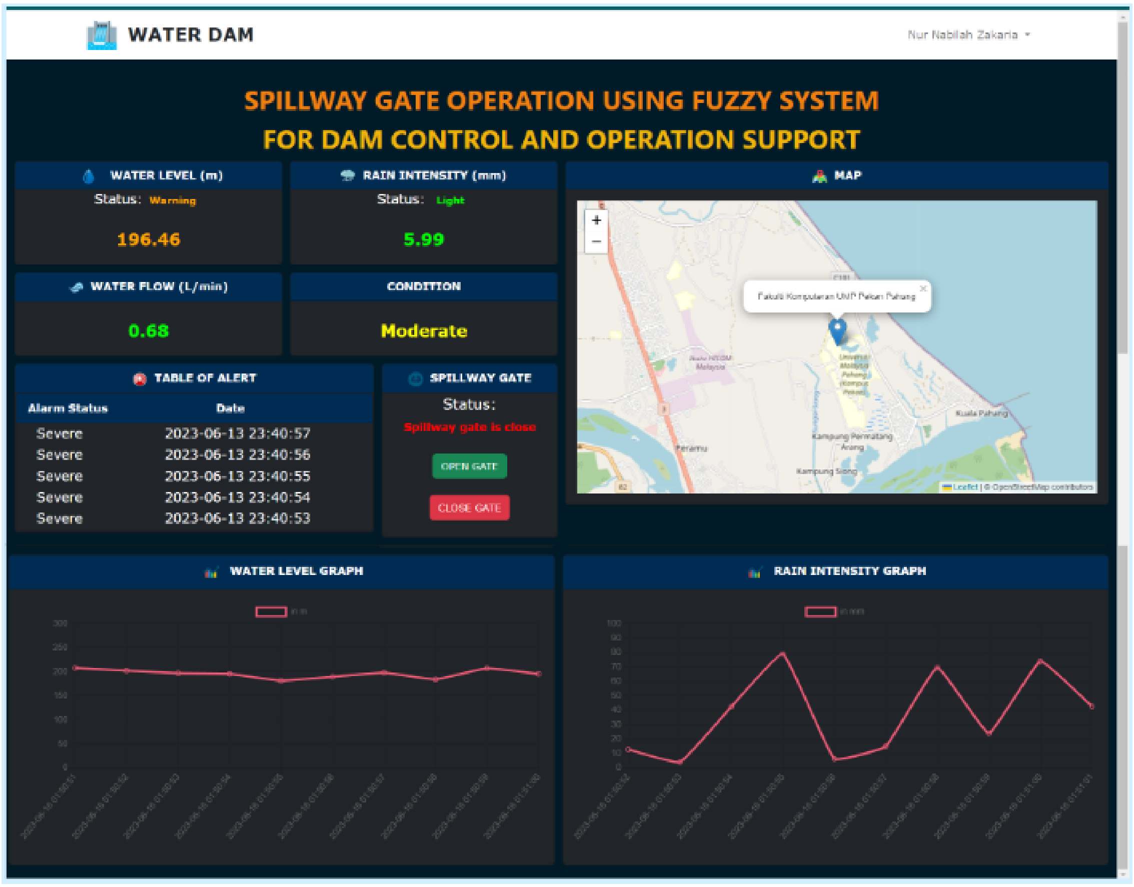
SYSTEM COMPONENT	ROUTE	METHOD	MIDDLEWARE GROUP	DESCRIPTION
dbcontroller.php	/route/rainfallroute	GET	web.php	Connect to dbcontroller.php
	/route/waterlevelroute			
	/route/waterflowroute			
alarmcontroller.php	/route/alarm			Connect to alarmcontroller.php
views	/			Redirect to LOGIN page
	/dashboard			Redirect to DASHBOARD page
WebSocket events	/opengate			Allow actuator control via WebSocket event trigger messaging
	/closegate			
apicontroller.php	/api/myWaterLevellink		api.php	Connection to apicontroller.php
	/api/myRainfalllink		web.php	

4.2.2.2 Events

Establishing event channels and triggers is crucial since WebSocket is an event-driven protocol, making it easier for web applications to connect to servers. The web application can send messages to the linked Raspberry Pi processor via these channels and triggers, and the attached Raspberry Pi responds with actuator responses as feedback. To accomplish this, it is necessary to construct an event class and a channel specification in order to establish a reliable connection between devices using the same channel name. A successful communication link between the Raspberry Pi microprocessor and the web application can be created by referencing the event class of the same name on the Raspberry Pi microprocessor. The two parts of the Smart Dam Monitoring System may communicate and exchange data without interruption thanks to this communication link.

4.2.2.2 Dashboard

Figure 4.1 below shows the dashboard of the system:



Alarm Status	Date
Severe	2023-06-13 23:40:57
Severe	2023-06-13 23:40:56
Severe	2023-06-13 23:40:55
Severe	2023-06-13 23:40:54
Severe	2023-06-13 23:40:53

Figure 4.1: Dashboard of the system

The dashboard of the Smart Dam Monitoring System incorporates several important features to provide users with comprehensive information and functionality. It includes registration, login, and logout functions to ensure secure access to the system. Figure 4.1 showcases the main page of the dashboard, where users can view all the essential data and values. At the top of the dashboard, the system's title prominently displays to identify the purpose of the platform. The water level (in meters) and rain intensity (in millimeters) are displayed on separate sections of the dashboard, indicating both the numerical value and the corresponding status of the data. The use of different colors helps users easily distinguish between the various data statuses.

For the water level, if the value falls within the range of 180-185m, the status is considered normal, represented by the color green. When the water level ranges from 186-190m, it triggers an alert status, represented by the color yellow. If the water level reaches the range of 191-200m, the status becomes a warning, denoted by the color orange. Lastly, if the water level exceeds 200m, it is categorized as dangerous, displayed in red.

For the rain intensity data, the dashboard distinguishes between different levels of intensity based on predefined thresholds. If the rain intensity falls within the range of 1-10mm, the status is categorized as light, indicated by the color green. This signifies a relatively low intensity of rainfall. When the rain intensity ranges from 11-30mm, the status changes to moderate, displayed in yellow. This indicates a moderate level of rainfall, suggesting a more substantial amount of precipitation. If the rain intensity reaches the range of 31-60mm, the status is labeled as heavy, represented by the color orange. This indicates a significant amount of rainfall, suggesting potentially adverse weather conditions. Lastly, if the rain intensity exceeds 60mm, it is classified as very heavy, denoted by the color red. This represents an intense and potentially severe amount of rainfall.

The water flow sensor section only displays the value of the flow rate in liters per minute (L/min). This information is primarily used to check if water is flowing out of the dam through the spillway gate when it is open. Therefore, the status of the water flow is not emphasized, as its main purpose is to provide a measurement rather than an alarm condition.

The fuzzy logic implementation in the system determines the condition of the dam based on the collected data. However, in the dashboard, only the status resulting from the fuzzy logic analysis is presented, without displaying the specific fuzzy rules or intermediate variables used in the calculation.
The alert table in the dashboard shows the history of alarm occurrences, indicating the date and time when the alarm was triggered. This allows users to track past alarm events and take appropriate actions if necessary. The spillway gate card includes buttons that provide an easy and quick way to check the current status of the gate. This feature simplifies the monitoring process and facilitates the decision-making process for opening or closing the spillway gate.

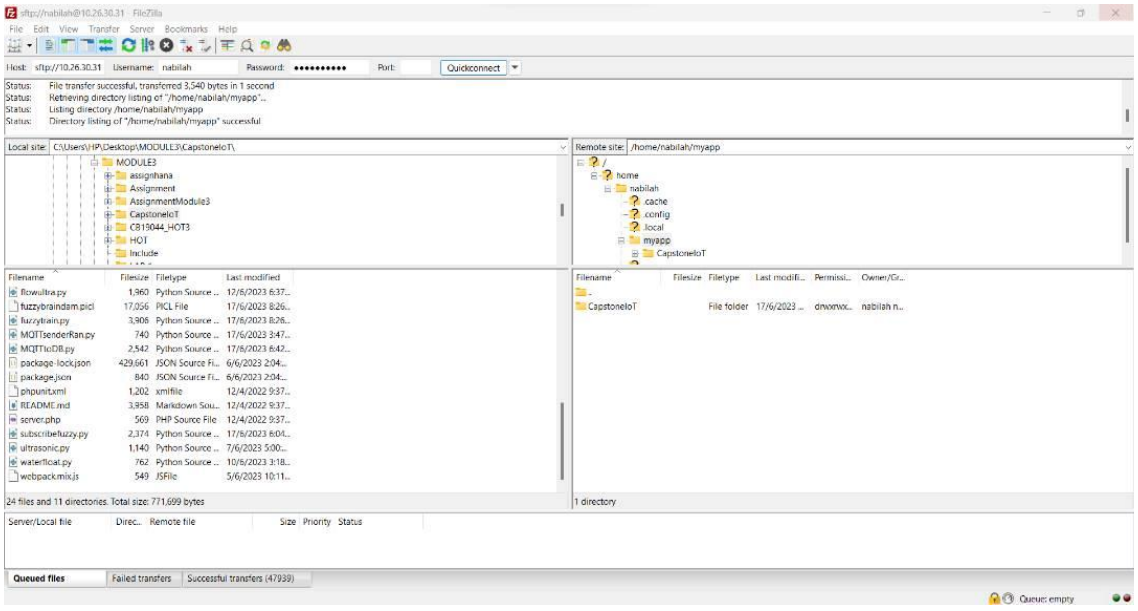
Additionally, a map is integrated into the dashboard to provide users with a visual representation of the dam's location or the placement of the sensors within the dam area. This feature helps users understand the geographical context of the monitoring system. Finally, the dashboard includes graphs displaying the latest ten data points for both water level and rain intensity. These real-time graphs provide a visual representation of the trends and fluctuations in these variables over time, allowing users to analyze and monitor changes effectively.

In conclusion, the Smart Dam Monitoring System's dashboard offers a comprehensive range of features and functionalities. It presents the necessary data and values in an organized manner, providing real-time information on water level, rain intensity, and water flow. The use of colors, status indicators, and interactive components enhances user experience and facilitates efficient decision-making. The integration of fuzzy logic analysis, alarm history, spillway gate status, map visualization, and real-time graphs further contributes to the system's effectiveness in monitoring and managing dam conditions.

4.2.3 Live Server Development

The next step is to deploy the system onto a live server after configuring and developing all of the environment configurations and web application pages in Laravel. With this setup, the system's communication links are all guaranteed to run quickly and to be able to support several users both inside and beyond the local network. The Laravel project folder must be uploaded to the server via the FTP network protocol to deploy the system; this is commonly done using programmes like FileZilla. Once the folder has been uploaded, certain commands must be run on the folder's properties to enable ongoing hosting of the web application's final development build at a certain address. In order to ensure that the required data is accessible, the MySQL database must also be moved to the server environment for device offloading.

Finally, in order to guarantee that all routings function as planned, additional routes must be defined. Through correct handling and routing of incoming requests, which are ensured in this step, the web application will be able to run and interact with the deployed system without any interruptions. These deployment procedures can be used to make the Smart Dam Monitoring System available to users, facilitating effective communication and offering the application to numerous users in diverse places.



Host: sftp://10.26.30.31 Username: nabilah Password: ●●●●●●●●●● Port:

```
Status:   File transfer successful, transferred 3,540 bytes in 1 second
Status:   Retrieving directory listing of "/home/nabilah/myapp"...
Status:   Listing directory /home/nabilah/myapp
Status:   Directory listing of "/home/nabilah/myapp" successful
```

Local site: C:\Users\HP\Desktop\MODULE3\CapstoneIoT\

```
MODULE3
  assignhana
  Assignment
  AssignmentModule3
  CapstoneIoT
  CB19044_HOT3
  HOT
  Include
```

Filename	Filesize	Filetype	Last modified
flowultra.py	1,960	Python Source ...	12/6/2023 6:37...
fuzzybraindam.picl	17,056	PICL File	17/6/2023 8:26...
fuzzytrain.py	3,906	Python Source ...	17/6/2023 8:26...
MQTTsenderRan.py	740	Python Source ...	17/6/2023 3:47...
MQTTtoDB.py	2,542	Python Source ...	17/6/2023 6:42...
package-lock.json	429,661	JSON Source Fi...	6/6/2023 2:04:...
package.json	840	JSON Source Fi...	6/6/2023 2:04:...
phpunit.xml	1,202	xmlfile	12/4/2022 9:37...
README.md	3,958	Markdown Sou...	12/4/2022 9:37...
server.php	569	PHP Source File	12/4/2022 9:37...
subscribefuzzy.py	2,374	Python Source ...	17/6/2023 6:04...
ultrasonic.py	1,140	Python Source ...	7/6/2023 5:00:...
waterfloat.py	762	Python Source ...	10/6/2023 3:18...
webpack.mix.js	549	JSFile	5/6/2023 10:11...

24 files and 11 directories. Total size: 771,699 bytes

Remote site: /home/nabilah/myapp

```
/
  home
    nabilah
      .cache
      .config
      .local
      myapp
        CapstoneIoT
```

Filename	Filesize	Filetype	Last modifi...	Permissi...	Owner/Gr...
..					
CapstoneIoT		File folder	17/6/2023 ...	drwxrwx...	nabilah n...

1 directory

Server/Local file	Direc...	Remote file	Size	Priority	Status

Queued files | Failed transfers | Successful transfers (47939)

Queue: empty

Figure 4.2: Transfer the project into the 'myapp' directory on the server side with FileZilla

When everything is finished, the working web application can now be hosted from a server environment with an access point address of 'http://10.26.30.31:8000/~nabilah' for local IP network hosting and 'http://103.53.35.133:8000/~nabilah' for external IP network hosting, respectively.

4.2.3 AI Technique Implementation

The smart dam monitoring system incorporates fuzzy logic to assess and determine the condition of the dam based on the inputs from various sensors. Fuzzy logic provides a flexible and intuitive approach to handle imprecise and uncertain data, allowing for more robust and accurate decision-making in complex systems.

In the context of the smart dam monitoring system, fuzzy logic is employed to analyze two crucial parameters: water level and rain intensity. These parameters play a significant role in assessing the overall condition and safety of the dam. By utilizing fuzzy logic, the system can effectively interpret and evaluate the inputs from the sensors, enabling a more comprehensive understanding of the current state of the dam.

The fuzzy logic system consists of linguistic variables, membership functions, and a set of predefined rules. The linguistic variables, such as water level and rain intensity, are defined with specific membership functions that represent their respective ranges and degrees of membership. These membership functions enable the system to assign linguistic terms to the inputs, such as "normal," "alert," "warning," and "danger," based on their measured values in Figure 4.3 below.

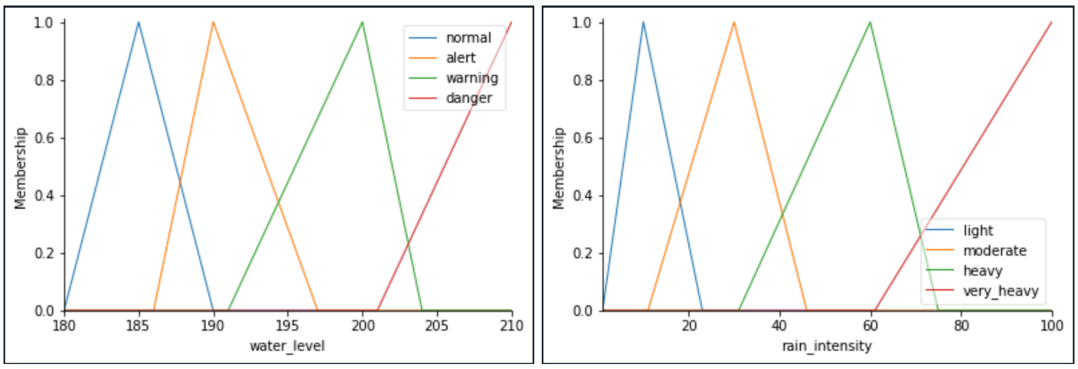


Figure 4.3: The graph of membership for water level and rain intensity in fuzzy

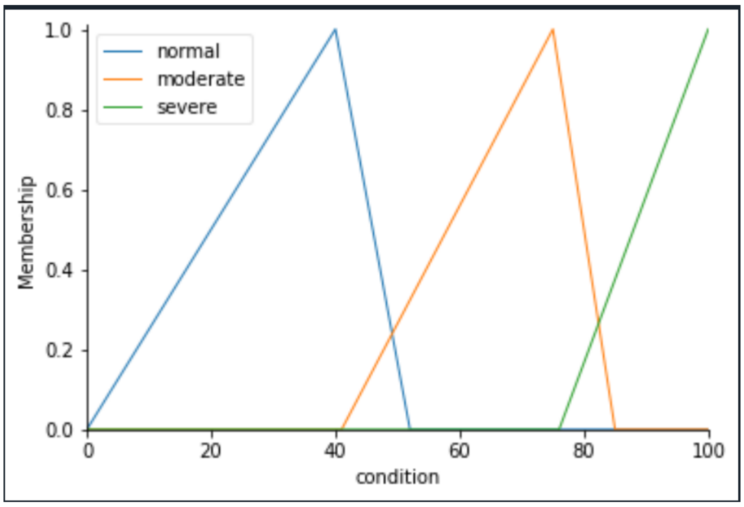


Figure 4.4: The output/condition of the fuzzy

The predefined rules in the fuzzy logic system establish the relationships between the input variables and the output variable, which is the condition of the dam. These rules capture the domain knowledge and expertise of dam monitoring experts, translating their insights into a computational framework. The rules define how the system should respond to different combinations of water level and rain intensity, ultimately determining the condition of the dam as "normal", "moderate", or "severe". The condition is shown in Figure 4.4.

By employing fuzzy logic, the smart dam monitoring system can handle the inherent uncertainties and vagueness in sensor data, providing a more comprehensive and accurate assessment of the dam's condition. This enables timely and informed decision-making, facilitating proactive measures to mitigate risks and ensure the safety and stability of the dam. The use of fuzzy logic enhances the reliability and effectiveness of the smart dam monitoring system, making it a valuable tool in dam management and risk prevention.

CHAPTER 5

CONCLUSION

5.1 Introduction

This chapter provides a comprehensive summary of the smart dam monitoring system, highlighting its achievements in meeting the project's objectives and scope. Throughout the project, the primary goal was to develop a reliable and efficient system for monitoring dams and assessing their conditions.

5.2 Conclusion of the Project

The smart dam monitoring system has first and foremost successfully integrated a variety of sensors and microcontrollers, including the WiFi Uno with ESP32 support and Raspberry Pi Model 3B+. These devices have made it possible to capture vital metrics including water level, rain intensity, and water flow in real-time data. The system has proven to be capable of gathering and processing precise and trustworthy data, giving a thorough insight into the dam's current state.

The dashboard is a crucial component of the smart dam monitoring system. It is clear and simple to use. A centralized platform for visualizing and analyzing the gathered data is offered by the dashboard. Users can quickly access important details including the water level, the severity of the rain, and the state of the dam. The system's usability is improved by the graphical depiction of data, which includes charts and color-coded status indicators. This facilitates effective decision-making.

A key component in determining the dam's condition has been the system's incorporation of fuzzy logic. The system can manage erroneous and questionable data by using fuzzy logic techniques, giving a more complex assessment of the dam's state. Based on the inputs from the water level and rain intensity sensors, the system has been able to identify the state as "normal," "moderate," or "severe" according to the defined membership functions and rule base. This fuzzy logic-based classification aids in spotting potential risks and taking suitable action by offering insightful information for dam management.

The smart dam monitoring system has also proven its ability to produce alerts and messages at the appropriate times. The technology instantly delivers alerts across many channels, including the dashboard and Telegram, when the water level rises above a certain threshold. This proactive alarm system enables stakeholders and dam managers to react quickly to urgent events, safeguarding the security of the dam and the environment around it.

In summary, the smart dam monitoring system has shown to be a useful instrument for tracking and evaluating the state of dams. The accuracy and effectiveness of dam management have increased thanks to the integration of numerous sensors, user-friendly interface, and use of fuzzy logic approaches. The system's capacity to deliver real-time data, categorize dam conditions, and produce appropriate alerts greatly aids proactive decision-making and risk reduction.

REFERENCES

1. V. Ramani Bai, & Mohamad Rom Tamjis. (2007). *Fuzzy Logic Model on Operation and Control of Hydro-Power Dams in Malaysia*. *4*(1), 31–40. https://doi.org/10.3970/icces.2007.004.031

2. Imran, M., Muzammal Zulfqar, Rasheed, H., Shahzadi Tayyaba, Muhammad WASEEM Ashraf, & Ahmad, Z. (2014). Fuzzy Logic Based Flow Controller of Dam Gates. *Journal of Engineering Research and Technology*, *1*(3). http://journal.iugaza.edu.ps/index.php/JERT/article/view/1619/1551

3. M. Abbas, M Saleem Khan, & Nasir Ali. (2011). *Fuzzy Logic Based Hydro -Electric power Dam Control System*. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=34b3302f9db86a55f9d27304a1bbc99208205dd6#page=16

4. Dervis Karaboga, Aytekin Bagis, & Tefaruk Haktanir. (2008). *Controlling spillway gates of dams by using fuzzy logic controller with optimum rule number*. *8*(1), 232–238. https://doi.org/10.1016/j.asoc.2007.01.004

5. Dhandre, N. M., Kamalasekaran, P. D., & Pandey, P. (2016). *Dam parameters monitoring system*. https://doi.org/10.1109/iicpe.2016.8079375

6. V. Sathya, Arun, K., Mahajan, H., & Singh, A. (2019). *Automate the Functioning of Dams Using IoT*. https://doi.org/10.1109/iccmc.2019.8819778

7. Mijovic, S., Shehu, E., & Buratti, C. (2016). *Comparing application layer protocols for the Internet of Things via experimentation*. https://doi.org/10.1109/rtsi.2016.7740559

8. Mijovic, S., Shehu, E., & Buratti, C. (2016). *Comparing application layer protocols for the Internet of Things via experimentation*. https://doi.org/10.1109/rtsi.2016.7740559

9. Dong, L., Shu, W., Sun, D., Li, X., & Zhang, L. (2017). *Pre-Alarm System Based on Real-Time Monitoring and Numerical Simulation Using Internet of Things and Cloud Computing for Tailings Dam in Mines*. *5*, 21080–21089. https://doi.org/10.1109/access.2017.2753379

10. Eridani, D., & Eko Didik Widianto. (2018). *Performance of Sensors Monitoring System using Raspberry Pi through MQTT Protocol*. https://doi.org/10.1109/isriti.2018.8864473

11. Oh Am Suk. (2018). *Design and Implementation of MQTT based on Arduino - ProQuest*. https://www.proquest.com/openview/c435f54b602d6db8d33d5badf2e87038/1?pq-origsite=gscholar&cbl=936334

APPENDIX

RASPBERRY PI (VNC Viewer)

DAMsensors.py

```
import paho.mqtt.client as mqtt
import RPi.GPIO as GPIO
import time, sys

# Set for MQTT broker
port = 1883
mqttBroker = "10.26.30.33"
client = mqtt.Client("Suhu")
client.username_pw_set("umpfk", "u4h%w1Tr12")
client.connect(mqttBroker,port)

GPIO.setmode(GPIO.BCM)

# Set the GPIO pins for the flow sensor
FLOW_SENSOR_GPIO = 13

# Set the GPIO pins for the ultrasonic sensor
GPIO_TRIGGER = 23
GPIO_ECHO = 24

GPIO.setup(FLOW_SENSOR_GPIO, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

global count
count = 0
```

```
def countPulse(channel):
    global count
    if start_counter == 1:
        count = count + 1

GPIO.add_event_detect(FLOW_SENSOR_GPIO, GPIO.FALLING,
callback=countPulse)

def distance_measurement():
    GPIO.output(GPIO_TRIGGER, True)
    # Set trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    start_time = time.time()
    stop_time = time.time()

    # Save start time
    while GPIO.input(GPIO_ECHO) == 0:
        start_time = time.time()

    # Save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        stop_time = time.time()

    # Calculate time difference
    time_elapsed = stop_time - start_time
    # Multiply with the sonic speed (34300 cm/s) and divide by 2
    distance = (time_elapsed * 34300) / 2

    return distance
```

```
def map_value(value, in_min, in_max, out_min, out_max):
    return (value - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def rain_intensity(distance):
    if distance < 1:
        distance = 1
    elif distance > 16:
        distance = 16

    intensity = map_value(distance, 16, 1, 1, 100)

    return int(intensity)

while True:
    try:
        start_counter = 1
        time.sleep(1)
        start_counter = 0
        flow = (count / 7.5) # Pulse frequency (Hz) = 7.5Q, Q is flow rate in L/min.

        distance = distance_measurement()
        intensity = rain_intensity(distance)

        print("The flow is: %.3f Liter/min" % flow)
        print(f"Distance: {distance} cm")
        print(f"Rain Intensity: {intensity} mm")

        client.publish("DAM/waterflow","{:3f}".format(flow))
        client.publish("DAM/rainfall",intensity)
        print("Just publish data to topic data")

        count = 0
        time.sleep(1)
```

```
   except KeyboardInterrupt:
      print('\nkeyboard interrupt!')
      GPIO.cleanup()
      sys.exit()
```

controlbutton.py

```
#import pusherclient
import sys
import time
import pysher
import pusher
import json

import RPi.GPIO as GPIO
from time import sleep
GPIO.setwarnings(False)
servoPIN = 18
GPIO.setmode(GPIO.BCM)
GPIO.setup(servoPIN, GPIO.OUT)

p = GPIO.PWM(servoPIN, 50)
p.start(12.5)

PUSHER_APP_ID='1'
PUSHER_APP_KEY='umpfkpusher'
PUSHER_APP_SECRET='u%M15z2h%3A'
PUSHER_APP_CLUSTER='mt1'
PUSHER_APP_HOST= '10.26.30.32'
PUSHER_APP_PORT = 6001
```

```
pusher_client = pusher.Pusher(host = PUSHER_APP_HOST,port =
PUSHER_APP_PORT,app_id=PUSHER_APP_ID, key=PUSHER_APP_KEY,
secret=PUSHER_APP_SECRET, ssl = False, cluster=PUSHER_APP_CLUSTER)

status = 0
# Add a logging handler so we can see the raw communication data
import logging
root = logging.getLogger()
root.setLevel(logging.INFO)
ch = logging.StreamHandler(sys.stdout)
root.addHandler(ch)

pusherpy = pysher.Pusher(PUSHER_APP_KEY)

def  my_func(*args, **kwargs):
   datacap = args[0]

   print(datacap)
   if datacap == '{"message":"on"}':
      print('Open')
      p.ChangeDutyCycle(7.5)
      status == 1
#        sleep(5)

   elif datacap == '{"message":"off"}':
      print('Close')
      p.start(2.5)
      p.ChangeDutyCycle(12.5)
      status == 0
#        sleep(1)
# p.stop()
# We can't subscribe until we've connected, so we use a callback handler
# to subscribe when able
def connect_handler(data):
    channel = pusherpy.subscribe('ToControlDam')
    #channel.bind('trade', callback)
    channel.bind('App\\Events\\ControlDeviceDam', my_func)



#pusher = pusherclient.Pusher(PUSHER_APP_ID, PUSHER_APP_KEY,
PUSHER_APP_SECRET, PUSHER_APP_CLUSTER)
pusherpy.connection.bind('pusher:connection_established', connect_handler)
pusherpy.connect()

while True:
    try:
        pusher_client.trigger(u'GateStatus', u'App\Events\ControlGateStatus',{u'status':
status})
        sleep(10)
    except:
        aaa=3;
# #        print('reconnect')
```

autogateDam.py

```
#import pusherclient
import sys
import time
import pysher
import pusher
import json

import RPi.GPIO as GPIO # Import Raspberry Pi GPIO library
```

```
from time import sleep # Import the sleep function from the time module
GPIO.setwarnings(False) # Ignore warning for now

servoPIN = 18
# Define the GPIO pins for the LEDs
led_pins = [17, 27] #y, b, y, b

# Set GPIO mode
GPIO.setmode(GPIO.BCM)
GPIO.setup(servoPIN, GPIO.OUT)
p = GPIO.PWM(servoPIN, 50)
p.start(10.5)

PUSHER_APP_ID='1'
PUSHER_APP_KEY='umpfkpusher'
PUSHER_APP_SECRET='u%M15z2h%3A'
PUSHER_APP_CLUSTER='mt1'
PUSHER_APP_HOST= '10.26.30.32'
PUSHER_APP_PORT = 6001

pusher_client = pusher.Pusher(host = PUSHER_APP_HOST,port = PUSHER_APP_PORT,app_id=PUSHER_APP_ID, key=PUSHER_APP_KEY, secret=PUSHER_APP_SECRET, ssl = False, cluster=PUSHER_APP_CLUSTER)



# Set the LED pins as output
GPIO.setup(led_pins, GPIO.OUT)

# Function to vibrate the motor continuously
import logging
root = logging.getLogger()
root.setLevel(logging.INFO)
ch = logging.StreamHandler(sys.stdout)
```

```
root.addHandler(ch)

pusherpy = pysher.Pusher(PUSHER_APP_KEY)
GPIO.output(17, GPIO.LOW)
GPIO.output(27, GPIO.LOW)
def led(*args, **kwargs):

    datacap = json.loads(args[0])
    print(datacap["Result"])
    if datacap["Result"] == "SEVERE":
        print('Gate open')
        p.ChangeDutyCycle(2.5)
        GPIO.output(27, GPIO.HIGH)
        GPIO.output(17, GPIO.LOW)
    elif datacap["Result"] == "MODERATE":
        print('Gate close')
        p.ChangeDutyCycle(10.5)
        GPIO.output(17, GPIO.HIGH)
        GPIO.output(27, GPIO.LOW)
    elif datacap["Result"] == "NORMAL":
        print('Gate close')
        p.ChangeDutyCycle(10.5)
        GPIO.output(27, GPIO.LOW)
        GPIO.output(17, GPIO.LOW)

def connect_handler(data):
    channel = pusherpy.subscribe('fuzzyControlDam')
    #channel.bind('trade', callback)
    channel.bind('App\\Events\\FuzzyEventDam', led)

# pusher = pusherclient.Pusher(PUSHER_APP_ID, PUSHER_APP_KEY,
PUSHER_APP_SECRET, PUSHER_APP_CLUSTER)
pusherpy.connection.bind('pusher:connection_established', connect_handler)
pusherpy.connect()

while True:

       aaa=10
           #print('reconnect')

def connect_handler(data):
   channel = pusherpy.subscribe('fuzzyControlDam')
   #channel.bind('trade', callback)
   channel.bind('App\\Events\\FuzzyEventDam', my_func)



pusher_client.trigger(u'GateStatus',      u'App\Events\ControlGateStatus',      {u'status':
GPIO.input(17)})
time.sleep(10)
           #print('reconnect')
```

Arduino IDE

dammonitorsystem.ino

```
#include <WiFi.h>
#include <PubSubClient.h>

//Wifi credentials
//const char* ssid = "Galaxy A22 5G8930"; //"Tihani"; // Enter your WiFi name
//const char* password =  "tihani6700";
//const char* ssid = "nurbz"; // Enter your WiFi name
//const char* password =  "nnabilahz"; // Enter WiFi password
//const char* ssid = "Chakepopo"; // Enter your WiFi name
//const char* password =  "mimiperi"; // Enter WiFi password
const char* ssid = "nadihahisa"; // Enter your WiFi name
const char* password =  "hamsterqiyut"; // Enter WiFi password

//MQTT broker details
const char* mqttServer = "103.53.35.135"; // data phone use external ip
const int mqttPort = 1883;
const char* mqttUser = "umpfk";
const char* mqttPassword = "u4h%w1Tr12";
const char* topic = "DAM/waterlevel"; // Enter the topic to publish the water level
data

const int sensorPin = 39;        // Analog input pin for the water level sensor
const float minWaterLevelCm = 21.0;  // Minimum water level in centimeters
const float maxWaterLevelCm = 24.0;  // Maximum water level in centimeters
const int sensorRange = 4095;         // Analog input range (0-4095)

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  Serial.begin(9600);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi..");
  }
  Serial.println("Connected to the WiFi network");

  client.setServer(mqttServer, mqttPort);

  while (!client.connected()) {
    Serial.println("Connecting to MQTT...");

    if (client.connect("ESP8266Client", mqttUser, mqttPassword )) {

      Serial.println("connected");

    } else {

      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);

    }
  }
}

void loop() {
  int sensorValue = analogRead(sensorPin);  // Read the analog input
  // Mapping is adjusted to consider the range from 210m to 240m, where 21cm
corresponds to 210m and 24cm corresponds to 240m.
```

```
  // Multiplication factor of 10 is used to convert centimeters to meters
  float waterLevelm = map(sensorValue, 0, sensorRange, minWaterLevelCm * 100,
maxWaterLevelCm * 100) / 10.0;  // Map analog value to the water level range in
meters
  //float waterLevelm = 34.5;
  Serial.println("Starting....");
  // Check if any reads failed and exit early (to try again)

  client.loop();

  if (!client.connected()) {
    reconnectMQTT();
  }

  Serial.println("Data collected from sensor");
  char ccounter[7];
  String Mystr = String(waterLevelm, 2);
  // String Mystr(waterLevelm);
  Mystr.toCharArray(ccounter,7);
  //client.publish(topic, ccounter); //Topic name
  client.publish(topic, ccounter);
  Serial.println("Published..");
  Serial.println(ccounter);

  // Add a small delay to allow for processing and to avoid flooding the MQTT broker
  delay(1000);
}

void reconnectMQTT() {
  while (!client.connected()) {
    Serial.println("Connecting to MQTT...");

    if (client.connect("ESP8266Client", mqttUser, mqttPassword)) {
```

```
    Serial.println("Connected to MQTT");
  } else {
    Serial.print("Failed to connect to MQTT with state ");
    Serial.print(client.state());
    delay(2000);
  }
}}
```

Visual Studio Code

MQTTtoDB.py

```
#receiver only -- only accept code from receiver
import json
import mysql.connector
import paho.mqtt.client as mqtt

MQTT_Broker = "10.26.30.33"
MQTT_Port = 1883
Keep_Alive_Interval = 45
MQTT_Topic = "DAM/#"

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="",
  database="dammonitoring"
)

mycursor = mydb.cursor()
```

```
# Function to save Humidity to DB Table
def goToWaterlevel(jsonData):

    Value = float(json.loads(jsonData))
    # print(Value)

    sql = "INSERT INTO waterlevel (id,Value) VALUES (%s,%s)"
    val = ('',Value)

    mycursor.execute(sql, val)
    mydb.commit()
    print("Water Level Data Stored")

def goToRainfall(jsonData):

    Value = float(json.loads(jsonData))
    # print(Value)

    sql = "INSERT INTO rainfall (id,Value) VALUES (%s,%s)"
    val = ('',Value)

    mycursor.execute(sql, val)
    mydb.commit()
    print("Rainfall Data Stored")

def goToWaterflow(jsonData):

    Value = float(json.loads(jsonData))
    # print(Value)

    sql = "INSERT INTO waterflow (id,Value) VALUES (%s,%s)"
    val = ('',Value)
    mycursor.execute(sql, val)
    mydb.commit()
    print("Water Flow Data Stored")

#===========================================================================
===
# Master Function to Select DB Funtion based on MQTT Topic

def sensor_Data_Handler(Topic, jsonData):
    if Topic ==  'DAM/waterlevel':
        goToWaterlevel(jsonData)
    elif Topic == 'DAM/rainfall':
        goToRainfall(jsonData)
    elif Topic == 'DAM/waterflow':
        goToWaterflow(jsonData)

#===========================================================================
===

#Subscribe to all Sensors at Base Topic
def on_connect(mosq, obj,flag, rc):
    mqttc.subscribe(MQTT_Topic, 0)

#Save Data into DB Table
def on_message(mosq, obj, msg):
    # print ("MQTT Data Received...")
    # print ("MQTT Topic: " + msg.topic)
    # print ("Data: " + str(msg.payload))
    sensor_Data_Handler(msg.topic, msg.payload) # get msg.payload from broker

def on_subscribe(mosq, obj, mid, granted_qos):
    pass
```

```

mqttc = mqtt.Client()
# Assign event callbacks
mqttc.username_pw_set("umpfk", "u4h%w1Tr12")
#mqttc.username_pw_set("umpfk", "umpiot123")

mqttc.on_message = on_message
mqttc.on_connect = on_connect
mqttc.on_subscribe = on_subscribe

# Connect

mqttc.connect(MQTT_Broker, int(MQTT_Port), int(Keep_Alive_Interval))
# Continue the network loop
mqttc.loop_forever()

#===========================================================
===
```

env file

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:tCI0/tlfru/pz4hid4R0kUL08pBBNEwGc+Qp2H2V0hs=
APP_DEBUG=true
APP_URL=http://localhost

LOG_CHANNEL=stack
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug

DB_CONNECTION=mysql
```

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=dammonitoring
DB_USERNAME=root
DB_PASSWORD=

BROADCAST_DRIVER=pusher
CACHE_DRIVER=file
FILESYSTEM_DRIVER=local
QUEUE_CONNECTION=sync
SESSION_DRIVER=file
SESSION_LIFETIME=120

MEMCACHED_HOST=127.0.0.1

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_MAILER=smtp
MAIL_HOST=mailhog
MAIL_PORT=1025
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS=null
MAIL_FROM_NAME="${APP_NAME}"

AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_DEFAULT_REGION=us-east-1
AWS_BUCKET=
AWS_USE_PATH_STYLE_ENDPOINT=false
```

```
PUSHER_APP_ID=1
PUSHER_APP_KEY=umpfkpusher
PUSHER_APP_SECRET=u%M15z2h%3A
PUSHER_APP_CLUSTER=mt1

MIX_PUSHER_APP_KEY="${PUSHER_APP_KEY}"
MIX_PUSHER_APP_CLUSTER="${PUSHER_APP_CLUSTER}"
```

bootstrap.js

```
window._ = require('lodash');

try {
   require('bootstrap');
} catch (e) {}

/**
 * We'll load the axios HTTP library which allows us to easily issue requests
 * to our Laravel back-end. This library automatically handles sending the
 * CSRF token as a header based on the value of the "XSRF" token cookie.
 */

window.axios = require('axios');

window.axios.defaults.headers.common['X-Requested-With'] = 'XMLHttpRequest';

/**
 * Echo exposes an expressive API for subscribing to channels and listening
 * for events that are broadcast by Laravel. Echo and event broadcasting
 * allows your team to easily build robust real-time web applications.
 */
```

```

import Echo from 'laravel-echo';

window.Pusher = require('pusher-js');

window.Echo = new Echo({
    broadcaster: 'pusher',
    key: process.env.MIX_PUSHER_APP_KEY,
    cluster: process.env.MIX_PUSHER_APP_CLUSTER,
    wsHost: '10.26.30.32', //window.location.hostname
    wsPort: 6001,
    forceTLS: false
});
```

broadcasting.php

```
<?php

return [

    /*
    |--------------------------------------------------------------------------
    | Default Broadcaster
    |--------------------------------------------------------------------------
    |
    | This option controls the default broadcaster that will be used by the
    | framework when an event needs to be broadcast. You may set this to
    | any of the connections defined in the "connections" array below.
    |
    | Supported: "pusher", "ably", "redis", "log", "null"
    |
    */
```

```
'default' => env('BROADCAST_DRIVER', 'null'),

/*
|--------------------------------------------------------------------------
| Broadcast Connections
|--------------------------------------------------------------------------
|
| Here you may define all of the broadcast connections that will be used
| to broadcast events to other systems or over websockets. Samples of
| each available type of connection are provided inside this array.
|
*/

'connections' => [

    'pusher' => [
        'driver' => 'pusher',
        'key' => env('PUSHER_APP_KEY'),
        'secret' => env('PUSHER_APP_SECRET'),
        'app_id' => env('PUSHER_APP_ID'),
        'options' => [
            'cluster' => env('PUSHER_APP_CLUSTER'),
            'encrypted' => true,
            'host' => '10.26.30.32',
            'port' => 6001,
            'scheme' => 'http'
        ],
    ],

    'ably' => [
        'driver' => 'ably',
        'key' => env('ABLY_KEY'),
        ],

        'redis' => [
            'driver' => 'redis',
            'connection' => 'default',
        ],

        'log' => [
            'driver' => 'log',
        ],

        'null' => [
            'driver' => 'null',
        ],

    ],

];
```

ControlDeviceDam.php

```
<?php

namespace App\Events;

use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Broadcasting\PresenceChannel;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;
```

```

class ControlDeviceDam implements ShouldBroadcast

{
    use Dispatchable, InteractsWithSockets, SerializesModels;
    public $message;
    /**
     * Create a new event instance.
     *
     * @return void
     */
    public function __construct($message)
    {
        $this->message = $message;
    }

    /**
     * Get the channels the event should broadcast on.
     *
     * @return \Illuminate\Broadcasting\Channel|array
     */
    public function broadcastOn()
    {
        return new Channel('ToControlDam');
    }}
```

apicontroller.php

```
<?php

namespace App\Http\Controllers;

```

```
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use App\Models\waterlevel;
use App\Models\rainfall;

class apicontroller extends Controller
{
   public function __construct()
   {
      $this->waterlevel = new waterlevel();
      $this->rainfall = new rainfall();
   }

   public function APIlistWaterlevel()
   {
      $blocks = DB::table('waterlevel')
      ->select('Value')
      ->latest('DateTime')
      ->limit(1)
      ->pluck('Value');
      return (compact('blocks')); # block is for value / block2 is for date create
   }
   public function APIlistRainfall()
   {
      $blocks = DB::table('rainfall')
      ->select('Value')
      ->latest('DateTime')
      ->limit(1)
      ->pluck('Value');
      return (compact('blocks')); # block is for value / block2 is for date create
   }
}
```

dbcontroller.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use App\Models\waterlevel;
use App\Models\rainfall;
use App\Models\waterflow;

class dbcontroller extends Controller
{
   public function __construct()
   {
      $this->waterlevel = new waterlevel();
      $this->raindrop = new rainfall();
      $this->waterflow = new waterflow();

   }

   public function getWaterlevel()
   {
      $blocks = DB::table('waterlevel')
      ->select('Value')
      ->latest('DateTime')
      ->limit(10)
      ->pluck('Value');

      $blocks2 = DB::table('waterlevel')
      ->select('Value','DateTime')
      ->latest('DateTime')
      ->limit(10)
      ->pluck('DateTime');
      return (compact('blocks','blocks2')); # block is for value / block2 is for date
create
   }

   public function getRainfall()
   {
      $blocks = DB::table('rainfall')
      ->select('Value')
      ->latest('DateTime')
      ->limit(10)
      ->pluck('Value');

      $blocks2 = DB::table('rainfall')
      ->select('Value','DateTime')
```

```
        ->latest('DateTime')
        ->limit(10)
        ->pluck('DateTime');
        return (compact('blocks','blocks2')); # block is for value / block2 is for date
create
    }

    public function getWaterflow()
    {
        $blocks = DB::table('waterflow')
        ->select('Value')
        ->latest('DateTime')
        ->limit(10)
        ->pluck('Value');

        $blocks2 = DB::table('waterflow')
        ->select('Value','DateTime')
        ->latest('DateTime')
        ->limit(10)
        ->pluck('DateTime');
        return (compact('blocks','blocks2')); # block is for value / block2 is for date
create
    }
}
```

alarmcontroller.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use App\Models\alarm;

class alarmcontroller extends Controller
{
    public function __construct()
    {
        $this->alarm = new alarm();
    }

    public function getAlarm()
    {
        $blocks = DB::table('alarm')
        ->select('DeviceID')
        ->latest('DateTime')
        ->limit(5)
        ->pluck('DeviceID');

        $blocks2 = DB::table('alarm')
        ->select('DeviceID','DateTime')
        ->latest('DateTime')
        ->limit(5)
        ->pluck('DateTime');
        return (compact('blocks','blocks2')); # block is for value / block2 is for date
create
    }
}
```

alarm.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class alarm extends Model
{
    use HasFactory;
}
```

rainfall.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class rainfall extends Model
{
    use HasFactory;
}
```

waterflow.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class waterflow extends Model
{
   use HasFactory;
}
```

waterlevel.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class waterlevel extends Model
{
   use HasFactory;
}
```

app.blade.php

```
<!doctype html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
<head>
   <meta charset="utf-8">
   <meta name="viewport" content="width=device-width, initial-scale=1">

   <!-- CSRF Token -->
   <meta name="csrf-token" content="{{ csrf_token() }}">

   <title>{{ config('app.name', 'Laravel') }}</title>
                                                                        <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1
oBoqyl2QvZ6jIW3" crossorigin="anonymous">
                                                                      <script
```

```
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0
LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>
   <!-- Scripts -->
   <script src="{{ asset('js/app.js') }}"></script>

   <!-- Fonts -->
   <link rel="dns-prefetch" href="//fonts.gstatic.com">
   <link href="https://fonts.googleapis.com/css?family=Nunito" rel="stylesheet">

   <!-- Styles -->
   <link href="{{ asset('css/app.css') }}" rel="stylesheet">
   <link href="{{ asset('css/sidebars.css') }}" rel="stylesheet">
   <script src="js/sidebars.js"></script>



   <!-- chart -->
   <script src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0"></script>
   <script src="https://cdn.jsdelivr.net/npm/@jaames/iro@5"></script>
                                                                                                                   <script
src="https://cdn.jsdelivr.net/npm/chartjs-gauge@0.3.0/dist/chartjs-gauge.min.js"></sc
ript>
                                                                                                                   <script
src="https://cdnjs.cloudflare.com/ajax/libs/chartjs-chart-box-and-violin-plot/2.4.0/Ch
art.BoxPlot.min.js"></script>
   <!-- <script src="chart.js/chartjs-gauge.js"></script> -->


   <!-- geocode pack -->
   <script src="https://unpkg.com/leaflet/dist/leaflet-src.js"></script>
   <script src="https://unpkg.com/esri-leaflet"></script>
   <script src="https://unpkg.com/esri-leaflet-geocoder"></script>

   <!-- routing -->
  <link rel="stylesheet" href="https://unpkg.com/leaflet@1.2.0/dist/leaflet.css" />
                                                   <link                    rel="stylesheet"
href="https://unpkg.com/leaflet-routing-machine@latest/dist/leaflet-routing-machine.
css" />
  <!-- <script src="https://unpkg.com/leaflet@1.2.0/dist/leaflet.js"></script> -->
                                                                                                                   <script
src="https://unpkg.com/leaflet-routing-machine@latest/dist/leaflet-routing-machine.j
s"></script>


   <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
   <link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

</head>
```

```
<body>

   <div id="app">
      <nav class="navbar navbar-expand-md navbar-light bg-white shadow-sm">
         <div class="container">
            <a class="navbar-brand" href="#">
                              <img src="DAM.png" alt="Logo" width="40" height="40"
class="d-inline-block align-text-top">
                   <a class="nav-link active" aria-current="page" href="/dashboardpage"
style="font-size: 25px;"><b>WATER DAM</b></a>
            </a>

               <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent"
aria-expanded="false" aria-label="{{ __('Toggle navigation') }}">
               <span class="navbar-toggler-icon"></span>
            </button>

            <div class="collapse navbar-collapse" id="navbarSupportedContent">
               <!-- Left Side Of Navbar -->
               <ul class="navbar-nav me-auto">

               </ul>

               <!-- Right Side Of Navbar -->
               <ul class="navbar-nav ms-auto">
                  <!-- Authentication Links -->
                  @guest
                     @if (Route::has('login'))
                        <li class="nav-item">
                             <a class="nav-link" href="{{ route('login') }}">{{ __('Login')
}}</a>
                        </li>
                     @endif

                     @if (Route::has('register'))
                        <li class="nav-item">
                                        <a class="nav-link" href="{{ route('register') }}">{{
__('Register') }}</a>
                        </li>
                     @endif
                  @else
                     <li class="nav-item dropdown">
                              <a id="navbarDropdown" class="nav-link dropdown-toggle"
href="#"    role="button"    data-bs-toggle="dropdown"    aria-haspopup="true"
aria-expanded="false" v-pre>
                           {{ Auth::user()->name }}
                        </a>
```

```
                                        <div class="dropdown-menu dropdown-menu-end"
aria-labelledby="navbarDropdown">
                                <a class="dropdown-item" href="{{ route('logout') }}"
                                  onclick="event.preventDefault();

document.getElementById('logout-form').submit();">
                                   {{ __('Logout') }}
                                </a>

                                    <form id="logout-form" action="{{ route('logout') }}"
method="POST" class="d-none">
                                   @csrf
                                </form>
                             </div>
                          </li>
                       @endguest
                    </ul>
                 </div>
              </div>
           </nav>

           <main class="py-4">
              @yield('content')
           </main>
        </div>
</body>
</html>
```

dashboard.blade.php

```
@extends('layouts.app')

@section('content')
<body style="background-color: #011a27;">
   <style>
      body {
      background-color: #011a27;
      }

      h1 {
      color: #f0810f; /* Replace with your desired color value */
      font-weight: bold;
      }

      .icon {
         width: 20px; /* Adjust the width of the icon */
```

```
    height: 20px; /* Adjust the height of the icon */
    margin-right: 10px; /* Adjust the spacing between the icon and text */
    vertical-align: middle; /* Align the icon vertically with the text */
}

.icon2 {
    width: 40px; /* Adjust the width of the icon */
    height: 20px; /* Adjust the height of the icon */
    margin-right: 10px; /* Adjust the spacing between the icon and text */
    vertical-align: middle; /* Align the icon vertically with the text */
}


.icongraph {
    width: 30px; /* Adjust the width of the icon */
    height: 30px; /* Adjust the height of the icon */
    margin-right: 10px; /* Adjust the spacing between the icon and text */
    vertical-align: middle; /* Align the icon vertically with the text */
}

.grid-container {
display: grid;
grid-gap: 10px;
grid-template-columns: repeat(12, 1fr);
}

/* .grid-container2 {
display: grid;
grid-gap: 10px;
grid-template-columns: repeat(2, 1fr);
} */

.grid-item {
width: 100%;
height: 100%;
}

.card {
background-color: #002c54;
color: black;
}

.card-header {
background-color: #002c54;
color: white;
font-weight: bold;
text-align: center;
}
```

```
.card-body {
font-size: 30px;
text-align: center;
}

.card-body1 {
font-size: 20px;
text-align: center;
}

.card-body2 {
font-size: 30px;
text-align: center;
}

.WATERLEVEL {
grid-row-start: 1;
grid-row-end: 2;
grid-column-start: 1;
grid-column-end: 4;
}

.RAINFALL {
grid-row-start: 1;
grid-row-end: 2;
grid-column-start: 4;
grid-column-end: 7;
}

.WATERFLOW {
grid-row-start: 2;
grid-row-end: 3;
grid-column-start: 1;
grid-column-end: 4;
}

.FUZZYRESULT {
grid-row-start: 2;
grid-row-end: 3;
grid-column-start: 4;
grid-column-end: 7;
}

.MAP {
grid-row-start: 1;
grid-row-end: 7;
grid-column-start: 7;
grid-column-end: 13;
}
```

```
/* .FUZZYGAUGE {
grid-row-start: 1;
grid-row-end: 2;
grid-column-start: 1;
grid-column-end: 2;
}

.FUZZYVALUE {
grid-row-start: 1;
grid-row-end: 2;
grid-column-start: 2;
grid-column-end: 3;
} */

.ALARM{
grid-row-start: 3;
grid-row-end: 7;
grid-column-start: 1;
grid-column-end: 5;
text-align: center;
}

.GATE {
grid-row-start: 3;
grid-row-end: 7;
grid-column-start: 5;
grid-column-end: 7;
}

.WATERLEVELgraph {
grid-row-start: 7;
grid-row-end: 10;
grid-column-start: 1;
grid-column-end: 7;
}

.RAINFALLgraph {
grid-row-start: 7;
grid-row-end: 10;
grid-column-start: 7;
grid-column-end: 13;
}

/* .WATERFLOWgraph {
grid-row-start: 7;
grid-row-end: 10;
grid-column-start: 9;
grid-column-end: 13;
```

```
      } */

      .chart-axis-values {
         color: #f5f5f7; /* Replace with your desired light color value */
      }

   </style>
   <div class="container-fluid">
      <div class="dashboard">
               <h1 class="text-center" style="color: ##FFDE2E;">SPILLWAY GATE
OPERATION USING FUZZY SYSTEM</h1>
             <h1 class="text-center" style="color: #efb509;">FOR DAM CONTROL
AND OPERATION SUPPORT</h1>
         <!-- <br> -->
         <div class="grid-container"><!-- OPEN CONTAINER -->
            <!-- Card WATERLEVEL -->
            <div class="grid-item WATERLEVEL">
               <div class="card bg-dark text-white">
                  <!-- WATERLEVEL value -->
                  <div class="card-header">
                        <img src="water.png" class="icon2">WATER LEVEL (m)<span
class="badge badge-info float-right"></span>
                  </div>
                        <div class="card-body1" style= "font-size: 18px"> Status:<span
class="badge badge-info float-right" id="wtrlvlstatus"></span></div>
                     <div class="card-body"><span class="badge badge-info float-right"
id="wtrlvlvalue"></span></div>
               </div>
            </div>

            <!-- Card RAINFALL -->
            <div class="grid-item RAINFALL">
               <div class="card bg-dark text-white">
                  <!-- RAINFALL value -->
                  <div class="card-header">
                      <img src="rain.png" class="icon">RAIN INTENSITY (mm)<span
class="badge badge-info float-right"></span>
                  </div>
                       <div class="card-body1" style= "font-size: 18px"> Status: <span
class="badge badge-info float-right" id="rainstatus"></span></div>
                     <div class="card-body"><span class="badge badge-info float-right"
id="rainvalue"></span></div>
               </div>
            </div>

            <!-- Card WATERFLOW -->
            <div class="grid-item WATERFLOW">
               <div class="card bg-dark text-white">
                  <!-- WATERFLOW value -->
```

```
                    <div class="card-header">
                         <img src="flow.png" class="icon">WATER FLOW (L/min)<span
class="badge badge-info float-right"></span>
                    </div>
                      <div class="card-body"><span class="badge badge-info float-right"
id="wtrflowvalue"></span></div>
                </div>
            </div>

            <!-- Card WATERFLOW -->
            <div class="grid-item FUZZYRESULT">
                <div class="card bg-dark text-white">
                    <!-- WATERFLOW value -->
                    <div class="card-header">
                        CONDITION<span class="badge badge-info float-right"></span>
                    </div>
                      <div class="card-body"><span class="badge badge-info float-right"
id="fuzzyresult"></span></div>
                </div>
            </div>

            <!-- Card MAP -->
            <div class="grid-item MAP">
                <div class="card bg-dark text-white">
                    <!-- MAP -->
                    <div class="card-header">
                              <img src="map.png" class="icon">MAP<span class="badge
badge-info float-right"></span>
                    </div>
                    <div class="card-body">
                        <div id="map" style="height: 400px;"></div>
                    </div>
                </div>
            </div>

            <!-- Card FUZZYRESULT -->
            <!-- <div class="grid-item FUZZYRESULT">
                <div class="card bg-dark text-white">
                          <div class="card-header">FUZZY RESULT<span class="badge
badge-info float-right"></span></div>
                    <div class="grid-containerfuzzy">
                        <div class="grid-item FUZZYGAUGE"> -->
                            <!-- FUZZYRESULT graph -->
                                    <!-- <div class="card-body d-flex justify-content-center
align-items-center">
                                    <canvas id="chartfuzzy" class="chartjs" style="position:
relative; height:40vh; width:40vw"></canvas>
                            </div>
                        </div>
                        <div class="grid-item FUZZYVALUE"> -->
                            <!-- FUZZYRESULT graph -->
                          <!-- <div class="card-header">CONDITION<span class="badge
badge-info float-right"></span></div>
                                <div class="card-body2"><span class="badge badge-info
float-right" id="fuzzyvalue"></span></div>
                        </div>
                    </div>
                </div>
            </div>-->

            <!-- ALARM -->
            <div class="grid-item ALARM">
                <div class="card bg-dark text-white">
                    <div class="card-header">
                            <img src="alarm.png" class="icon">TABLE OF ALERT<span
class="badge badge-info float-right" id="myAlert"></span>
                    </div>
                            <div class="card bg-dark text-white"class="card-body1 d-flex
justify-content-center align-items-center" >
                        <!-- <br> -->

                        <!-- Alert Status -->
                            <div class="card bg-dark text-white" style="width: 100%; height:
auto;">
                            <div class="custom-header card-header" style= "height: 40px;">
                                <div class="d-flex justify-content-between">
                                <div style="flex: 1;">
                                    <table id="myTablehead">
                                        <tr>
                                            <td>Alarm Status</td>
                                            <td style="width: 75%">Date</td>
                                        </tr>
                                    </table>
                                </div>
                                </div>
                            </div>

                                <div class="card-body1" style="overflow: auto; max-height:
150px;">
                                <div class="d-flex justify-content-between">
                                <div style="flex: 1;">
                                        <table id="myTable" style="width: 100%; font-size:
18px; color: white;">
                                        <tr>
                                            <td></td>
                                            <td></td>
                                        </tr>
                                </table>
                              </div>
                            </div>
                          </div>

                        </div>
                      </div>
                    </div>
                  </div>

                  <!-- Card GATE -->
                  <div class="grid-item GATE">
                    <div class="card bg-dark text-white">
                      <div class="card-header">
                              <img src="gate.png" class="icon">SPILLWAY GATE<span
class="badge badge-info float-right"></span>
                      </div>
                             <div class="card bg-dark text-white"class="card-body1 d-flex
justify-content-center align-items-center" >

                        <!-- ACTIONS -->
                           <div class="card-body1">Status:<span class="badge badge-info
float-right" id="ActuatorStatus"></span></div>
                        <br>
                                  <form class="card-body1 d-flex justify-content-center
align-items-center" action="/opengate" method="post">
                                 <input type="submit" value="OPEN GATE" class="btn
btn-success">
                          {{csrf_field()}}
                        </form>
                        <br>
                                  <form class="card-body1 d-flex justify-content-center
align-items-center" action="/closegate" method="post">
                               <input type="submit" value="CLOSE GATE" class="btn
btn-danger">
                          {{csrf_field()}}
                        </form><br>
                      </div>
                    </div>
                  </div>

                  <!-- Card WATERLEVEL -->
                  <div class="grid-item WATERLEVELgraph">
                    <div class="card bg-dark text-white">
                      <!-- WATERLEVEL value-->
                      <div class="card-header">
                              <img src="graph.png" class="icongraph">WATER LEVEL
GRAPH<span class="badge badge-info float-right"></span>
                      </div>
```

```
                                        <div  class="card-body  d-flex  justify-content-center
align-items-center">
                              <canvas id="chartwtrlvl" class="chartjs" style="position: relative;
height:40vh; width:40vw"></canvas>
                    </div>
                </div>
            </div>

            <!-- Card RAINFALL -->
            <div class="grid-item RAINFALLgraph">
                <div class="card bg-dark text-white">
                    <!-- RAINFALL value-->
                    <div class="card-header">
                                <img src="graph.png" class="icongraph">RAIN INTENSITY
GRAPH<span class="badge badge-info float-right"></span>
                    </div>
                                        <div  class="card-body  d-flex  justify-content-center
align-items-center">
                                <canvas id="chartrain" class="chartjs" style="position: relative;
height:40vh; width:40vw"></canvas>
                    </div>
                </div>
            </div>

            <!-- Card WATERFLOW -->
            <!-- <div class="grid-item WATERFLOWgraph">
                <div class="card bg-dark text-white"> -->
                    <!-- WATERFLOW value-->
                                <!--  <div  class="card-header">WATER  FLOW  GRAPH<span
class="badge badge-info float-right"></span></div>
                                        <div  class="card-body  d-flex  justify-content-center
align-items-center">
                            <canvas id="chartwtrflow" class="chartjs" style="position: relative;
height:40vh; width:40vw"></canvas>
                    </div>
                </div>
            </div> -->
        </div><!-- CLOSE CONTAINER -->

    </div>
  </div>
</body>
<script>

  setInterval(ajaxCall, 5000);

  function ajaxCall() {
    // data water level
    $.getJSON('/route/waterlevelroute', function(blocksall) {
```

```
var datas = blocksall.blocks.map(Number);
datas = datas.reverse();
console.log(datas)

var datasx = blocksall.blocks2.map(String);
datasx = datasx.reverse();
console.log(datasx)

var waterLevelValue = datas[datas.length - 1].toFixed(2);
document.getElementById("wtrlvlvalue").innerHTML = waterLevelValue;
setTextColor("wtrlvlvalue", waterLevelValue);

function setTextColor(elementId, value) {
   var color;
   if (value <= 185) {
      color = "#00FF00"; // Green
   } else if (value >= 186 && value < 190) {
      color = "#FFFF00"; // Yellow
   } else if (value >= 191 && value < 200) {
      color = "#FFA500"; // Orange
   } else {
      color = "#FF0000"; // Red
   }
   document.getElementById(elementId).style.color = color;
}

var wtrlvlstatus;
var lastwtrlvl = datas[datas.length - 1];

if (lastwtrlvl <= 185) {
   wtrlvlstatus = "Normal";
} else if (lastwtrlvl >= 186 && lastwtrlvl < 190) {
   wtrlvlstatus = "Alert";
} else if (lastwtrlvl >= 191 && lastwtrlvl < 200) {
   wtrlvlstatus = "Warning";
} else {
   wtrlvlstatus = "Danger";
}

var wtrlvlstatusElement = document.getElementById("wtrlvlstatus");
wtrlvlstatusElement.innerHTML = wtrlvlstatus;

if (wtrlvlstatus == "Normal") {
   wtrlvlstatusElement.style.color = "#00FF00"; // Green
} else if (wtrlvlstatus == "Alert") {
   wtrlvlstatusElement.style.color = "#FFFF00"; // Yellow
} else if (wtrlvlstatus == "Warning") {
   wtrlvlstatusElement.style.color = "#FFA500"; // Orange
} else if (wtrlvlstatus == "Danger"){
        wtrlvlstatusElement.style.color = "#FF0000"; // Red
    }


    var chart = new Chart(document.getElementById('chartwtrlvl'), {
        type: 'line',
        data: {
            labels : datasx,
            datasets: [
                {
                label: 'in m',
                data: datas,
                fill: false,
                borderColor: '#f55d7a',
                tension: 0.1
                }]
            },

        'options':{
            animation: {
                duration: 100,
                easing: 'easeOutBounce'
            },
            scales:{
                yAxes: [{
                    display:true,
                    stacked:true,
                    ticks:{
                        min:0, //minimum value
                        max:300 //maximum value
                    }
                }]
            }
        },

    });
});

// data rainfall
$.getJSON('/route/rainfallroute', function(blocksall) {
    var datas = blocksall.blocks.map(Number);
    datas = datas.reverse();
    console.log(datas)

    var datasx = blocksall.blocks2.map(String);
    datasx = datasx.reverse();
    console.log(datasx)

    // Classification based on rain// Red
```

```
var rainfallValue = datas[datas.length - 1];
document.getElementById("rainvalue").innerHTML = rainfallValue;
setTextColor("rainvalue", rainfallValue);

function setTextColor(elementId, value) {
   var color;
   if (value <= 10) {
      color = "#00FF00"; // Green
   } else if (value >= 11 && value < 30) {
      color = "#FFFF00"; // Yellow
   } else if (value >= 31 && value < 60) {
      color = "#FFA500"; // Orange
   } else {
      color = "#FF0000"; // Red
   }
   document.getElementById(elementId).style.color = color;
}

var rainstatus;
var lastrain = datas[datas.length - 1];

if (lastrain <= 10) {
   rainstatus = "Light";
} else if (lastrain >= 11 && lastrain < 30) {
   rainstatus = "Moderate";
} else if (lastrain >= 31 && lastrain < 60) {
   rainstatus = "Heavy";
} else {
   rainstatus = "Very Heavy";
}

var rainstatusElement = document.getElementById("rainstatus");
rainstatusElement.innerHTML = rainstatus;

if (rainstatus == "Light") {
   rainstatusElement.style.color = "#00FF00"; // Green
} else if (rainstatus == "Moderate") {
   rainstatusElement.style.color = "#FFFF00"; // Yellow
} else if (rainstatus == "Heavy") {
   rainstatusElement.style.color = "#FFA500"; // Orange
} else if (rainstatus == "Very Heavy"){
   rainstatusElement.style.color = "#FF0000"; // Red
}

var chart = new Chart(document.getElementById('chartrain'), {
   type: 'line',
   data: {
      labels : datasx,
      datasets: [
                {
                label: 'in mm',
                data: datas,
                fill: false,
                borderColor: '#f55d7a',
                tension: 0.1
                }]
            },

        'options':{
            animation: {
                duration: 100,
                easing: 'easeOutBounce'
            },
            scales:{
                yAxes: [{
                    display:true,
                    stacked:true,
                    ticks:{
                        min:0, //minimum value
                        max:100 //maximum value
                    }
                }]
            }
        },

    });
});

// data water flow
$.getJSON('/route/waterflowroute', function(blocksall) {
    var datas = blocksall.blocks.map(Number);
    datas = datas.reverse();
    console.log(datas)

    var datasx = blocksall.blocks2.map(String);
    datasx = datasx.reverse();
    console.log(datasx)

    var waterFlowValue = datas[datas.length - 1].toFixed(2);
    document.getElementById("wtrflowvalue").innerHTML = waterFlowValue;
    setTextColor("wtrflowvalue", waterFlowValue);

    function setTextColor(elementId, value) {
        var color;
        if (value <= 10) {
            color = "#00FF00"; // Green
        } else if (value >= 11 && value < 30) {
            color = "#FFFF00"; // Yellow
            } else if (value >= 31 && value < 60) {
                color = "#FFA500"; // Orange
            } else {
                color = "#FF0000"; // Red
            }
            document.getElementById(elementId).style.color = color;
        }
    });
}

// Add your map initialization logic here
var map = L.map('map').setView([3.546468, 103.428211], 13);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; OpenStreetMap contributors'
}).addTo(map);

// Add a marker for the location
var marker = L.marker([3.546468, 103.428211]).addTo(map)
.bindPopup('Fakulti Komputeran UMP Pekan Pahang')
.openPopup();



// FUZZY RESULT
Echo.channel('fuzzyControlDam').listen('FuzzyEventDam', (e) => {
    // console.log(1);
    // console.log(e['Pred0']);

    let fuzzyResultElement = document.getElementById("fuzzyresult");
    let fuzzyStatus = "";
    let fuzzyColor = "";

    if (e['Pred0'] <= 65.5) {
        fuzzyStatus = "Normal";
        fuzzyColor = "#00FF00"; // Green
    } else if (e['Pred0'] >= 65.6 && e['Pred0'] <= 75) {
        fuzzyStatus = "Moderate";
        fuzzyColor = "#FFFF00"; // Yellow
    } else if (e['Pred0'] > 76) {
        fuzzyStatus = "Severe";
        fuzzyColor = "#FF0000"; // Red
    }

    fuzzyResultElement.innerHTML = fuzzyStatus;
    fuzzyResultElement.style.color = fuzzyColor;
});
```

```
   // Button: Spillway Gate Status ON/OFF
   Echo.channel('GateStatus').listen('ControlGateStatus', (e) => {
      if (e['status'] == 1) { //e[status] because the same in raspberry pi
           document.getElementById("ActuatorStatus").innerHTML = 'Spillway gate is
open';
         document.getElementById("ActuatorStatus" ).style.color= "#00FF00";
      } else {
           document.getElementById("ActuatorStatus").innerHTML = 'Spillway gate is
close';
         document.getElementById("ActuatorStatus" ).style.color= "#FF0000";
      }
   });


   $.getJSON('/route/alarm', function(blocksall){
      // console.log(1);
      //console.log(blocksall.blocks)
      var datas = blocksall.blocks.map(String);
      datas = datas.reverse();
      // console.log(datas)
      var datasx = blocksall.blocks2.map(String);
      datasx = datasx.reverse();

      var table = document.getElementById("myTable");

      for (let step = 0; step < datas.length; step++) {
         // console.log(datas[step]);
         var row = table.insertRow(0);
         var cell1 = row.insertCell(0);
         var cell2 = row.insertCell(1);
         cell1.innerHTML = datas[step];
         cell2.innerHTML = datasx[step];
      }
   });

   Echo.channel('DeviceAlarm').listen('AlarmStatDam', (e) => {
      var table = document.getElementById("myTable");
      var row = table.insertRow(0);
      var cell1 = row.insertCell(0);
      var cell2 = row.insertCell(1);
      cell1.innerHTML = e['DeviceID'];
      cell2.innerHTML = new Date().toLocaleString('en-CA', {hour12: false,});
      document.getElementById("myAlert").innerHTML = 'Alarm Trigger!!';
      document.getElementById("myAlert" ).style.color = "#FF0000";
   });
</script>
@endsection
```

api.php

```
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/

Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
    return $request->user();
});

Route::get('myWaterLevellink','App\Http\Controllers\apicontroller@APIlistWaterlev
el');//myNoiselink adalah nama utk bukak di browser
Route::get('myRainfalllink','App\Http\Controllers\apicontroller@APIlistRainfall');
```

web.php

```
<?php

use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Route::group(['middleware' => 'auth'], function () {

    Route::get('/',function(){
        return view('welcome');
    });

     Route::get('/home', function () { //dashboard is the one that we will type in laravel
browser
            return view('dashboard'); //dashboard is the one that is called to display in
laravel browser
    });

      Route::get('/dashboard', function () { //dashboard is the one that we will type in
laravel browser
            return view('dashboard'); //dashboard is the one that is called to display in
laravel browser
    });

    Route::post('/opengate',function(){
        $message = 'on';
        event(new App\Events\ControlDeviceDam($message));
        return redirect('dashboard');
    });

    Route::post('/closegate',function(){
        $message = 'off';
        event(new App\Events\ControlDeviceDam($message));
        return redirect('dashboard');
    });

    Route::get('/route/rainfallroute','App\Http\Controllers\dbcontroller@getRainfall');

Route::get('/route/waterlevelroute','App\Http\Controllers\dbcontroller@getWaterlevel
');

Route::get('/route/waterflowroute','App\Http\Controllers\dbcontroller@getWaterflow'
);
    Route::get('/route/alarm','App\Http\Controllers\alarmcontroller@getAlarm');

});

Auth::routes();

Route::get('/home',                    [App\Http\Controllers\HomeController::class,
'index'])->name('home');
```

alarmtodb.py

```
# -*- coding: utf-8 -*-
"""
```

```
Created on Tue Mar  1 19:42:32 2022

@author: user
"""



import json
import mysql.connector
import paho.mqtt.client as mqtt
import requests
import pusher

pusher_client          =          pusher.Pusher(app_id='1',          key=u'umpfkpusher',
secret=u'u%M15z2h%3A', cluster=u'mt1', ssl=False, host=u'10.26.30.32', port=6001)
# MQTT Settings
# MQTT_Broker = "test.mosquitto.org"
MQTT_Broker = "10.26.30.33"
MQTT_Port = 1883
Keep_Alive_Interval = 45
MQTT_Topic = "DAM/#"


mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="",
  database="dammonitoring"
)

mycursor = mydb.cursor()

# Function to save Humidity to DB Table
def goToWaterlevel(jsonData):

    Value = float(json.loads(jsonData))
    print(Value)

    if (Value > 200):
        # if there is alarm trigger, data will go this flow
        try:
                    pusher_client.trigger(u'DeviceAlarm',  u'App\Events\AlarmStatDam',
{u'DeviceID': 'Severe'})
        except:
            print("Danger")
        TOKEN = "5978153873:AAF1vy6kJurqmRt1wsLVHR7Wbb2HR4dboEw"
        chat_id = "-947338938"
        message = "System Alert : Alert Dam will be broken"
                                                                        url          =
f"https://api.telegram.org/bot{TOKEN}/sendMessage?chat_id={chat_id}&text={mes
sage}"
        requests.get(url).json()
        sql = "INSERT INTO alarm (id,DeviceID) VALUES (%s,%s)"
        val = ("",'Severe')
        mycursor.execute(sql, val)
        mydb.commit()
        print('Alarm trigger')
    else:
    # send the sensor value
        sql = "INSERT INTO waterlevel (id,Value) VALUES (%s,%s)"
        val = ('',Value)

        mycursor.execute(sql, val)
        mydb.commit()
        print("Water Level Data Stored")


#=================================================================
===
# Master Function to Select DB Funtion based on MQTT Topic

def sensor_Data_Handler(Topic, jsonData):
    if Topic ==  'DAM/waterlevel':
        goToWaterlevel(jsonData)

#=================================================================
===
#Subscribe to all Sensors at Base Topic
def on_connect(mosq, obj,flag, rc):
    mqttc.subscribe(MQTT_Topic, 0)

#Save Data into DB Table
def on_message(mosq, obj, msg):
    print ("MQTT Data Received...")
    # print ("MQTT Topic: " + msg.topic)
    # print ("Data: " + str(msg.payload))
    sensor_Data_Handler(msg.topic, msg.payload)

def on_subscribe(mosq, obj, mid, granted_qos):
    pass


mqttc = mqtt.Client()
# Assign event callbacks
mqttc.username_pw_set("umpfk", "u4h%w1Tr12")
mqttc.on_message = on_message
mqttc.on_connect = on_connect
mqttc.on_subscribe = on_subscribe
```

```
# Connect

mqttc.connect(MQTT_Broker, int(MQTT_Port), int(Keep_Alive_Interval))
# Continue the network loop
mqttc.loop_forever()

#=============================================================
===
```

fuzzytrain.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu May 11 10:31:42 2023

@author: user
"""

# -*- coding: utf-8 -*-
"""
Created on Mon Dec 12 01:22:19 2022

@author: user
"""


import numpy as np
from skfuzzy import control as ctrl
import skfuzzy as fuzz
import pickle

# Define the fuzzy sets for the inputs
water_level = ctrl.Antecedent(np.arange(180, 211, 1), 'water_level')  # Range: 0-210
rain_intensity = ctrl.Antecedent(np.arange(1, 101, 1), 'rain_intensity')  # Range:
0-100

# Define the fuzzy sets for the output
condition = ctrl.Consequent(np.arange(0, 101, 1), 'condition')

water_level['normal'] = fuzz.trimf(water_level.universe, [180, 185, 190]) # min, max,
overlap
water_level['alert'] = fuzz.trimf(water_level.universe, [186, 190, 197])
water_level['warning'] = fuzz.trimf(water_level.universe, [191, 200, 204])
water_level['danger'] = fuzz.trimf(water_level.universe, [201, 210, 210])

rain_intensity['light'] = fuzz.trimf(rain_intensity.universe, [1, 10, 23])
rain_intensity['moderate'] = fuzz.trimf(rain_intensity.universe, [11, 30, 46])
rain_intensity['heavy'] = fuzz.trimf(rain_intensity.universe, [31, 60, 75])
```

```
rain_intensity['very_heavy'] = fuzz.trimf(rain_intensity.universe, [61, 100, 100])

water_level.view()
rain_intensity.view()

# percentage
condition['normal'] = fuzz.trimf(condition.universe, [0, 40, 52])
condition['moderate'] = fuzz.trimf(condition.universe, [41, 75, 85])
condition['severe'] = fuzz.trimf(condition.universe, [76, 100, 100])

condition.view()

# Define the rule base using the 'rule' method
# Rule for normal
rule1 = ctrl.Rule(water_level['normal'] & rain_intensity['light'], condition['normal'])
rule2 = ctrl.Rule(water_level['normal'] & rain_intensity['moderate'],
condition['normal'])
rule3 = ctrl.Rule(water_level['alert'] & rain_intensity['light'], condition['normal'])

# Rule for moderate
rule4 = ctrl.Rule(water_level['normal'] & rain_intensity['heavy'],
condition['moderate'])
rule5 = ctrl.Rule(water_level['alert'] & rain_intensity['moderate'],
condition['moderate'])
rule6 = ctrl.Rule(water_level['alert'] & rain_intensity['heavy'], condition['moderate'])
rule7 = ctrl.Rule(water_level['warning'] & rain_intensity['light'],
condition['moderate'])
rule8 = ctrl.Rule(water_level['warning'] & rain_intensity['moderate'],
condition['moderate'])
rule9 = ctrl.Rule(water_level['warning'] & rain_intensity['heavy'],
condition['moderate'])

# Rule for severe
rule10 = ctrl.Rule(water_level['normal'] & rain_intensity['very_heavy'],
condition['severe'])
rule11 = ctrl.Rule(water_level['alert'] & rain_intensity['very_heavy'],
condition['severe'])
rule12 = ctrl.Rule(water_level['warning'] & rain_intensity['very_heavy'],
condition['severe'])
rule13 = ctrl.Rule(water_level['danger'] & rain_intensity['light'], condition['severe'])
rule14 = ctrl.Rule(water_level['danger'] & rain_intensity['moderate'],
condition['severe'])
rule15 = ctrl.Rule(water_level['danger'] & rain_intensity['heavy'], condition['severe'])
rule16 = ctrl.Rule(water_level['danger'] & rain_intensity['very_heavy'],
condition['severe'])

# Create the control system using the rules
machine_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6,
rule7,rule8,
                    rule9, rule10, rule11, rule12, rule13,rule14, rule15, rule16])

# Create a control system simulation
machine = ctrl.ControlSystemSimulation(machine_ctrl)

# # Pass inputs to the control system using Antecedent labels with Pythonic API
# # Note: if you like passing many inputs all at once, use .inputs(dict_of_data)
machine.input['water_level'] = 195
machine.input['rain_intensity'] = 46


# # Crunch the numbers
machine.compute()

# # Print the result in a human-friendly form
print(machine.output['condition'])


file = open('fuzzybraindam.picl', 'wb')
pickle.dump(machine,file)
file.close()
```

subscribefuzzy.py

```
import paho.mqtt.client as mqtt
import json
import mysql.connector
import numpy as np
from datetime import datetime
from skfuzzy import control as ctrl
import skfuzzy as fuzz
import pickle
import joblib
import time
import pusher
import requests

pusher_client = pusher.Pusher(app_id='1', key=u'umpfkpusher',
secret=u'u%M15z2h%3A', cluster=u'mt1', ssl=False, host=u'10.26.30.32', port=6001)


# Function to save  to DB Table
def goToAPI():
  try:
    # api-endpoint
    URLwater_level = "http://10.66.49.255:8000/api/myWaterLevellink"
    rwater_level = requests.get(URLwater_level) # we want to access html
```

```
      datawater_level = rwater_level.json()
      wtrlvlresult = datawater_level['blocks']
       wtrlvlresult = np.float32(wtrlvlresult) # flooat because maybe data came in type
of string (convert the data)

      URLrain_intensity = "http://10.66.49.255:8000/api/myRainfalllink"
      rrain_intensity = requests.get(URLrain_intensity)
      datarain_intensity = rrain_intensity.json()
      rainresult = datarain_intensity['blocks']
      rainresult = np.float32(rainresult)

      objectRep = open("fuzzybraindam.picl", "rb")

      mynet = pickle.load(objectRep)

        # # # Pass inputs to the control system using Antecedent labels with Pythonic
API
      # # # Note: if you like passing many inputs all at once, use .inputs(dict_of_data)
      mynet.input['water_level'] = wtrlvlresult # / 500
      mynet.input['rain_intensity'] = rainresult

      mynet.compute()
      # result = mynet.output()
      # Print the result in a human-friendly form
      # print(mynet.output['condition'])

      aaa = mynet.output['condition']
      print(aaa[0])
      print(str(wtrlvlresult[0]))
      print(str(rainresult[0]))
      # print(str(result[0]))
           pusher_client.trigger(u'fuzzyControlDam',  u"App\Events\FuzzyEventDam",
{u'Pred0': str(aaa[0])})
      print('publish to socket')

   except:
      print('retry')

while 1:
   goToAPI()
   time.sleep(10)
```