



Faculty of Computer Science and Information Technology

***IOT-BASED CRACK DETECTION AND MONITORING SYSTEM***

Nur Raiyani Binti Mohd Yusri Azhar

Bachelor of Computer Science with Honours (Network Computing)

2025

# **IOT-BASED CRACK DETECTION AND MONITORING SYSTEM**

**NUR RAIYANI BINTI MOHD YUSRI AZHAR**

This project is submitted in partial fulfilment of the  
requirements for the degree of  
Bachelor of Computer Science with Honors (Network Computing)

Faculty of Computer Science and Information Technology  
UNIVERSITI MALAYSIA SARAWAK

2025

# **SISTEM PENGESANAN DAN PEMANTAUAN RETAK BERASASKAN IOT**

**NUR RAIYANI BINTI MOHD YUSRI AZHAR**

Projek ini merupakan salah satu keperluan untuk  
Ijazah Sarjana Muda Sains Komputer dan Teknologi Maklumat  
(Pengkomputeran Rangkaian)

Fakulti Sains Komputer dan Teknologi Maklumat

**UNIVERSITI MALAYSIA SARAWAK**

2025

## DECLARATION

I, Nur Raiyani Binti Mohd Yusri Azhar with the matric number of 80685, hereby solemnly declare that the report titled '**IoT-based Crack Detection and Monitoring System**' is prepared and completed authentically by me in partial fulfilment of the requirements for the degree of Bachelor of Computer Science with Honours (Network Computing) is a record of an original work carried out by me under the guidance of Madam Seleviawati Binti Tarmizi. I further declare that the work reported in this project has not been previously or concurrently submitted for any other degree at UNIVERSITI MALAYSIA SARAWAK (UNIMAS) or any other institutions. I declare that I have not copied or plagiarize from any other sources except for quotations and citations which have been duly acknowledged.



-----  
Nur Raiyani Binti Mohd Yusri Azhar (80685)

Faculty of Computer Science and Information Technology

UNIVERSITI MALAYSIA SARAWAK

## **ACKNOWLEDGEMENT**

Bismillahirrahmannirrahim, first and foremost, all praises to Allah SWT, who gave me the strength to complete this Final Year Project and for guiding me through all the difficulties every day.

I want to express my deepest appreciation to my supervisor, Madam Seleviawati Binti Tarmizi, for every single bit of assistance, encouragement, kindness, and support. Her knowledge and insights have significantly enriched my learning experience and enhanced the quality of my project.

I would also like to express my appreciation to my family, especially my parents and siblings, who have always supported me and have been my driving force throughout my academic journey. I also want to extend my sincere gratitude to my friends who have stood by me, sharing this journey and giving support in countless ways, which has motivated me throughout the preparation of this report.

In a nutshell, I would love to express heartfelt thanks to every individual who has helped me, either indirectly or directly, throughout this project. Their collective encouragement has rewarded me with the successful completion of this report.

## ABSTRACT

Monitoring structural health is crucial for maintaining the reliability and safety of buildings and structures. Existing methods, such as manual and partially automated physical checks, tend to be time-consuming and highly susceptible to oversights and the absence of real-time monitoring functionality. This project proposes an Internet of Things (IoT)-based crack detection and monitoring system that utilizes wireless sensors and cloud-based systems to deliver continuous and real-time structural monitoring with the purpose of solving these shortcomings. ESP32 microcontrollers and ultrasonic sensors are incorporated into the system for identifying cracks. Data is subsequently sent to a cloud platform for evaluation and display via a web interface. The system's development process, particularly focused on affordability, adaptability, and intuitive interface design, was executed adopting the Agile methodology. By eliminating the necessity for physical checks, optimizing the precision of initial crack detection, and facilitating preventative maintenance, the proposed strategy eventually decreases the probability of building collapses and maintenance operating costs. The system's capacity for continuous monitoring signifies an essential development in health and safety along with preventative maintenance, especially for tall buildings. The project indicates how the Internet of Things (IoT) could transform current infrastructure management by allowing data-driven choices that optimize resources and guarantee safety for everyone.

## ABSTRAK

Memantau kesihatan struktur adalah penting untuk mengekalkan kebolehpercayaan dan keselamatan bangunan dan struktur. Kaedah yang sedia ada, seperti pemeriksaan fizikal manual dan separa automatik, cenderung memakan masa dan sangat terdedah kepada kesilapan serta ketiadaan fungsi pemantauan masa nyata. Projek ini mencadangkan sistem pengesanan dan pemantauan retak berasaskan “Internet of Things” (IoT) yang menggunakan sensor tanpa wayar dan sistem Cloud untuk memberikan pemantauan struktur secara berterusan dan masa nyata dengan tujuan menyelesaikan kekurangan-kekurangan ini. Mikropengawal ESP32 dan sensor ultrasonik dimasukkan ke dalam sistem untuk mengenal pasti retakan. Data kemudiannya dihantar ke platform Cloud untuk penilaian dan paparan melalui antara muka web. Proses pembangunan sistem, yang khususnya memberi tumpuan kepada kemampuan milik, kebolehsuaian, dan reka bentuk antara muka yang intuitif, dilaksanakan dengan menggunakan metodologi Agile. Dengan menghapuskan keperluan untuk pemeriksaan fizikal, mengoptimumkan ketepatan pengesanan retak awal, dan memudahkan penyelenggaraan pencegahan, strategi yang dicadangkan akhirnya mengurangkan kebarangkalian runtuhnya bangunan dan kos operasi penyelenggaraan. Keupayaan sistem untuk pemantauan berterusan menandakan perkembangan penting dalam kesihatan dan keselamatan serta penyelenggaraan pencegahan, terutamanya untuk bangunan tinggi. Projek ini menunjukkan bagaimana IoT boleh mengubah pengurusan infrastruktur semasa dengan membenarkan pilihan berasaskan data yang mengoptimumkan sumber dan menjamin keselamatan untuk semua.

## TABLE OF CONTENTS

|                                                                                                     |      |
|-----------------------------------------------------------------------------------------------------|------|
| DECLARATION .....                                                                                   | i    |
| ACKNOWLEDGEMENT .....                                                                               | ii   |
| ABSTRACT .....                                                                                      | iii  |
| ABSTRAK .....                                                                                       | iv   |
| LIST OF TABLES .....                                                                                | viii |
| LIST OF FIGURES .....                                                                               | ix   |
| LIST OF ABBREVIATIONS .....                                                                         | xi   |
| CHAPTER 1: INTRODUCTION .....                                                                       | 12   |
| 1.1 Project Title .....                                                                             | 12   |
| 1.2 Introduction .....                                                                              | 12   |
| 1.3 Problem Statement .....                                                                         | 13   |
| 1.4 Scope .....                                                                                     | 15   |
| 1.5 Aims and Objectives .....                                                                       | 15   |
| 1.6 Brief Methodology .....                                                                         | 16   |
| 1.7 Significance of Project .....                                                                   | 18   |
| 1.8 Project Schedule .....                                                                          | 19   |
| 1.9 Expected Outcome .....                                                                          | 19   |
| CHAPTER 2: LITERATURE REVIEW .....                                                                  | 21   |
| 2.1 Introduction .....                                                                              | 21   |
| 2.2 Review on Similar Existing System .....                                                         | 21   |
| 2.2.1 Computer Vision-Based Crack Detection Methods .....                                           | 22   |
| 2.2.2 Ultrasonic System for Structural Health Monitoring .....                                      | 23   |
| 2.2.3 Structural Health Monitoring Technique for Crack Detection Using<br>Strain Gauge System ..... | 24   |
| 2.3 Tools and Technology Used .....                                                                 | 25   |
| 2.4 Comparison between Existing System .....                                                        | 27   |
| 2.5 Discussion .....                                                                                | 28   |
| 2.6 Summary .....                                                                                   | 29   |
| CHAPTER 3: REQUIREMENT ANALYSIS AND DESIGN .....                                                    | 30   |
| 3.1 Introduction .....                                                                              | 30   |
| 3.2 Methodology .....                                                                               | 30   |
| 3.3 Requirement Elicitation .....                                                                   | 31   |
| 3.4 Software and Hardware Requirements .....                                                        | 39   |
| 3.4.1 Software Requirements .....                                                                   | 39   |
| 3.4.2 Hardware Requirements .....                                                                   | 40   |
| 3.5 Functional and Non-functional Requirements .....                                                | 40   |

|                                                            |    |
|------------------------------------------------------------|----|
| 3.5.1 Functional Requirements.....                         | 40 |
| 3.5.2 Non-functional Requirements .....                    | 41 |
| 3.6 System Design.....                                     | 41 |
| 3.6.1 Context Diagram.....                                 | 42 |
| 3.6.2 Level 0 Diagram.....                                 | 43 |
| 3.6.3 Level 1 Diagram.....                                 | 44 |
| 3.6.4 Database Design .....                                | 48 |
| 3.6.5 System Architecture .....                            | 49 |
| 3.6.6 Interface Design .....                               | 51 |
| 3.7 Implementation.....                                    | 54 |
| 3.8 Testing .....                                          | 55 |
| 3.9 Deployment.....                                        | 55 |
| 3.10 Review .....                                          | 55 |
| 3.11 Summary.....                                          | 56 |
| CHAPTER 4: IMPLEMENTATION.....                             | 57 |
| 4.1 Introduction .....                                     | 57 |
| 4.2 Proposed System Implementation.....                    | 57 |
| 4.2.1 Setup of Firebase.....                               | 57 |
| 4.2.2 Setup of ThingSpeak.....                             | 60 |
| 4.2.3 Installation and Setup of Arduino IDE.....           | 64 |
| 4.2.4 Installation and Setup of Visual Studio Code .....   | 68 |
| 4.2.5 Setup of EmailJS and Twilio.....                     | 69 |
| 4.3 Workflow of the Proposed System .....                  | 72 |
| 4.4 System Feature and Operation .....                     | 73 |
| 4.4.2 Real-time crack data monitoring.....                 | 73 |
| 4.4.3 Cloud-based data visualization .....                 | 75 |
| 4.4.4 Threshold detection and alert trigger.....           | 76 |
| 4.4.5 Firebase hosting and web application deployment..... | 80 |
| 4.5 Summary .....                                          | 80 |
| CHAPTER 5: TESTING.....                                    | 82 |
| 5.1 Introduction .....                                     | 82 |
| 5.2 Functional Testing.....                                | 82 |
| 5.2.1 Unit Testing .....                                   | 83 |
| 5.2.2 Integration Testing.....                             | 87 |
| 5.3 Non-Functional Testing.....                            | 91 |
| 5.3.1 Reliability Testing.....                             | 91 |
| 5.3.2 Performance Testing .....                            | 93 |

|                                              |     |
|----------------------------------------------|-----|
| 5.3.3 Usability Testing .....                | 95  |
| 5.4 Discussion on Results .....              | 102 |
| 5.5 Summary .....                            | 103 |
| CHAPTER 6: CONCLUSION AND FUTURE WORKS ..... | 104 |
| 6.1 Introduction .....                       | 104 |
| 6.2 Objectives and Achievements .....        | 104 |
| 6.2.1 Discussion on Achievements .....       | 105 |
| 6.3 Project Limitations and Constraints..... | 106 |
| 6.4 Future Works .....                       | 109 |
| 6.5 Conclusion .....                         | 110 |
| REFERENCES .....                             | 112 |
| APPENDICES.....                              | 115 |
| Appendix A.....                              | 115 |
| Appendix B.....                              | 117 |

## LIST OF TABLES

|                                                                                                  |     |
|--------------------------------------------------------------------------------------------------|-----|
| Table 1.1: Project Schedule for IoT-Based Crack Detection and Monitoring System .....            | 19  |
| Table 2.1: Reviewed System Comparison .....                                                      | 27  |
| Table 3.1: Software Requirements for Development .....                                           | 39  |
| Table 3.2: Hardware Requirements for Development.....                                            | 40  |
| Table 5.1: Unit Test Case for Sensor Reading Module .....                                        | 83  |
| Table 5.2: Unit Test Case for Wi-Fi Connection .....                                             | 84  |
| Table 5.3: Unit Test Case for ThingSpeak Transmission .....                                      | 84  |
| Table 5.4: Unit Test Case for User Login.....                                                    | 85  |
| Table 5.5: Unit Test Case for Firebase Web Display .....                                         | 85  |
| Table 5.6: Unit Test Case for Settings Page .....                                                | 86  |
| Table 5.7: Unit Test Case for Data History Page .....                                            | 87  |
| Table 5.8: Unit Test Case for Logout .....                                                       | 87  |
| Table 5.9: Integration Test Case for Hardware to Cloud Platform.....                             | 88  |
| Table 5.10: Integration Test Case for IoT Crack Detection and Monitoring System Web<br>App ..... | 89  |
| Table 5.11: Test Cases for Reliability Testing .....                                             | 92  |
| Table 5.12: Performance Test Cases for Crack Depth Detection Accuracy.....                       | 94  |
| Table 6.1: Summary of Objectives and Achievements .....                                          | 104 |

## LIST OF FIGURES

|                                                                                            |    |
|--------------------------------------------------------------------------------------------|----|
| Figure 1.1: Agile Methodology Phases.....                                                  | 16 |
| Figure 2.1: Procedure of CNN Deep Learning-based Crack Detection.....                      | 23 |
| Figure 2.2: Framework of the Ultrasonic Wave-based Structural Health .....                 | 24 |
| Figure 2.3: Framework of the Structural Health Monitoring using Strain Gauge Sensors ..... | 25 |
| Figure 3.1: The First Question for Section B.....                                          | 32 |
| Figure 3.2: The Second Question for Section B.....                                         | 33 |
| Figure 3.3: The Third Question for Section B.....                                          | 33 |
| Figure 3.4: The Fourth Question for Section B.....                                         | 34 |
| Figure 3.5: The Fifth Question for Section B.....                                          | 34 |
| Figure 3.6: The First Question for Section C.....                                          | 35 |
| Figure 3.7: The Second Question for Section C.....                                         | 36 |
| Figure 3.8: The Third Question for Section C.....                                          | 36 |
| Figure 3.9: The Fourth Question for Section C.....                                         | 37 |
| Figure 3.10: The Fifth Question for Section C.....                                         | 37 |
| Figure 3.11: The Sixth Question for Section C.....                                         | 38 |
| Figure 3.12: Context Diagram of IoT-based Crack Detection and Monitoring System.....       | 42 |
| Figure 3.13: Level 0 of IoT-based Crack Detection and Monitoring System.....               | 43 |
| Figure 3.14: Level 1 for Process Receive Sensor Data .....                                 | 44 |
| Figure 3.15: Level 1 for Process Analyse Data .....                                        | 45 |
| Figure 3.16: Level 1 for Process Generate Alerts.....                                      | 46 |
| Figure 3.17: Level 1 for Process View Data.....                                            | 47 |
| Figure 3.18: Entity Relationship Diagram (ERD).....                                        | 48 |
| Figure 3.19: Hardware and Software Connection of the Proposed System.....                  | 50 |
| Figure 3.20: User Authentication Page.....                                                 | 51 |
| Figure 3.21: Dashboard Page.....                                                           | 52 |
| Figure 3.22: Data History Page.....                                                        | 53 |
| Figure 3.23: Settings Page .....                                                           | 53 |
| Figure 4.1: Create a New Project in Firebase Console.....                                  | 58 |
| Figure 4.2: Displays when the Project is Successfully Created.....                         | 58 |
| Figure 4.3: Activating Firebase Hosting for the Project.....                               | 59 |
| Figure 4.4: Run a Command to Install Firebase CLI .....                                    | 59 |
| Figure 4.5: Setup of Firebase Hosting for the Project .....                                | 60 |
| Figure 4.6: Deploying the Firebase Hosting Web Application.....                            | 60 |
| Figure 4.7: Creating a Free MathWorks Account to Access ThingSpeak .....                   | 61 |
| Figure 4.8: New Channel Created in ThingSpeak.....                                         | 61 |
| Figure 4.9: Setup of Channel Settings.....                                                 | 62 |
| Figure 4.10: Write API Key for ThingSpeak.....                                             | 62 |
| Figure 4.11: Update the Channel Sharing Settings to Public.....                            | 62 |
| Figure 4.12: Line Graph of Real-Time Monitoring Data.....                                  | 63 |
| Figure 4.13: Update the Display Setting for the Line Graph.....                            | 63 |
| Figure 4.14: Block of Code to Display the Graph in the Web Interface.....                  | 64 |
| Figure 4.15: The Latest Version of Arduino IDE.....                                        | 64 |
| Figure 4.16: Complete Installation of Arduino IDE.....                                     | 65 |
| Figure 4.17: Installation of ESP32 board by Espressif Systems.....                         | 65 |
| Figure 4.18: Choosing the Accurate Tools that are used for the Project .....               | 66 |
| Figure 4.19: Libraries that were Installed for the Project.....                            | 67 |
| Figure 4.20: Compiling and Uploading the Code to ESP32 .....                               | 67 |
| Figure 4.21: Messages Displayed by Serial Monitor.....                                     | 68 |
| Figure 4.22: Installation of Visual Studio Code (VS Code) .....                            | 68 |

|                                                                                               |     |
|-----------------------------------------------------------------------------------------------|-----|
| Figure 4.23: Created New Email Service using Gmail .....                                      | 69  |
| Figure 4.24: Customized an Email Template for Crack Alerts Message .....                      | 70  |
| Figure 4.25: Code to Initiate the Email Alert in JavaScript .....                             | 70  |
| Figure 4.26: Twilio Account Information .....                                                 | 71  |
| Figure 4.27: Code to Initiate the SMS Alert in JavaScript .....                               | 71  |
| Figure 4.28: loop() Function to Read the Crack Depth .....                                    | 73  |
| Figure 4.29: Code for the Real-Time Crack Data Monitoring .....                               | 74  |
| Figure 4.30: Dashboard Page that Displays the Real-Time Data .....                            | 75  |
| Figure 4.31: Code that Displays the ThingSpeak Line Graph .....                               | 76  |
| Figure 4.32: The Live Sensor Data Being Displayed on the Dashboard Page .....                 | 76  |
| Figure 4.33: Code for checkCrackDepth() Function for the Threshold Detection<br>Feature ..... | 77  |
| Figure 4.34: Code for Email Alert Notification .....                                          | 77  |
| Figure 4.35: Code for SMS Alert Notification .....                                            | 77  |
| Figure 4.36: SMS Server Console Output .....                                                  | 78  |
| Figure 4.37: Email Alert Notification Message .....                                           | 78  |
| Figure 4.38: SMS Alert Notification Message .....                                             | 78  |
| Figure 4.39: Customization of Threshold Value and Notification Method .....                   | 79  |
| Figure 4.40: SMS Chosen as Notification Method .....                                          | 79  |
| Figure 4.41: Deployment of Firebase Hosting .....                                             | 80  |
| Figure 5.1: The First Question for Section B .....                                            | 96  |
| Figure 5.2: The Second Question for Section B .....                                           | 97  |
| Figure 5.3: The Third Question for Section B .....                                            | 97  |
| Figure 5.4: The First Question for Section C .....                                            | 98  |
| Figure 5.5: The Second Question for Section C .....                                           | 98  |
| Figure 5.6: The Third Question for Section C .....                                            | 98  |
| Figure 5.7: The First Question for Section D .....                                            | 99  |
| Figure 5.8: The Second Question for Section D .....                                           | 100 |
| Figure 5.9: The Third Question for Section D .....                                            | 100 |
| Figure 5.10: The Fourth Question for Section D .....                                          | 101 |

## LIST OF ABBREVIATIONS

|              |                                    |
|--------------|------------------------------------|
| <b>API</b>   | Application Programming Interface  |
| <b>CDN</b>   | Content Delivery Network           |
| <b>CNN</b>   | Convolutional Neural Networks      |
| <b>COM</b>   | Communication Port                 |
| <b>CSS</b>   | Cascading Style Sheets             |
| <b>DFD</b>   | Data Flow Diagram                  |
| <b>ERD</b>   | Entity Relationship Diagram        |
| <b>GND</b>   | Ground                             |
| <b>GPIO</b>  | General Purpose Input/Output       |
| <b>HTML</b>  | Hyper Text Markup Language         |
| <b>HTTPS</b> | HyperText Transfer Protocol Secure |
| <b>IoT</b>   | Internet of Things                 |
| <b>PDF</b>   | Portable Document Format           |
| <b>SDK</b>   | Software Development Kit           |
| <b>SID</b>   | Session Identifier                 |
| <b>SMS</b>   | Short Message Service              |
| <b>SPA</b>   | Single Page Application            |
| <b>SSL</b>   | Secure Sockets Layer               |
| <b>TLS</b>   | Transport Layer Security           |
| <b>UAT</b>   | User Acceptance Testing            |
| <b>URL</b>   | Uniform Resource Locator           |
| <b>USB</b>   | Universal Serial Bus               |
| <b>VCC</b>   | Voltage Common Collector           |
| <b>Wi-Fi</b> | Wireless Fidelity                  |

## **CHAPTER 1: INTRODUCTION**

### **1.1 Project Title**

IoT-Based Crack Detection and Monitoring System

### **1.2 Introduction**

Natural disasters often occur near areas where people live and work, impacting buildings and infrastructure. These events may arise from natural forces or human actions. With the rapid pace of urbanization, the density of building networks has continuously increased, and the number of high-rise buildings has grown significantly. However, the complexity of tall building environments and their construction processes introduces multiple risk factors. These factors can compromise structural integrity, leading to potentially severe consequences like building collapses, which endanger lives and damage property. Regular approaches to structural inspection, which usually involve semi-automated or manual visual analyses, have downsides such as high expenses, labour-intensiveness, and the likelihood of human error (Sarkom et al., 2023). Therefore, it is essential to implement effective damage detection and early warning systems to monitor the health and stability of buildings, enabling timely intervention and enhancing public safety.

The Internet of Things (IoT) and modern sensor technology are utilized in this project's IoT-Based Crack Detection and Monitoring System to supply constant, real-time structural health monitoring with the aim to solve these issues. To locate the first signs and progression of cracks, the system will be integrated with wireless sensors that are installed strategically on significant parts of a building. These sensors will deliver data to a cloud-based platform for analysis and interpretation over wireless communication protocols. Continuous maintenance decisions based on real-time information are made

possible by the cloud infrastructure, which guarantees that data is readily accessible to stakeholders for visualization and statistical analysis (Rahita et al., 2024).

The advantages of this system go beyond increased safety. In addition to lowering the frequency of manual inspections and minimizing downtime related to conventional inspection techniques, it provides an automated, effective, and economical option for infrastructure maintenance (Butler, 2022). This system can automatically notify engineers and facility managers when crack growth exceeds programmed degrees, enabling prompt action to prevent severe damages (Rahita et al., 2024). The system emphasizes initiatives on buildings displaying initial signs of damage, as well as ensuring safety for people, minimizing maintenance expenses, and allocating resources as efficiently as possible.

The project's integration of IoT technology with structural health monitoring is in line with current engineering methodologies that prioritize predictive maintenance and smart buildings. This system not only eliminates the existing constraints in structural monitoring but additionally establishes a new standard for infrastructure management that utilizes data to make decisions (Revolutionizing structural health monitoring with AI, ML, IoT, and sensor Technology, 2024). This system can show how technology can be used to increase operational effectiveness, public safety, and the lifespan of vital infrastructure.

### **1.3 Problem Statement**

Infrequent manual inspection can cause delays in the detection of cracks and structural weaknesses. Since structural cracks and fractures might develop gradually over time, manual checks are usually inadequate to identify them. When inspections are not performed frequently enough, cracks in a building may go overlooked, increasing the risk of serious harm or even structural collapse.

Furthermore, human error can also occur during manual checks, where people can incorrectly assess and misjudge a subtle but serious structural fault in a building. Such mistakes may result in poor and inaccurate assessments of a building's state, risking safety and failing to address vulnerabilities. Even one minor inspection mistake could turn into a serious threat in crowded places, putting thousands of lives at risk and seriously affecting the areas surrounding it. The implications are especially high in tall buildings, where minor collapse of structures could end up in severe harm that would make evacuation difficult.

Moreover, manual inspections are not capable of continuous monitoring, thereby making it difficult to conduct a constant check of the structural health of a building. The lack of real-time data causes early cracking symptoms to become unnoticed, which is crucial for applying preventative maintenance procedures into practice. The destruction may not be detected until it reaches a critical stage without continuous monitoring, at a point where maintenance is usually more expensive and challenging.

Additionally, some buildings, like bridges and tall buildings, can be especially hard for inspection because of their height and location. The logistical challenges, financial concerns, and potential risks associated with accessing hard-to-reach areas restrict manual inspections of such buildings. As a result, fewer inspections are conducted on these buildings, which increases the possibility of serious structural issues remaining overlooked.

The human labour that is used frequently in current inspection methods is likewise time-consuming and costly. Delays in discovering structural weaknesses might result in higher operating expenses and inefficiencies. The inspection and maintenance cycle are slowed down by the dependency on manual procedures, which increases the financial and safety risks resulting from aging or damaged buildings.

## **1.4 Scope**

This project entails the development of an IoT-based crack detection and monitoring system that utilizes wireless sensors to identify cracks in buildings, enabling a continuous assessment of structural health in real time via a cloud-based platform. The system operates continuously, indicating that it runs at all times, guaranteeing constant monitoring and timely alerts. Additionally, this system is crucial for engineers, relevant authorities, and organizations that are responsible for building safety. The system can be implemented at any stage of a building's lifespan while providing reliable and affordable monitoring. Besides, the system provides real-time data early alerts that could help prevent collapses and reduce maintenance costs. These alerts enable timely interventions and preventive maintenance, which is important for averting potential structural failures, particularly in tall buildings where undetected cracks could have severe consequences.

## **1.5 Aims and Objectives**

1. To design a wireless crack detection system that utilizes sensors to constantly monitor for and detect structural faults, ensuring immediate action.
2. To achieve a 90% accuracy rate in tracking and identifying structural cracks in buildings for accurate data and increased safety.
3. To implement affordable sensors and IoT technology that continuously monitor cracks and provide real-time alerts, which will make it a feasible choice for broad building maintenance adoption.

## 1.6 Brief Methodology

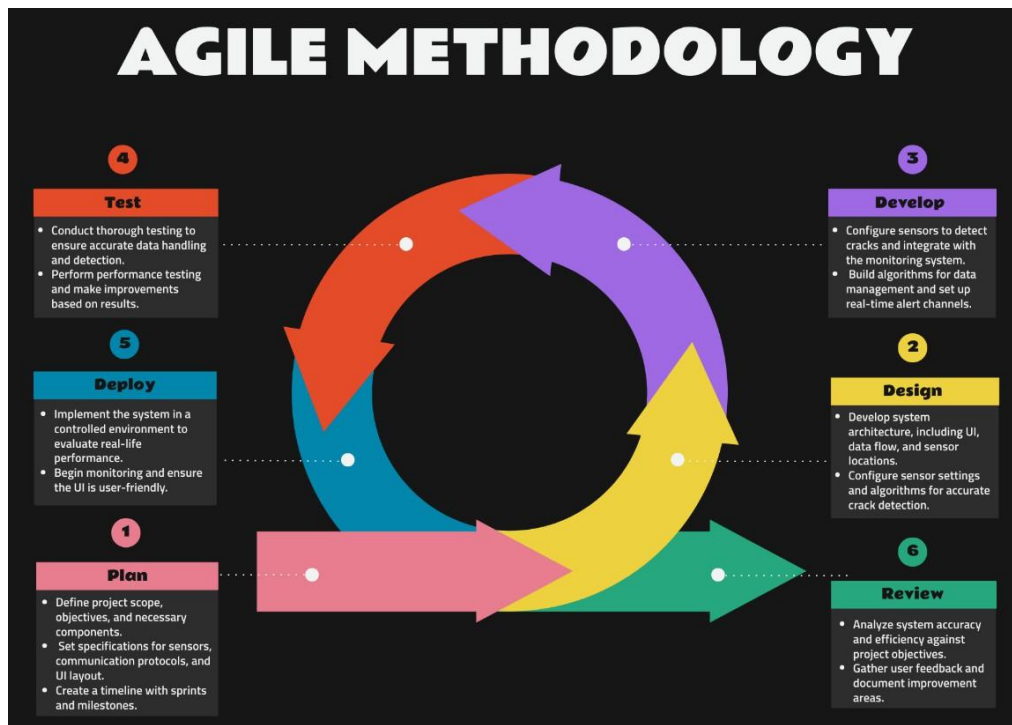


*Figure 1.1: Agile Methodology Phases*

The Agile methodology will be applied in this project with the aim to guarantee iterative development, which enables flexibility and adjustment in accordance with continuous testing and feedback. The six fundamental phases of the approach including planning, designing, developing, testing, deploying, and reviewing, as stated in Figure 1.1, will aid in the system's constant enhancement. This method will provide an organized and flexible procedure for facilitating the development of the IoT-based crack detection and monitoring system.

### 1. Plan

The project's scope, specific objectives along with needed software and hardware components are going to be decided throughout the planning phase. The project specifications, which includes selecting suitable sensors, protocols for communication, and layout of the user interface, will be determined during this

phase. A timeline will be created, breaking the project into smaller, manageable sprints and highlighting the crucial milestones.

## **2. Design**

In this phase, the system architecture is created, specifying the user interface design, data flow processes, and sensor locations. Sensor settings and data algorithms have been setup for precise structural crack recognition, and every component is developed to provide high quality performance for real-time monitoring.

## **3. Develop**

During this phase, sensors are configured to identify cracks and supply data to the monitoring system, integrating both software and hardware components. The system's algorithms are designed for managing data precisely, and channels of communication are set up to deliver warnings in real time.

## **4. Test**

To make sure that the program handles and interprets data precisely, as well as that sensors identify structural changes effectively, thorough testing is performed. Performance testing is carried out to verify that the system functions properly under various scenarios, and improvements can be implemented according to the tests outcomes.

## **5. Deploy**

To assess how well the system operates under real-life situations, it will be implemented in a controlled setting once testing is completed. Monitoring starts, with a focus on the convenience of use and simplicity of the user interface for building managers and engineers.

## **6. Review**

A final review analyses how effectively the system satisfies the project's objectives in terms of both accuracy and efficacy. To ensure that the system is accurate and adaptable, user feedback is collected and areas for enhancement are recorded for further improvements.

### **1.7 Significance of Project**

This IoT-based crack detection and monitoring system is significant because it may have an opportunity to completely transform structural health monitoring, specifically for structures that are potentially vulnerable to slow deterioration. The drawbacks of conventional checking procedures, which are usually inconsistent and rely on manual, visual inspections which could missed subtle but serious cracks, are solved by this project (Babu et al., 2022). This system decreases the risk of unexpected issues and could possibly save lives by ensuring that any emerging weaknesses in structure are noticed early on through persistent, real-time monitoring (Rahita et al., 2024).

Furthermore, through eliminating the need for routine inspections done manually and facilitating more productive, focused maintenance operations, this system gives a cost-efficient alternative (Mathé, 2023). By reducing the procedural burden of regular checks, this IoT technology enhances safety in an efficient and cost-effective manner for buildings in difficult-to-reach places, such tall buildings (Butler, 2022). The system also facilitates decision-making based on data through the integration of IoT technology, which enables authorities and building engineers to base their maintenance plans on accurate and timely information regarding the structural health of the building's structure (The role of IoT technology in optimizing facilities management: features and benefits across buildings, 2024).

## 1.8 Project Schedule

Table 1.1: Project Schedule for IoT-Based Crack Detection and Monitoring System

|                                        | 2024 |     |     | 2025 |     |     |     |     |     |     |
|----------------------------------------|------|-----|-----|------|-----|-----|-----|-----|-----|-----|
| Task                                   | Oct  | Nov | Dec | Jan  | Feb | Mar | Apr | May | Jun | Jul |
| Project Title Selection                |      |     |     |      |     |     |     |     |     |     |
| Preparation of Proposal Draft          |      |     |     |      |     |     |     |     |     |     |
| Submission of Proposal for Review      |      |     |     |      |     |     |     |     |     |     |
| Initial Overview                       |      |     |     |      |     |     |     |     |     |     |
| Research Background                    |      |     |     |      |     |     |     |     |     |     |
| Project Methodology and Requirements   |      |     |     |      |     |     |     |     |     |     |
| System Design and Development          |      |     |     |      |     |     |     |     |     |     |
| System Integration and Initial Testing |      |     |     |      |     |     |     |     |     |     |
| System Deployment and Evaluation       |      |     |     |      |     |     |     |     |     |     |
| Final Testing and Adjustments          |      |     |     |      |     |     |     |     |     |     |
| Summary and Next Steps                 |      |     |     |      |     |     |     |     |     |     |

## 1.9 Expected Outcome

The purpose of this IoT-based crack detection and monitoring system is to provide continuous, real-time structural crack monitoring, which enables an early recognition of potential risks before they worsen. When crack sizes exceeding specified safety standards, the system seeks to promptly alert relevant personnel, facilitating immediate action and allowing timely preventive measures (Rahita et al., 2024). The system's cloud-based data storage will allow for seamless access to both historical and current data, making it a crucial resource to analyse the structural health of buildings and scheduling maintenance.

This will allow for constant, data-driven monitoring, which will improve overall safety while minimizing the risk of possible structural collapses (Rahita et al., 2024).

The proposed solution will additionally minimize the need for manual checks, which will save operational expenses as well as enhance the effectiveness of standard maintenance processes (Mathé, 2023). The technology is intended to encourage long-term building sustainability and decrease the possibility of expensive urgent repairs through enabling prompt actions, which will positively impact occupant safety as well as preventative maintenance budget (Babu et al., 2022).

## **CHAPTER 2: LITERATURE REVIEW**

### **2.1 Introduction**

Monitoring the structural health of buildings, bridges, and other structures is vital for maintaining their durability and safety. In order to avoid serious damages, it is crucial to promptly discover and track fractures, since they are among the most prevalent signs of building deterioration. Traditionally, crack detection relied on specialized equipment or manual checks, which often lacked flexibility, effectiveness, and real-time monitoring capabilities. The latest technologies that integrate wireless communication, sensors, and IoT platforms have developed to cope with these issues as a result of technological advancements.

This chapter lays the path for the advancement of an IoT-based solution by providing an extensive review of the existing literature on crack detection and monitoring systems. In this chapter, three existing systems for crack detection and monitoring are discussed, along with the technology and tools employed, as well as their processes, advantages, and drawbacks. Following this, the chapter presents a comparison of the proposed IoT-based solution with the existing systems.

Analysing the comparison's findings, the discussion section emphasizes the project's significance while highlighting the main distinctions between the proposed solution and the existing ones. The findings are summarized at the end of the chapter, in tandem with how the proposed IoT-based system helps improve structural health monitoring. This literature review establishes a foundation for the system's subsequent planning and execution in the next chapters.

### **2.2 Review on Similar Existing System**

Three similar existing systems for crack detection and monitoring are the vision-based system, the ultrasonic system, and the strain gauge system. Every system utilizes a

different method for detecting and monitoring structural cracks for the purpose of accommodating various demands and applications.

### **2.2.1 Computer Vision-Based Crack Detection Methods**

The vision-based system analyses photos of a building to locate cracks utilizing computerized imaging technologies. Large-scale infrastructure like bridges and dams can gain an advantage from this technology's excellent precision, which can recognize even the most minor weaknesses (Sun & Zhang, 2020). The incorporation of Convolutional Neural Networks (CNNs) in many current vision-based systems optimizes their fracture-identifying abilities. Artificial intelligence models called CNNs are notably proficient at identifying patterns in photos, thus making it feasible to find cracks accurately and automatically without the need for human involvement (Sun & Zhang, 2020). The procedure of CNN deep learning-based crack detection and quantification are shown in Figure 2.1. In addition, for not reachable locations, the vision-based technologies will be helpful because they involve no direct interaction with the structure. However, the system is susceptible to surrounding aspects like sunlight and weather, and it needs a lot of computing resources and costly equipment (Khan et al., 2023). Furthermore, when incorporating complicated CNN algorithms, it often becomes essential to work with experienced workers in order to operate and upkeep the system proficiently (Sun & Zhang, 2020).

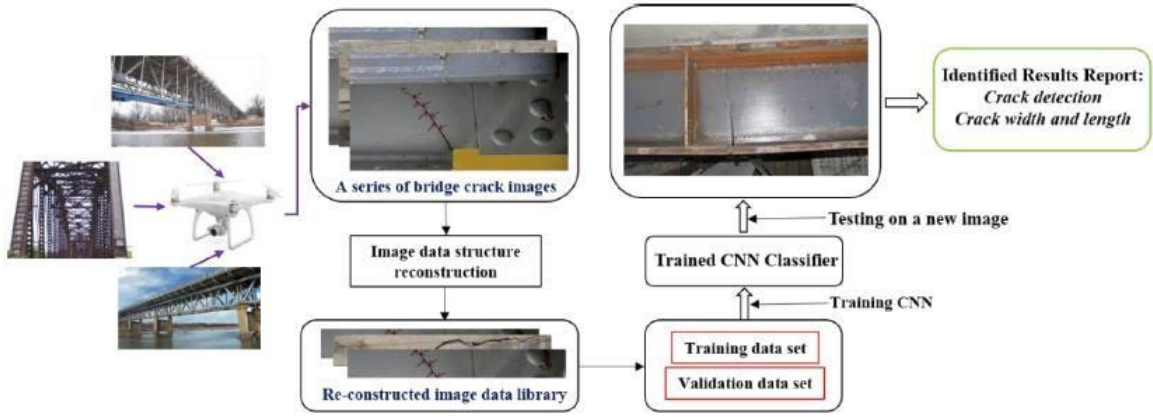


Figure 2.1: Procedure of CNN Deep Learning-based Crack Detection

(Sun & Zhang, 2020)

### 2.2.2 Ultrasonic System for Structural Health Monitoring

The ultrasonic system utilizes ultrasonic waves for identifying faults in the interiors of the buildings. According to Tanveer et al. (2024), this methodology functions most effectively in detecting underlying weaknesses, which makes it beneficial in manufacturing operations involving testing materials, pipeline inspections, and weld assessments. The technology can recognize changes in material qualities, which include cracks or voids, through projecting high-frequency sound waves into the building's structure and evaluating the reflected waves (Tanveer et al., 2024). It is affordable and convenient for focused localized assessments, facilitating efficient on-site assessments without requiring huge equipment. The system can be applied in numerous circumstances, such as remote locations where accessibility to more modern technology can become limited. For precise readings, the ultrasonic system needs to interact directly with the structure and is only suitable in small-scale areas, despite its benefits (Cantero-Chinchilla et al., 2021). Specialized transducers and coupling agents are needed for this, which could lengthen the setup period and complicate processes. Besides, it is not as efficient at recognizing cracks on the exterior, thus requiring trained personnel to correctly analyse the results (Babu et al., 2022). The system is not ideal for ongoing inspection of huge buildings because the process frequently demands regular checks and

lacks the capability to always offer continuous monitoring (Cantero-Chinchilla et al., 2021). On top of that, it is hard to install in difficult-to-reach or complex places, which therefore also limits its overall performance and applicability in large-scale monitoring. Figure 2.2 shows the framework of the ultrasonic wave-based structural health monitoring process.

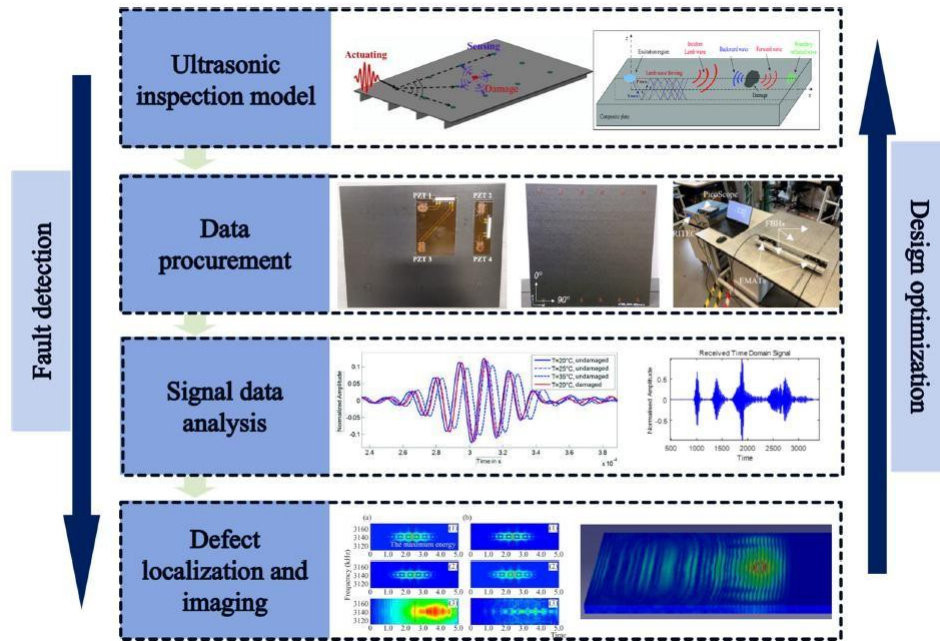


Figure 2.2: Framework of the Ultrasonic Wave-based Structural Health  
(Tanveer et al., 2024)

### 2.2.3 Structural Health Monitoring Technique for Crack Detection Using Strain Gauge System

The strain gauge system utilizes strain gauges to observe the emergence and growth of cracks as well as quantify structural tension. These strain gauges are most advantageous when analyzing localized stress, like in construction pillars or beams, since they have high sensitivity, making it possible for the identification of minor fractures (Li et al., 2019). Smaller-scale projects may employ the system because of its minimal maintenance costs and relative ease of setup (Liu et al., 2024). The dependence on a wired arrangement limits its adaptability and versatility because it can be challenging and not feasible to

install cables across huge building structures. It is therefore less suited for the monitoring of complex structures such as dams or bridges (Liu et al., 2024). The environmental factors, including changes in temperature or physical disruptions, may influence measurement precision, therefore, comprehensive calibration is necessary for maintaining reliability (Jung & Jang, 2024). Figure 2.3 shows the framework of the structural health monitoring using strain gauge sensors.

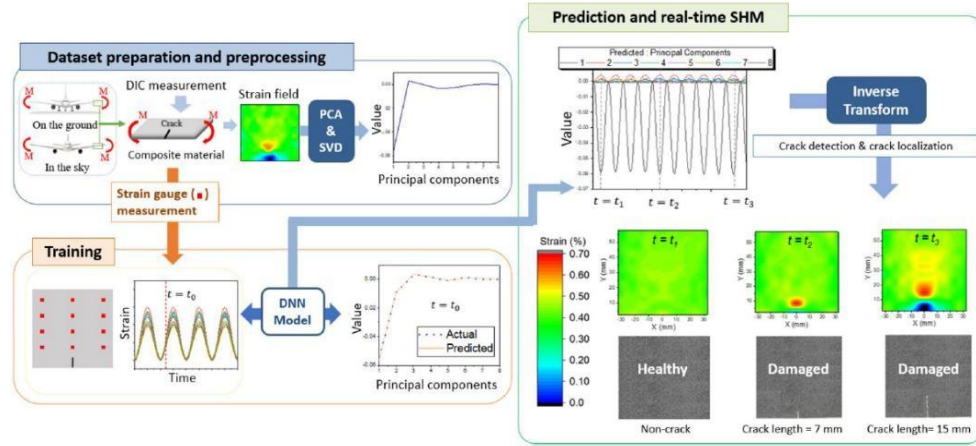


*Figure 2.3: Framework of the Structural Health Monitoring using Strain Gauge Sensors*  
(Yoon et al., 2022)

## 2.3 Tools and Technology Used

Several modern tools and technologies have been applied in the field of crack detection and monitoring systems that guarantee reliable and efficient outcomes. Convolutional Neural Networks (CNNs) in computer vision, strain gauges, and ultrasonic testing tools are some of the most used.

First and foremost, strain gauges are often utilized in structural health monitoring since they have the capacity for identifying subtle structural cracks. These sensors operate through observing shifts in an object's electrical resistance whenever it encounters tension. The strain gauge system is suitable for operations like analysing the structural strength of building columns or beams since it is specifically useful for targeted stress monitoring (Li et al., 2019). These tools provide a reasonable choice for specialized

checks and are rather easy to establish and manage. Nevertheless, the system's dependability on a wired configuration restricts its flexibility as stated by Liu et al. (2024), and it performs better at spotting minor flaws compared to those more expansive or complicated structures.

Subsequently, the ultrasonic testing tool is another common technology that utilizes high-frequency sound waves to locate internal cracks or faults in objects. This system works best on assessing any underlying faults that might not be noticeable from the outside. Ultrasonic devices employ transducers to generate sound waves, which are then returned and reviewed to identify issues like gaps or cracks (Tanveer et al., 2024). The effectiveness of ultrasonic testing to discover internal problems without harming the structure is its primary advantage. As mentioned by Cantero-Chinchilla et al. (2021), this method is limited to particular locations due to the requirement for close contact with the item being evaluated, and according to Tanveer et al. (2024), this system is not recommended for ongoing or continuous monitoring.

Apart from that, computer vision incorporated with convolutional neural networks (CNNs) is gaining prominence as a tool in current crack detection systems, specifically in monitoring huge infrastructures. In these technologies, CNNs interpret the pictures of the buildings that have been captured by high-resolution cameras. CNNs are an instance of artificially intelligent algorithms that are extremely proficient at recognizing patterns in pictures, which makes them appropriate for automated crack detection. A significant amount of graphical information may be examined by these algorithms, which can discover every minor defect without requiring human involvement (Sun & Zhang, 2020). CNN integration optimizes fracture identification effectiveness and precision, specifically for hard-to-reach structures. Despite these benefits, the technology is computationally demanding as well as vulnerable to environmental factors, such as weather and lighting, which might affect the clarity of images (Khan et al., 2023).

## 2.4 Comparison between Existing System

Table 2.1: Reviewed System Comparison

| System Name            | Computer Vision-Based Crack Detection Methods                                                                                                                                                                                                            | Ultrasonic System for Structural Health Monitoring                                                                                                                                                                                                         | Structural Health Monitoring Technique for Crack Detection Using Strain Gauge System                                                                                                                                                                                                | Proposed IoT-Based Crack Detection and Monitoring System                                                                                                                                                                                                         |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Tool / Technology used | Image Processing, Convolutional Neural Networks (CNNs)                                                                                                                                                                                                   | Ultrasonic Waves                                                                                                                                                                                                                                           | Strain Gauges                                                                                                                                                                                                                                                                       | IoT Devices, Cloud Computing                                                                                                                                                                                                                                     |
| Advantages             | <ul style="list-style-type: none"> <li>• Excellent precision in identifying extremely minor flaws.</li> <li>• Ideal for major infrastructure projects like dams and bridges.</li> <li>• Does not need direct contact and human involvement.</li> </ul>   | <ul style="list-style-type: none"> <li>• Effective for recognizing issues underneath the surface.</li> <li>• Affordable and convenient for small-scale inspections.</li> <li>• It is ideal for remote places.</li> </ul>                                   | <ul style="list-style-type: none"> <li>• Highly sensitive to even minimal fractures.</li> <li>• Low maintenance expenses and easy setup.</li> <li>• The most suitable option for monitoring localized stress.</li> </ul>                                                            | <ul style="list-style-type: none"> <li>• Continuous structural health monitoring.</li> <li>• Scalable and wireless.</li> <li>• Cloud-based analysis and data storage.</li> <li>• It eliminates the need for manual checks.</li> <li>• Cost-effective.</li> </ul> |
| Disadvantages          | <ul style="list-style-type: none"> <li>• Costly equipment and substantial computational capability are needed.</li> <li>• Environmentally sensitive which includes the weather and lighting.</li> <li>• Operation requires trained employees.</li> </ul> | <ul style="list-style-type: none"> <li>• Requires direct interaction and is limited to small-scale areas.</li> <li>• Ineffective in identifying surface cracks.</li> <li>• No real-time monitoring.</li> <li>• Requires workers with expertise.</li> </ul> | <ul style="list-style-type: none"> <li>• Scalability has limitations by its dependence on a wired setup.</li> <li>• Not ideal for huge building structures.</li> <li>• Precision is influenced by environmental factors.</li> <li>• Limited to certain monitoring spots.</li> </ul> | <ul style="list-style-type: none"> <li>• Connectivity difficulties may happen in remote locations.</li> <li>• Dependable energy sources are crucial for continuous operation.</li> <li>• Initial set up may be slightly challenging.</li> </ul>                  |

## 2.5 Discussion

Based on Table 2.1, the proposed IoT-based crack detection and monitoring system provides several key benefits in comparison with existing technologies. One of its primary advantages is that it is affordable. The system delivers an alternative that is achievable without the costly expenses related to vision-based or ultrasonic systems by employing cost-effective components. The system's affordability assures that either large-scale or small-scale operators can benefit from advanced monitoring features, enhancing its suitability to a variety of projects.

Furthermore, one major disadvantage of conventional technology has been solved by the system's continuous monitoring functionality. The proposed system continually checks the condition of the structure and sends out immediate alerts for any anomalies identified, in contrast to ultrasonic systems that are dependent on infrequent assessments. This characteristic enables it to recognize weakness in the structure at an early stage, which facilitates prompt maintenance and substantially minimizes the risk of severe damage, particularly in vital infrastructure like tall buildings.

Moreover, the system's wireless design and scalability enable it to be completely adapted for a wide range of purposes. The logistical difficulties resulting from wired configurations are mitigated by their wireless capabilities, which facilitate effortless implementation across several buildings or big projects. The system is an adaptable tool for monitoring structural health simply due to its flexibility, which implies that it may be utilized in urban locations, remote areas, as well as complicated structures with similar reliability.

Overall, the proposed IoT-based system presents a substantial milestone in structural health monitoring by supplying an expandable, real-time, and cost-effective solution that overcomes the primary weaknesses of the existing technology.

## **2.6 Summary**

This chapter covers an extensive review of the present crack detection and monitoring technologies, which includes strain gauge systems, computer vision-based methods, and ultrasonic systems. Each of the systems had distinctive positive and negative aspects, addressing the demands of certain areas and purposes. The review emphasized the primary constraints that prevent their broad application, ranging from excessive costs, a lack of continuous monitoring, and scalability challenges.

The proposed IoT-based crack detection and monitoring system solves these drawbacks by providing an affordable, real-time, and adaptable solution. The system incorporates effectiveness, convenience, and comprehensive monitoring features through employing wireless technology, cost-effective components, and integrated cloud services. These benefits make the system an innovative tool for maintaining building lifespan and safety, especially in settings that have distinct limitations on resources. The information acquired from this chapter establishes a solid basis for the planning and implementation of the proposed system in the following chapters, indicating its ability to completely transform structural health monitoring protocols.

## **CHAPTER 3: REQUIREMENT ANALYSIS AND DESIGN**

### **3.1 Introduction**

The core objective of this project is to produce an IoT-based crack detection and monitoring system that employs wireless sensors for detecting cracks in constructions and buildings. The stability and durability of buildings can be guaranteed by this advancement in technology, facilitating the utilization of the Internet of Things (IoT) to provide real-time monitoring along with early structural detection of weaknesses (Abdul Razak et al., 2022). Through the system's interconnection with cloud-based services, engineers, building managers, and other pertinent officials can remain in close monitoring of the structural condition of the building and promptly take measures before serious damages occur (Golding et al., 2022). The purpose is to develop an automated system that is expandable, dependable, and cost-effective. It can be implemented at any phase in a building's lifespan, generating significant early warnings to mitigate risks associated with structural collapses.

### **3.2 Methodology**

This project's methodology relies on the Agile method and these values, which include the ability to adapt, continual improvement, and continuous review. This methodology is most appropriate for projects that integrate both hardware and software elements due to its facilitating constant enhancements and adjustments in accordance with feedback from users and test outcomes (Simpson, 2024). For guaranteeing methodical progress along with efficient performance, the project has been broken down into six crucial phases.

Determining a specific implementation time frame, detailing the hardware and software specifications, and specifying the project's scope are the key goals of the initial phase, plan. To guarantee accessibility and functionality, the system architecture is established throughout the design phase, which additionally includes creating the user

interface, devising data algorithms, and deciding where to install sensors. The sensors, along with other hardware components, are incorporated with the software during the development phase, and connectivity and information-gathering platforms are set up. The system needs to undergo rigorous testing within the test phase to make certain that it is capable of accurately recognizing cracks and fulfilling the needs of real-time monitoring. Upon completing testing, the system will be deployed in a controlled setting during the deploy phase so that its operation and overall efficiency can be observed. Subsequently, the system is reviewed during the review phase to analyse its reliability, performance, and fundamental compliance with the project's objectives. The data obtained through this phase is employed to propose modifications and amendments, indicating that the system satisfies the requirements of its end users.

### **3.3 Requirement Elicitation**

According to the evaluation of questionnaire answers along with previous literature reviews, three major user requirements have been determined that correlate the project with its primary goals. These consist of producing a real-time IoT crack detection system that constantly evaluates the structural condition of a building and immediately notifies users for preventive measures upkeep, verifying that the technology is accessible, adaptable, and easy to implement allowing for broad deployment across numerous building types, and incorporating capabilities like timely notifications, remote accessibility, and historical data access that overcome the weaknesses of conventional crack detection approaches.

Sections A, B, and C formed the questionnaire that was conducted for this elicitation. To acquire further information about the target group's variety in terms of how they correspond to the system's expected usage, Section A obtained demographic details from the respondents, which include their age, gender, and occupation. Section B assessed the respondents' comprehension regarding current crack detection technologies, what

they experienced with them, and the issues that they endured. The constraints of the methods currently in use were carefully outlined in this section, which further addressed the importance of innovative solutions. Meanwhile, section C analysed the respondents' preferences for specific qualities and evaluated possible problems involving cost, functionality, and performance with the aim of understanding their demands for an IoT-based crack detection system.

Based on the feedback, affordability, accessibility, and continuous monitoring are crucial for user satisfaction. These outcomes influenced the system's operation and design to guarantee it adequately fulfils user requirements. The questionnaire's detailed outcomes are provided below.

### **Section B:** Familiarity with Crack Detection Systems

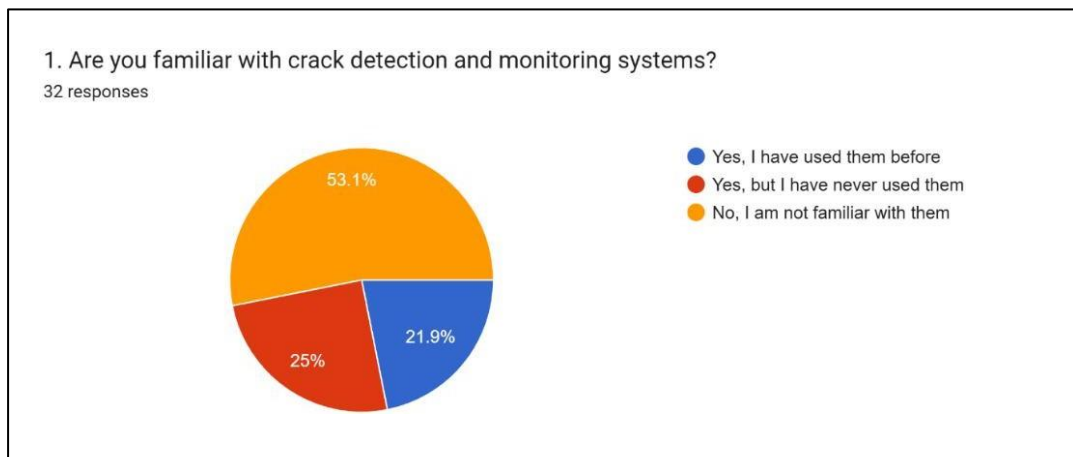


*Figure 3.1: The First Question for Section B*

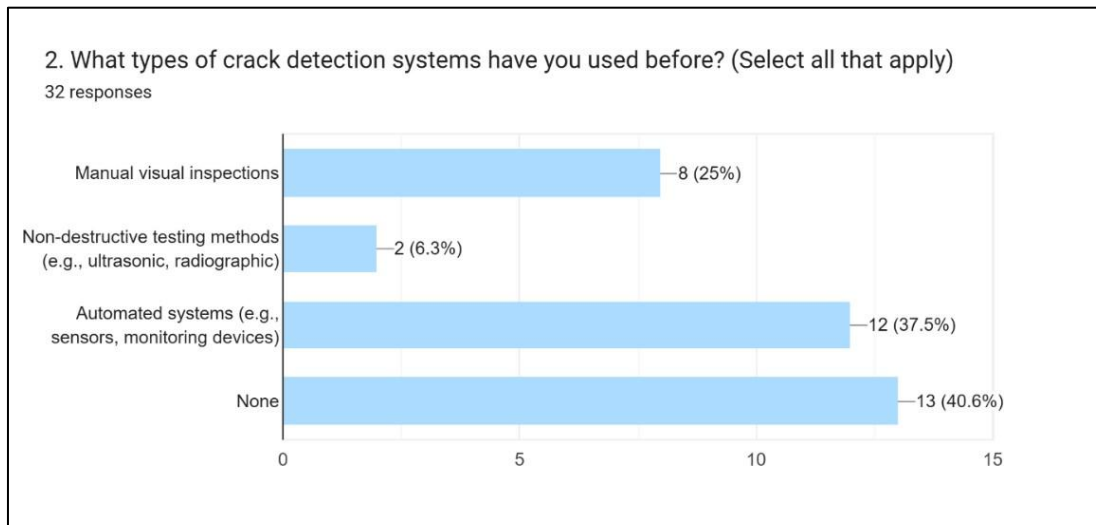


*Figure 3.2: The Second Question for Section B*

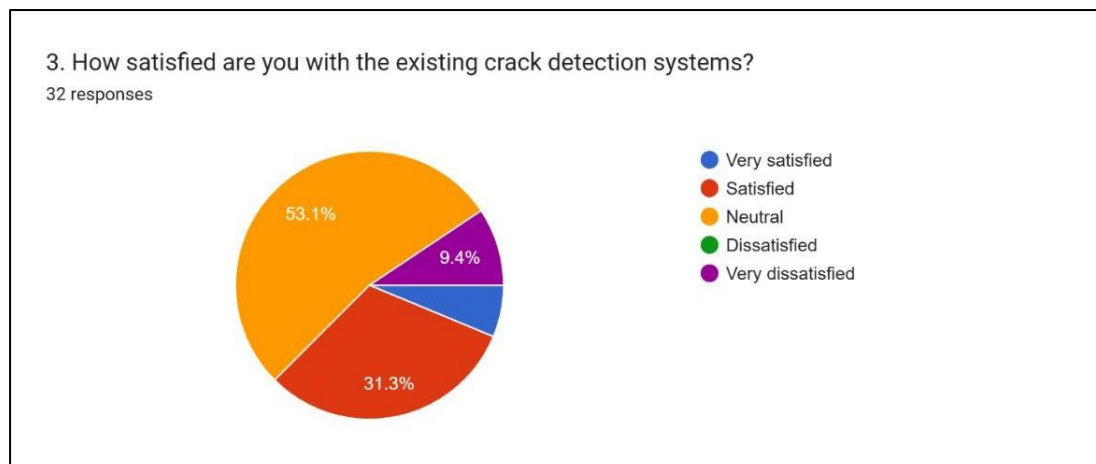


*Figure 3.3: The Third Question for Section B*

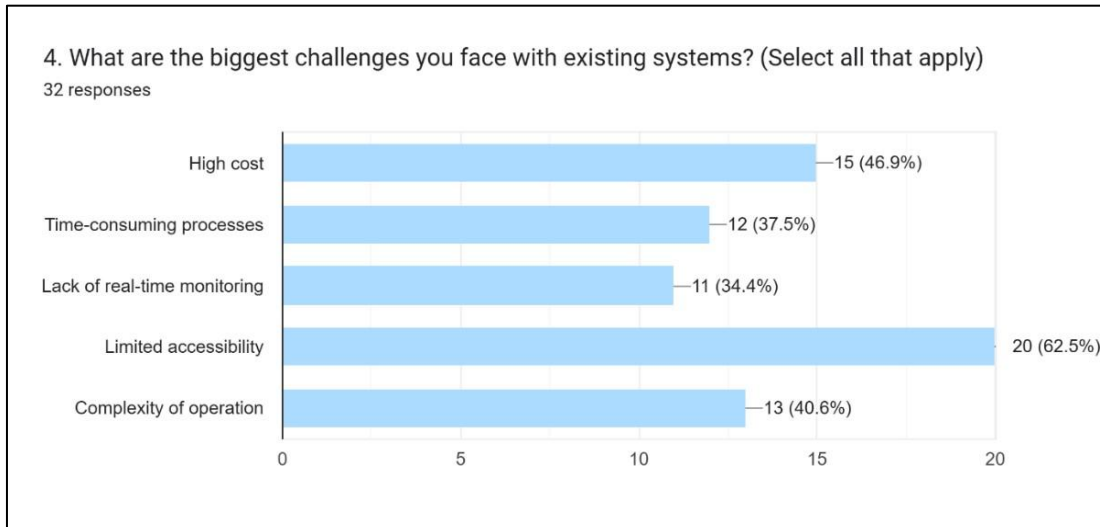


*Figure 3.4: The Fourth Question for Section B*

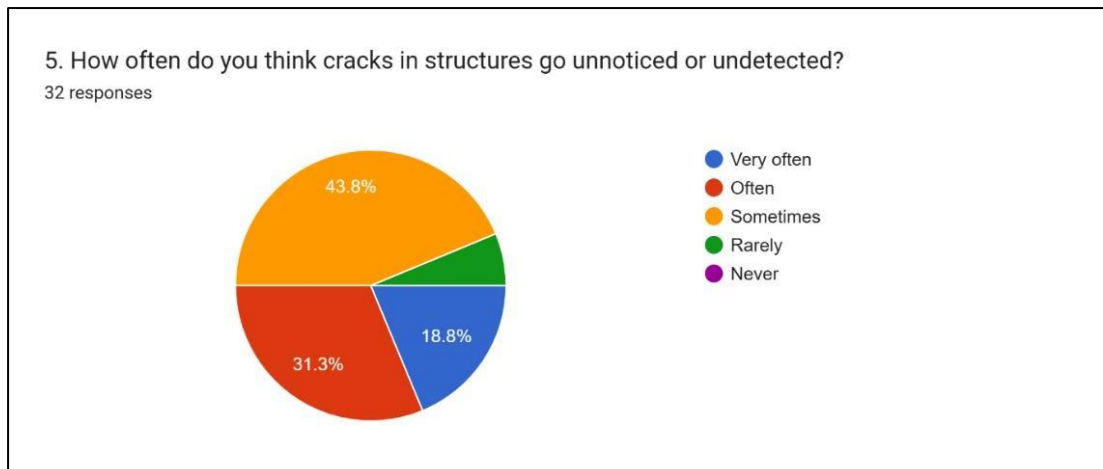


*Figure 3.5: The Fifth Question for Section B*

In accordance with the responses from Section B, particularly those related to respondents' familiarization with crack detection systems, more than half of the respondents (53.1%) weren't familiar with crack detection and monitoring systems, while just 21.9% of them were, as shown in Figure 3.1. Figure 3.2 shows that the automated systems were the most frequently employed (37.5%), then followed by manual visual inspections (25%), while 40.6% of respondents were unfamiliar with any sort of system before. As can be seen in Figure 3.3, a large percentage of respondents indicated neutrality (53.1%) or satisfaction (31.3%) with the present procedures, but none of them stated major

dissatisfaction. Figure 3.4 indicates that, most prominently, limitations with current methods were related to limited accessibility (62.5%), high cost (46.9%), and operational complexity (40.6%). According to Figure 3.5, it was believed that structural failures were overlooked almost regularly, with 43.8% of respondents claiming this happened "sometimes" and 31.3% suggesting it was occurring "often." Important discrepancies in knowledge, convenience, and satisfaction with existing systems have been identified by these outcomes, which underscores the necessity for innovative and comprehensible approaches.

### Section C: Proposed IoT-Based Crack Detection and Monitoring System

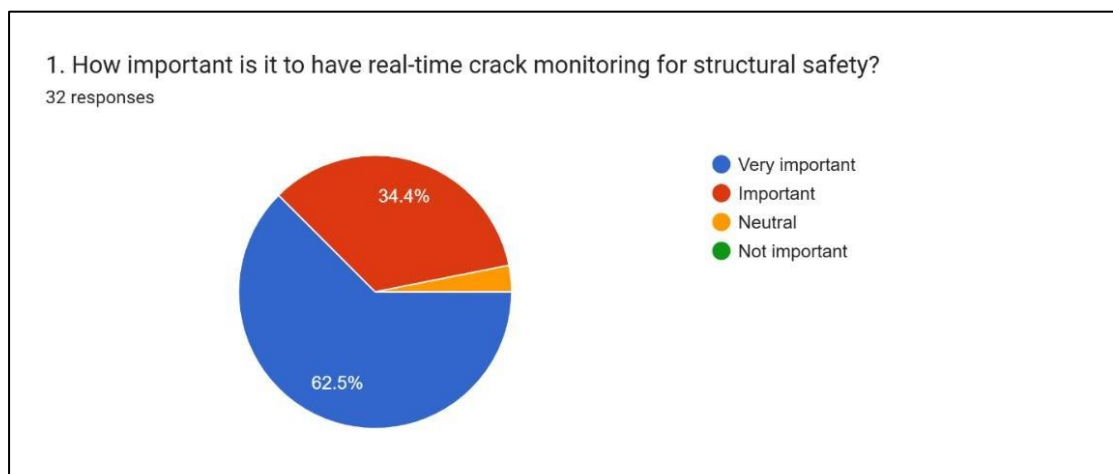


*Figure 3.6: The First Question for Section C*

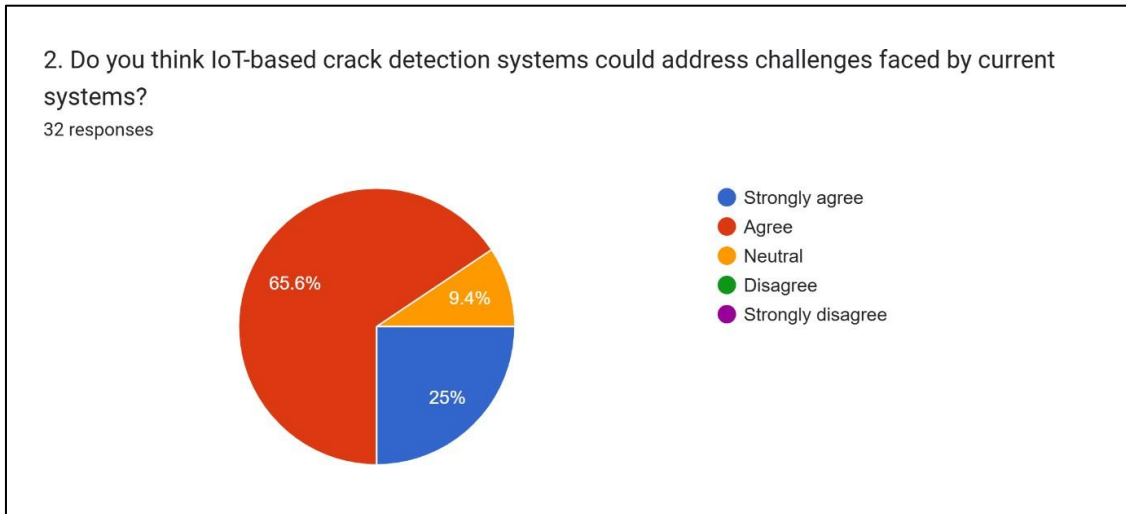


*Figure 3.7: The Second Question for Section C*

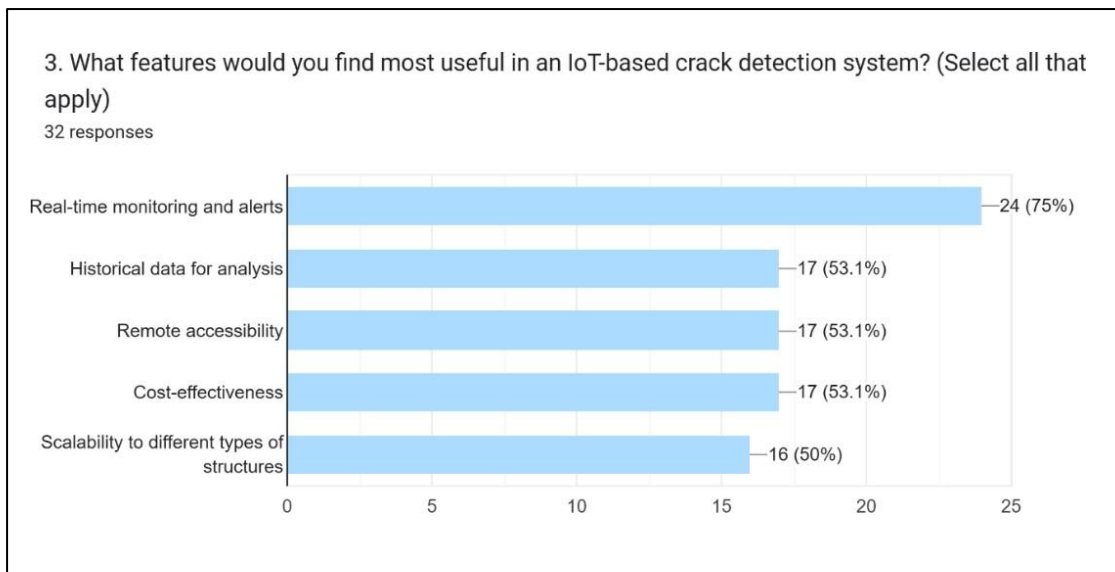


*Figure 3.8: The Third Question for Section C*

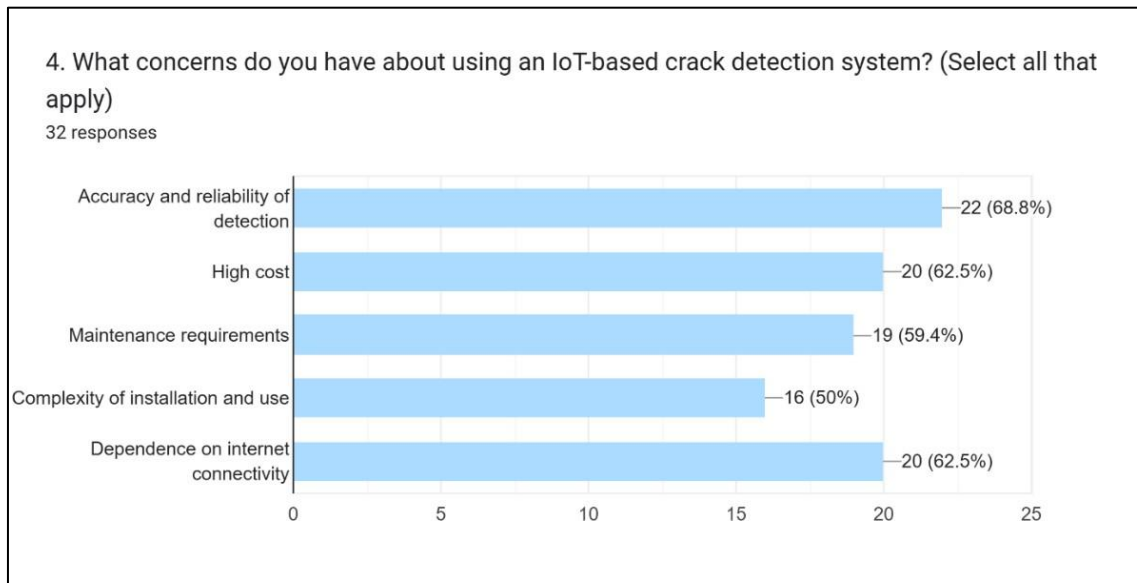


*Figure 3.9: The Fourth Question for Section C*

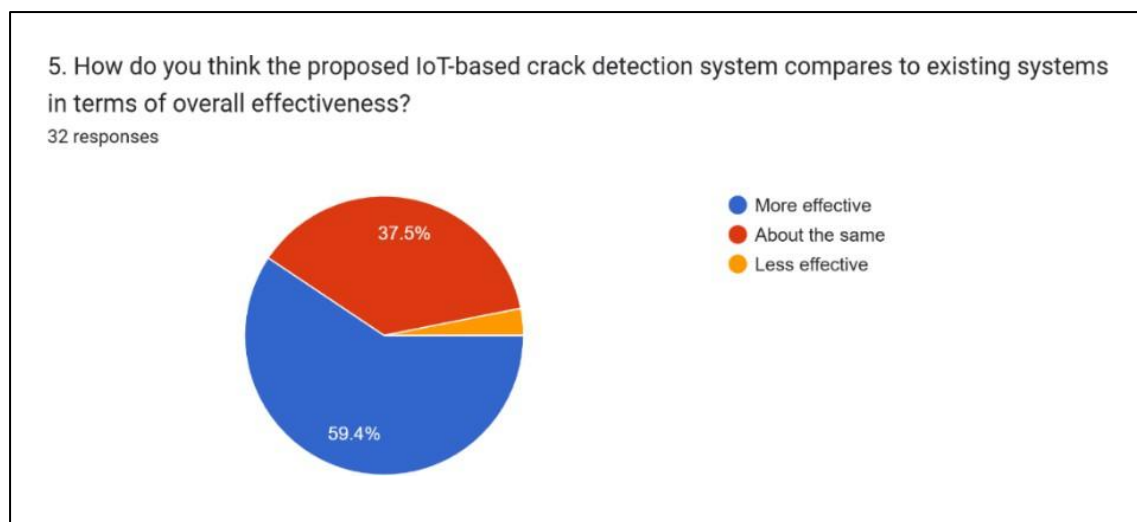


*Figure 3.10: The Fifth Question for Section C*

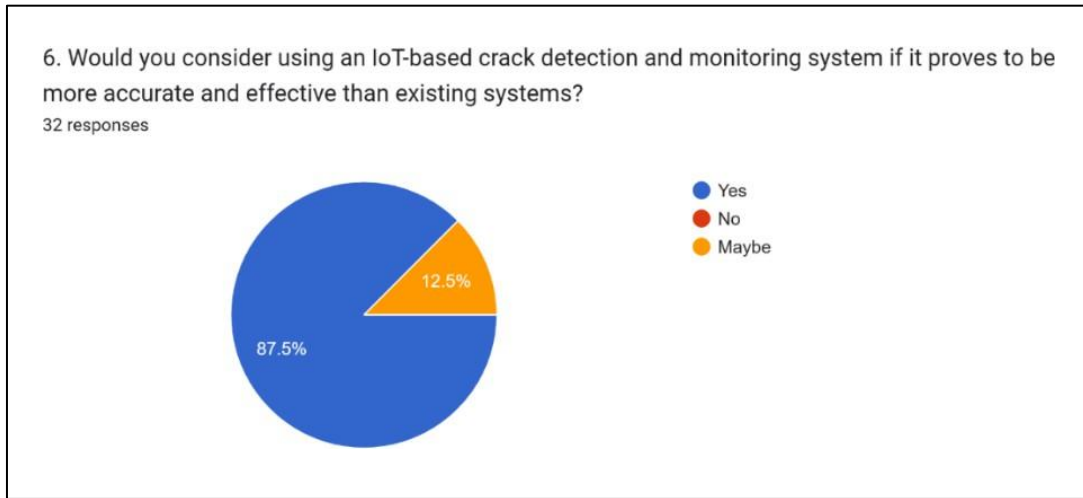


*Figure 3.11: The Sixth Question for Section C*

Responses to Section C's questionnaires on the proposed IoT-Based Crack Detection and Monitoring System clearly highlighted the significance of real-time monitoring for the safety of structures, with 34.4% classifying it as "important" and 62.5% as "very important," as shown in Figure 3.6. According to Figure 3.7, most respondents (65.6%) agreed, with 25% firmly agreeing that IoT-based technologies might resolve the issues that exist with present solutions. As can be seen in Figure 3.8, real-time monitoring and alerts (75%), access to historical data (53.1%), and remote accessibility (53.1%) were selected as the main capabilities demanded in such systems, implying a need for adaptable systems that effectively deliver practical insights. Figure 3.9 shows that the requirement for durable and reasonable options is made apparent by the concerns raised concerning accuracy as well as reliability (68.8%), high cost (62.5%), and reliance on the internet (62.5%). Figure 3.10 indicates that 59.4% of respondents ranked the proposed IoT-based system as "more effective" compared to conventional methods, whereas 87.5% of respondents responded that they were ready and interested to employ the system if it appeared to be advantageous, as shown in Figure 3.11. The results obtained indicate the significant need for advanced, dependable, and readily accessible IoT-based crack detection systems that fulfil user preferences and overcome current challenges.

### 3.4 Software and Hardware Requirements

The software and hardware requirements for the proposed IoT-based crack detection and monitoring system to be developed are being determined. The following Table 3.1 and Table 3.2 provide a concise overview of the software and hardware components to be employed in the development of the proposed system.

#### 3.4.1 Software Requirements

*Table 3.1: Software Requirements for Development*

| No. | Software        | Description                                                                                                                                                                                                                                              |
|-----|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1   | Google Firebase | Google Firebase is utilized for handling and storing data in real time, as well as hosting the website's user interface. It will be employed to facilitate expandable and reliable crack monitoring data collection and retrieval.                       |
| 2   | ThingSpeak      | An IoT cloud-based system that operates for gathering, processing, and providing data collected by sensors. The software system acts as a central hub for analysing and displaying data from sensors with the intent to gather observations.             |
| 3   | Web Interface   | The web interface is intended for displaying data analysis and alerts leveraging common web technologies, which involves HTML, CSS, and JavaScript. The aim is to deliver users a simple platform for monitoring data in real time and receiving alerts. |
| 4   | ESP32 SDK       | The ESP32 sensor module is established and controlled via this programming platform. Its primary purpose is the capability to produce personalized programs for handling and improving sensor functionality.                                             |

### 3.4.2 Hardware Requirements

Table 3.2: Hardware Requirements for Development

| No. | Hardware                 | Description                                                                                                                                                                                                        |
|-----|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1   | ESP32                    | The microcontroller is assigned to analysing data, integrating with sensors, and passing it to the cloud for storage. Their role is to function as the main processing unit for communication and data processing. |
| 2   | Ultrasonic Sensor        | Ultrasonic sensors are employed to identify structural issues. The intent is for detection of structural cracks through providing precise measurement data.                                                        |
| 3   | Wi-Fi / Cellular Network | Wi-Fi or cellular networks allow a cloud-based system and sensors to connect and interact with each other. The purpose is to deliver data transmission in real time for alerts and monitoring.                     |
| 4   | Power Supply             | The power supply gives energy to the ESP32 and any attached sensors. The aim is to make certain that the IoT system's components function consistently.                                                            |

## 3.5 Functional and Non-functional Requirements

### 3.5.1 Functional Requirements

The functional requirements describe the crucial characteristics of the system and have been determined from the project's primary objectives:

1. The ESP32 microcontroller is required to be connected with ultrasonic sensors to enable the system to detect and monitor structural faults in real time.
2. For clear interpretation, the system needs to store crack-related information on a cloud system and then present it via a simple web interface.
3. Every time severe crack behaviour appears, the system needs to immediately inform or notify users in advance so they may initiate the necessary measures.

4. The web interface must allow users to access the system remotely so that they can continue to keep close tabs on the overall condition of the structure without necessarily being onsite.

### **3.5.2 Non-functional Requirements**

The non-functional requirements emphasize guaranteeing that the system satisfies the intended performance criteria while remaining within the scope of the project:

1. The system's interface and design would be intuitive for users with little to no technical expertise to navigate smoothly.
2. The total cost of the project needs to be maintained reasonably through the adoption of affordable system components and tools.
3. To deliver lasting reliability, the system has to perform effectively and produce precise detection of crack results.
4. Regular web browsers are supposed to be available to access the system without needing other applications to be downloaded.
5. The system is intended to facilitate initial small-scale deployment, with the expectation of further development for accommodating bigger buildings or additional sensors.

### **3.6 System Design**

The IoT-based crack detection and monitoring system's structure and processes are portrayed comprehensively in the system design. It depicts how the sensors, microcontroller, cloud platform, and user interface connect with each other to reach the objectives of the project. To concisely convey the flow of data and interactions between systems as well as ensure compliance with all the requirements, the context diagram, Level 0 Data Flow Diagram (DFD), and Level 1 DFD are utilized.

### 3.6.1 Context Diagram

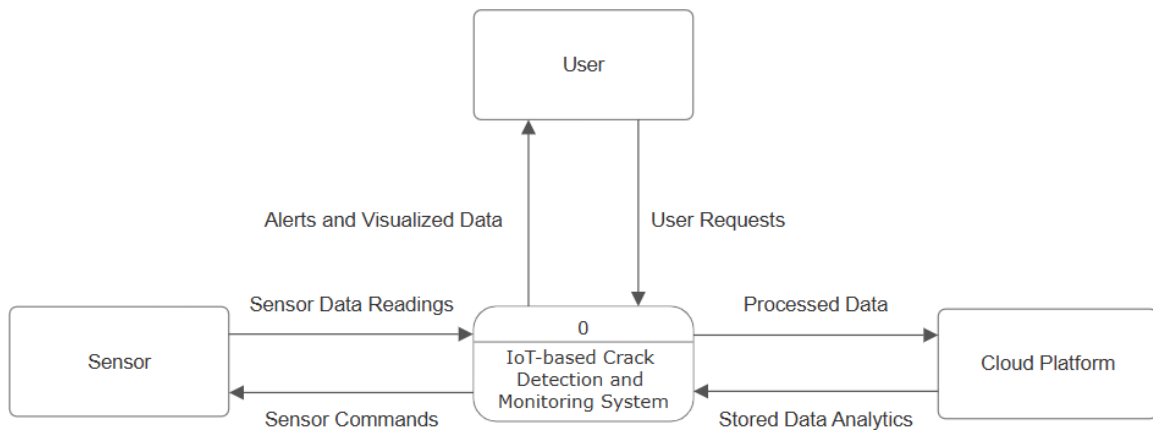


*Figure 3.12:* Context Diagram of IoT-based Crack Detection and Monitoring System

Figure 3.12 indicates the context diagram for IoT-based crack detection and monitoring system. Based on the context diagram, the sensor, cloud platform, and user are the main entities with which the system communicates. The system collects real-time sensor data readings from the sensor and then employs the results to transmit back instructions that regulate sensor processes. Through the transfer of data, the system uploads completed data for storage and availability as well as obtains stored data analytics from the cloud platform. Through requests made by users for configurations or data, the system engages with the user and delivers alerts and visualized data to notify the user of the condition of the structure state.

### 3.6.2 Level 0 Diagram

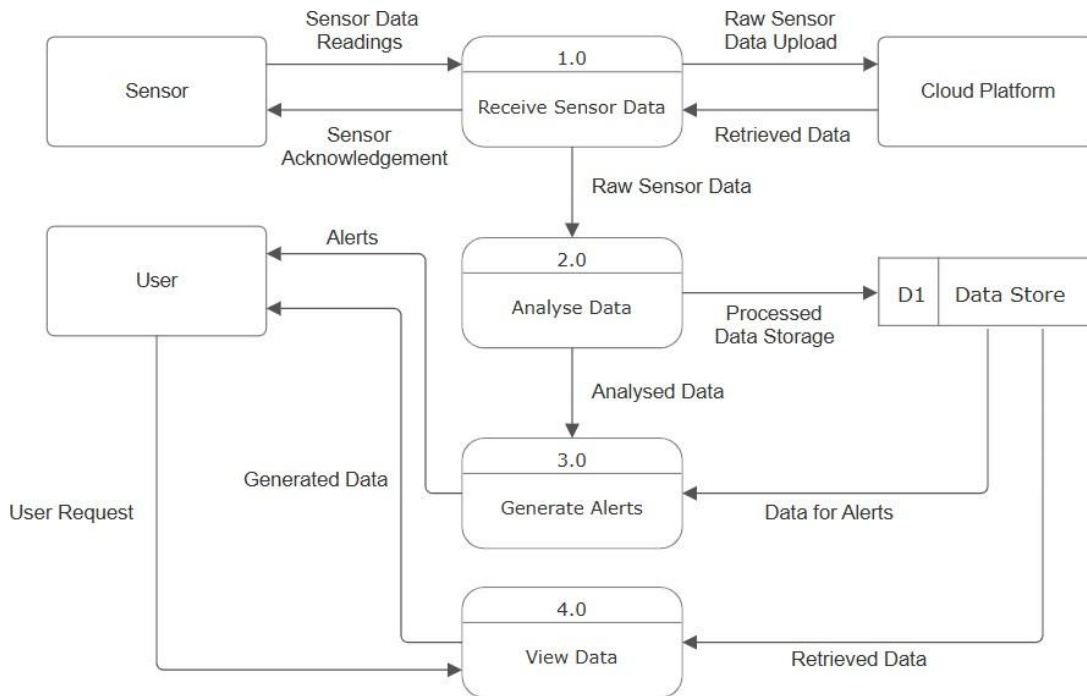


*Figure 3.13: Level 0 of IoT-based Crack Detection and Monitoring System*

The Level 0 diagram illustrates the internal activities of the system as well as how they cooperate with the data store and external entities in Figure 3.13:

#### **Process 1.0 – Receive Sensor Data**

Raw sensor data is transmitted to the cloud platform by the system following its receipt of sensor information obtained from the sensor and recognizes arrival with a sensor acknowledgement. In addition, raw sensor data can be acquired by the system from the cloud platform.

#### **Process 2.0 – Analyse Data**

After analysing the raw sensor data, the system keeps the processed data in the data storage (D1) for future usage.

### Process 3.0 – Generate Alerts

The system receives alert details from the data store, evaluates them, and subsequently employs the outcomes for generating alerts. The user receives these alerts and is reminded to perform an appropriate action.

### Process 4.0 – View Data

To view data, the user may send out a user request. To aid users with monitoring and decision-making processes. The system produces visualized data analyses by acquiring data for insights from the data store.

## 3.6.3 Level 1 Diagram

### 3.6.3.1 Process Receive Sensor Data

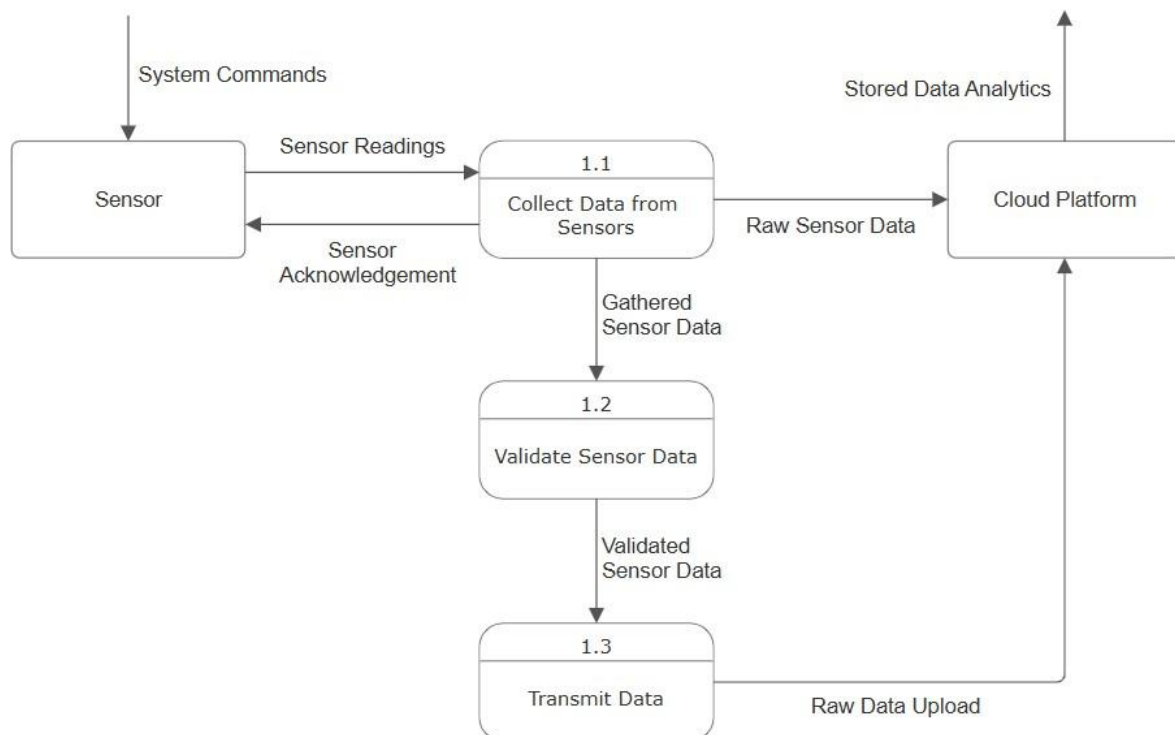


Figure 3.14: Level 1 for Process Receive Sensor Data

To start capturing data, the system delivers system commands towards the sensor as shown in Figure 3.14. In response, the sensor provides the system sensor readings. Following the raw data is obtained in Subprocess 1.1 (Collect Data from Sensors), it is passed on to Subprocess 1.2 (Validate Sensor Data) for the confirmation, creating Gathered Sensor Data. After having been verified, the data proceeds on to Subprocess 1.3 (Transmit Data), and there it gets transmitted as a raw data upload to the cloud platform and encapsulated as validated sensor data. Concurrently, the system acknowledges data reception through returning a sensor acknowledgment back to the sensor. Analytics for the data kept are made accessible through the system by the cloud platform.

### 3.6.3.2. Process Analyse Data

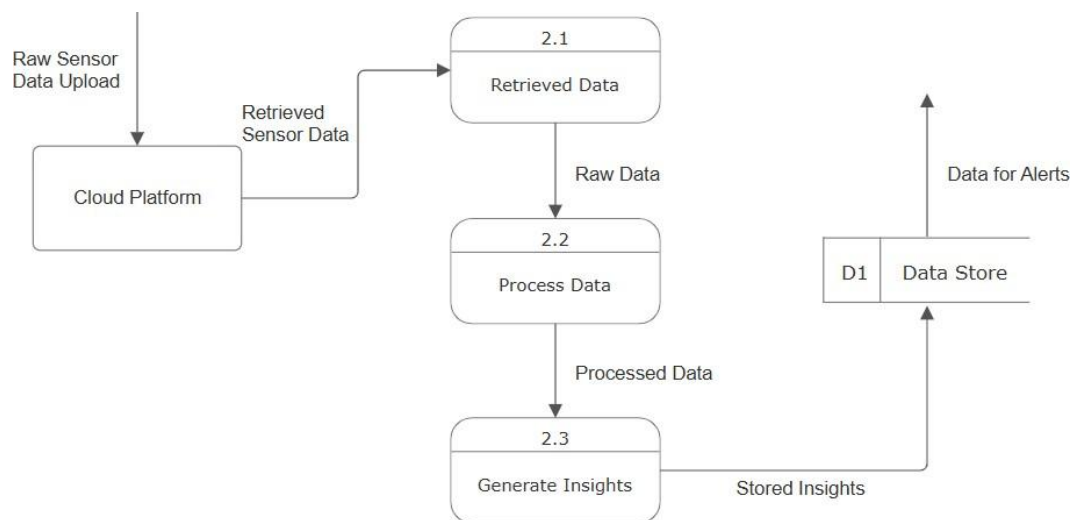


*Figure 3.15: Level 1 for Process Analyse Data*

Based on Figure 3.15, the system transforms the raw sensor data upload into retrieved sensor data after obtaining it from the cloud platform in subprocess 2.1 (Retrieve Data). Soon after being enhanced and translated into processed data, this data is delivered to Subprocess 2.2 (Process Data). The Data Store (D1) holds the relevant stored insights that are generated by additional analysis of the processed data in Subprocess 2.3

(Generate Insights). Following that, these insights will become accessible as data for alerts that help in further operations involving alert generation.

### 3.6.3.3. Process Generate Alerts

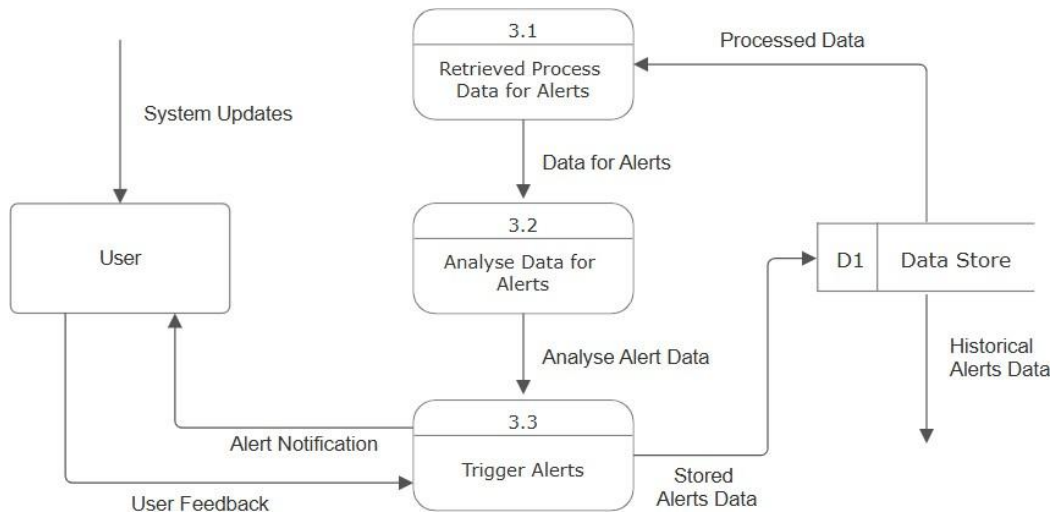


*Figure 3.16: Level 1 for Process Generate Alerts*

According to Figure 3.16, the system acquires processed data and historical alerts data from subprocess 3.1 (retrieve process data for alerts) in the data store (D1). Subprocess 3.2 (Analyze Data for Alerts) evaluates data that was recovered to discover any unusual patterns or major finds. Subprocess 3.3 (Trigger Alerts) follows by applying the results to inform the user of alerts. Also, the system sends the alerts to the data store as stored alerts data. The user may provide the system user feedback to enhance the alerting processes.

### 3.6.3.4. Process View Data

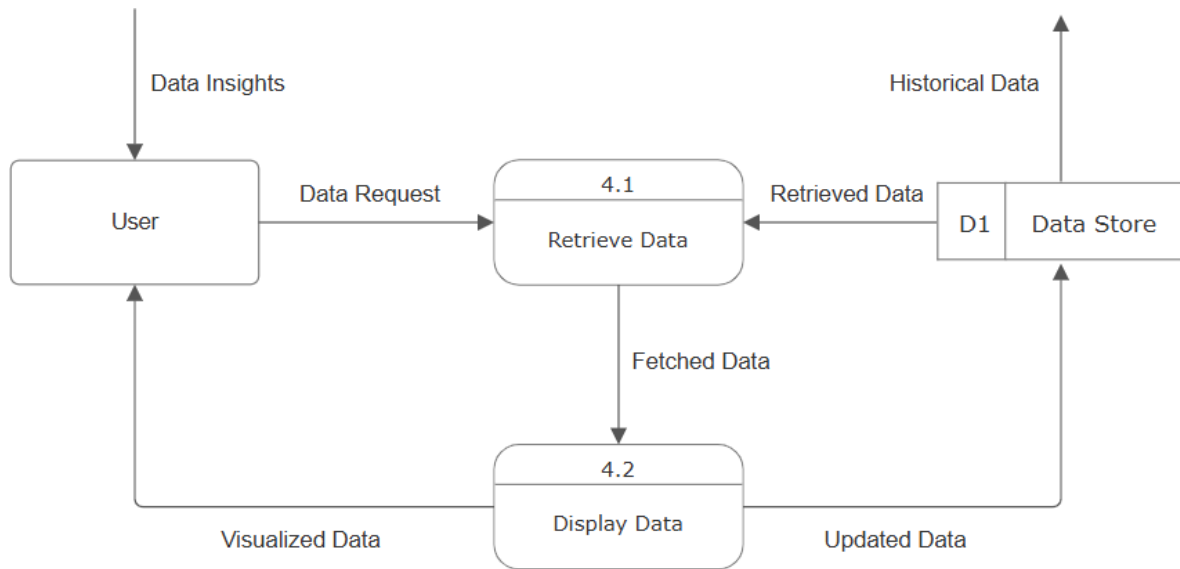


*Figure 3.17: Level 1 for Process View Data*

Figure 3.17 shows that the entire process is initiated by the user via Subprocess 4.1 (Retrieve Data), which transmits a Data Request to the system. From the Data Store (D1), the system collects historical data or any additional relevant retrieved data. After being brought, the data is passed on to Subprocess 4.2 (Display Data), and there it will be shown to the user as Visualized Data. Moreover, the system transfers updated data to the data store, making sure that all observations that are displayed remain available for further analysis.

### 3.6.4 Database Design

Figure 3.18 below depicts the IoT-based Crack Detection and Monitoring System's database design, which is designed to properly handle data and maintain uninterrupted interaction across users, the system, and sensors. Sensor, which gathers sensor information and readings; raw data, and processed data, which consist of unprocessed and analysed data. Moreover, alerts, which deal with alerts and user feedback; user, which records user information and communications; and system logs, which hold past data for monitoring and analysis, comprise its primary entities. The connections among these entities present effective data handling, retrieval, and storage, along with real-time crack detection and monitoring features.

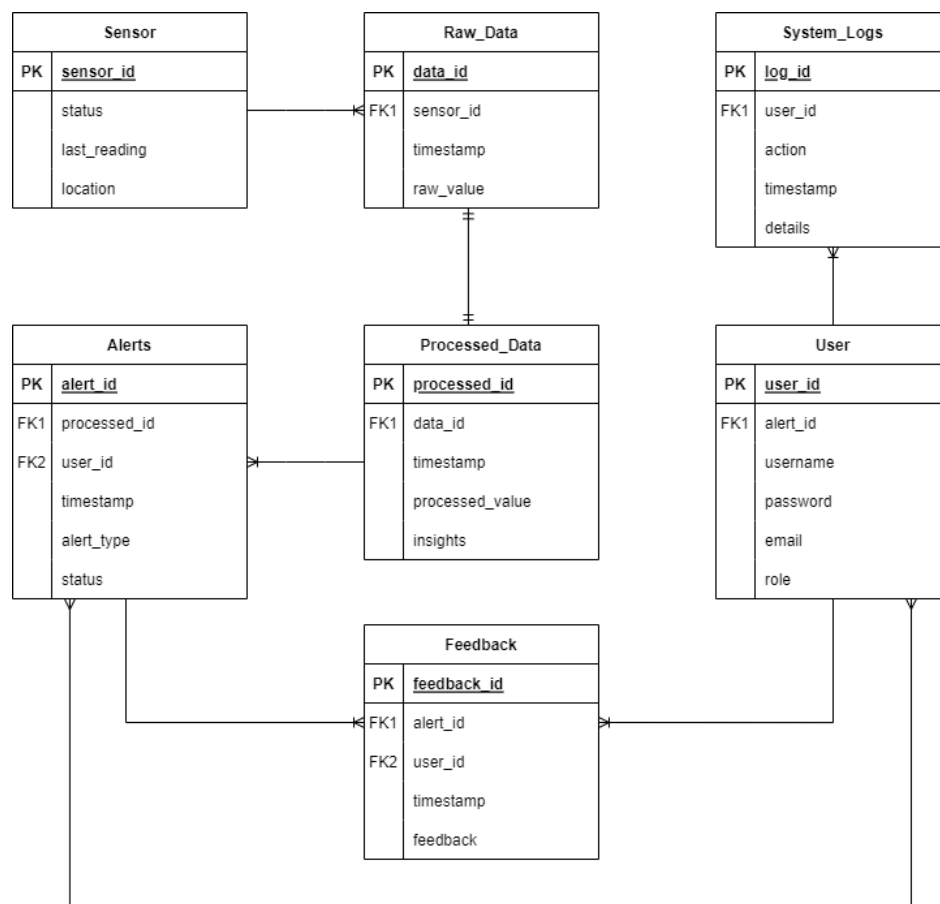


Figure 3.18: Entity Relationship Diagram (ERD)

### 3.6.5 System Architecture

An IoT-based crack detection and monitoring system is built based on the IoT architecture model, which comprises three key layers, including the hardware layer, the cloud layer, and the application layer. The first layer, also known as the hardware layer, consists of crack detection sensors, particularly ultrasonic sensors, that monitor conditions of structures involving cracks. The ESP32 microcontroller is linked to the sensor via the correct input and output ports. In accordance with the hardware needs, a 3.3V power source supplies the sensor and microcontroller. The ESP32 operates as the primary control module for transmission via wireless network, data handling, and acquisition. Upon obtaining unprocessed sensor data and analysing it for basic identification, it transfers the data across Wi-Fi to the cloud-based system. Analog or digital pins of the ESP32 accept inputs from the sensors, and data that has been analysed is uploaded to the cloud.

Furthermore, in the cloud layer, cloud platform like ThingSpeak collect and analyse sensor data, conduct data analysis, and provide insights. Upon obtaining data from the ESP32 microcontroller, the cloud platform transmits the processed data or insights to the web interface so that end users can engage with it. The processed data, system logs, feedback from users, and past alerts are all kept in a database stored in the cloud. This database makes it feasible to keep and access information for the generation of reports, user alerts, and system activity logs.

The application layer entails a web interface operated by Google Firebase. Utilizing an intuitive dashboard, this interface enables users to observe the identification of cracks data, alerts, and system activity. It delivers insights and illustrations for the user by obtaining both historical and current information from the cloud database. Apart from that, the user interface includes functions includes login for users, a dashboard that

displays real-time updates on crack detection, an overview of data history and analysis area, as well as alternatives for user input and alert preferences.

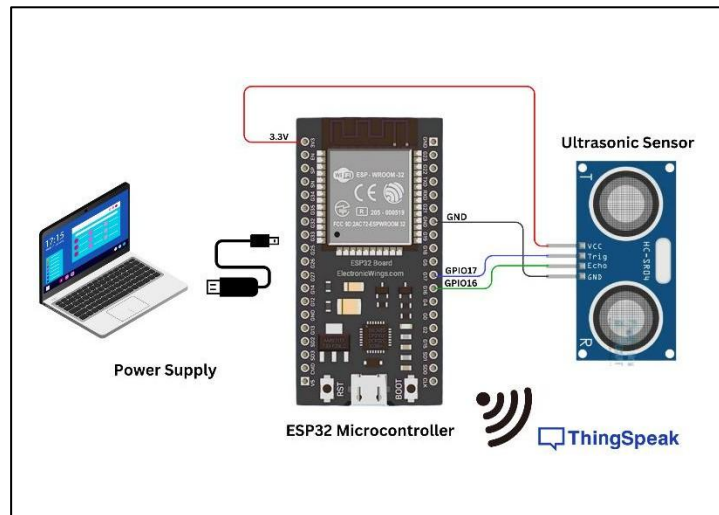


*Figure 3.19: Hardware and Software Connection of the Proposed System*

Figure 3.19 depicts the system's hardware and software connections as well as illustrating how the cloud platform, power supply, ESP32 microcontroller, and crack-detecting sensor interact. By connecting to the ESP32 microcontroller via the USB connector, the power source supplies it the voltage needed to perform its tasks effectively. The ESP32 acts as the central processing unit, receiving data gathered from the ultrasonic sensor and relaying it to ThingSpeak, a cloud platform, for data visualization and remote monitoring. The ESP32 has been linked to the ultrasonic sensor via specified GPIO pins: the ESP32's GPIO pins 16 and 17 are attached to the Trig and Echo pins, accordingly, while the VCC and GND pins are linked to power and ground. Through these connections, the ESP32 has the ability to estimate distance measurements by transmitting a trigger signal to the sensor while receiving the echo signal. Once the data is completely handled, the ESP32 delivers it via Wi-Fi to ThingSpeak, where it is kept and showcased for user observation. This architecture design facilitates real-time data record and storage by enabling smooth integration of cloud services and physical components.

### 3.6.6 Interface Design

Real-time data collected from sensors, including crack readings, alerts, and historical data, will be shown on the web interface. The convenience and accessibility will be put to the most consideration in the interface design, which guarantees that users without technical skills can comprehend the contents with ease.

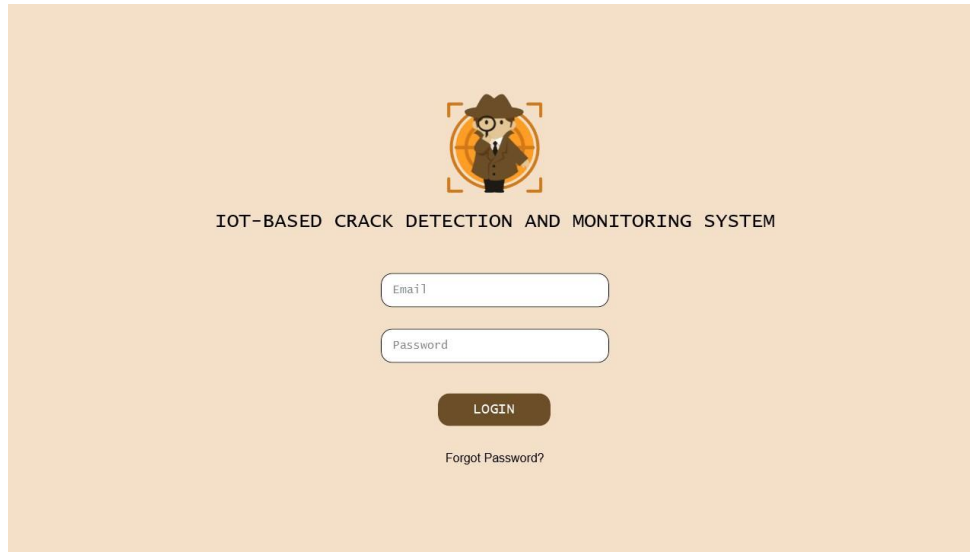


*Figure 3.20: User Authentication Page*

Figure 3.20 displays the user authentication page, where users may utilize the system via this page. The page possesses a basic, simplistic design and is centered. Following the system name and logo at the top are spaces for providing a password and email address. Upon completing these requirements, users may log into their account by pressing the "Login" button. They are entitled to a "Forgot Password?" button for recovering their passwords in case they forget their login details.

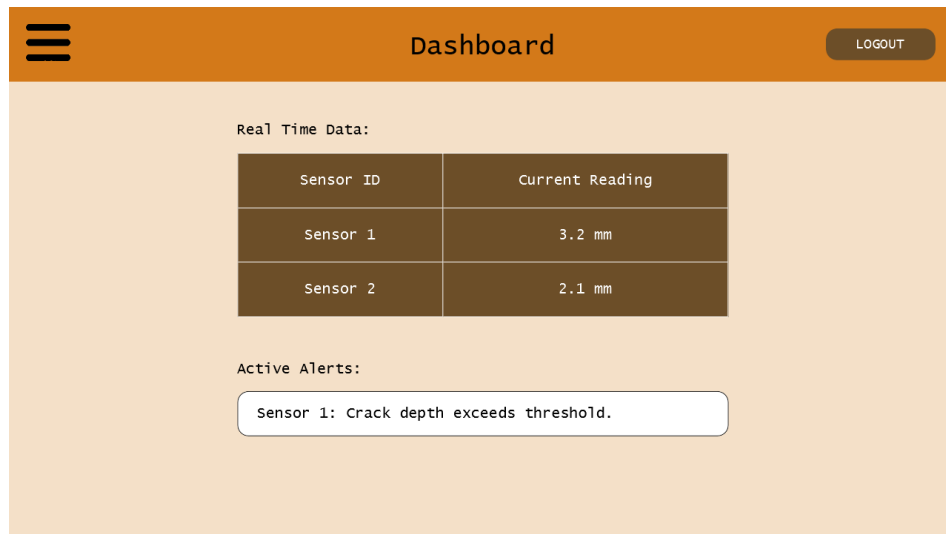
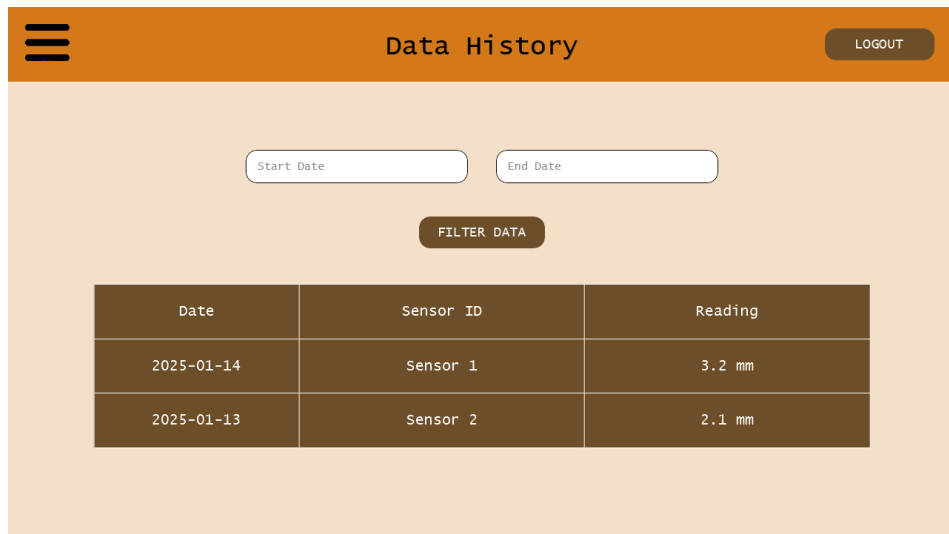


*Figure 3.21: Dashboard Page*

As shown in Figure 3.21, the Dashboard page's arrangement is straightforward, with a "Logout" button on the upper right for quick access by users and an apparent title at the top of the screen. Both the Real-Time Data and the Alerts part consist of the core section that can be seen under the header section. A basic table with columns for Sensor ID and Current Reading delivers the real-time data, providing the latest and most recent sensor readings, including crack depths. For maintaining users updated on vital situations in real time, the alerts section underneath displays any active alarms, for instance, when the sensor identifies a crack depth that surpasses the specified threshold.

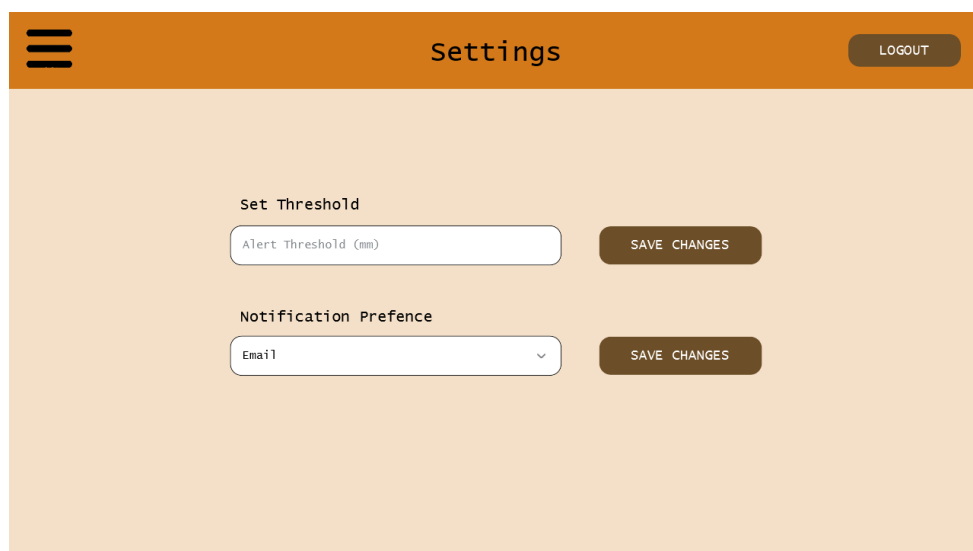


The Data History page features a top navigation bar with a hamburger menu icon, the title "Data History", and a "LOGOUT" button. Below the navigation bar, there are two input fields for "Start Date" and "End Date", followed by a "FILTER DATA" button. The main content area displays a table with three columns: "Date", "Sensor ID", and "Reading". The table contains two rows of data.

| Date       | Sensor ID | Reading |
|------------|-----------|---------|
| 2025-01-14 | Sensor 1  | 3.2 mm  |
| 2025-01-13 | Sensor 2  | 2.1 mm  |

*Figure 3.22: Data History Page*

Figure 3.22 is the Data History page, which enables users to review information obtained from sensors from previously collected data for a specific period. The date ranges filter at the top of the page facilitates users to narrow down data relating to specific time frames through entering a Start Date and an End Date. Under the selected filter, the page presents the results of the filter in an intuitive table with Date, Sensor ID, and Reading columns, generating an easy overview of historical sensor readings.



The Settings page features a top navigation bar with a hamburger menu icon, the title "Settings", and a "LOGOUT" button. Below the navigation bar, there are two sections: "Set Threshold" and "Notification Preference". The "Set Threshold" section has an input field for "Alert Threshold (mm)" and a "SAVE CHANGES" button. The "Notification Preference" section has a dropdown menu for "Email" and a "SAVE CHANGES" button.

*Figure 3.23: Settings Page*

User personalization to specific monitoring system functions is made attainable via the Settings page as shown in Figure 3.23. Users may indicate the crack depth threshold (in millimeters) at which they choose to be receiving alerts through the Alert Threshold input box on the page. The "Save Changes" option enables users to retain their decisions after updating the threshold. Furthermore, users may decide which method they would rather receive alerts via email or SMS via the Notification Preference dropdown on the screen. This page optimizes the overall user experience and offers users more control regarding alerts by enabling them to personalize the system according to their particular needs.

### **3.7 Implementation**

The execution of the IoT-based crack detection and monitoring system, which will be further developed in Final Year Project 2, starts with the installation of the ESP32 microcontroller and ultrasonic sensors. Real-time data is recorded by the sensors, which are installed for detecting cracks, and the data is passed through Wi-Fi to the cloud. The platform ThingSpeak, which stores, evaluates, and visualizes data collected from sensors, is employed to manage the cloud integration. A web interface that accurately provides information and gives alerts to users is built deploying basic web technologies, which include HTML, CSS, and JavaScript. The interface provides data in a straightforward manner upon obtaining it from the cloud. During the implementation phase, thorough testing is performed to validate accuracy in data gathering, exchange, and showcase along with the web interface's appropriate performance. The system's potential to consistently recognize and convey crack data is proven by producing and examining a simple prototype. In Final Year Project 2, the system will be improved, its efficiency will be optimized, and the way it operates will be strengthened in order to accomplish all of the intended objectives.

### **3.8 Testing**

The system's fulfilment of both functional and non-functional requirements is assessed during the testing procedures, which will be done in Final Year Project 2. The capability of ultrasonic sensors to recognize small cracks and deliver prompt alerts is initially demonstrated by testing multiple crack situations. Ongoing evaluation tests are carried out to determine the system's consistency with the aim of detecting any issues or anomalies in the data flow transmitted between the ESP32 and the cloud-based system. The web interface's accessibility is reviewed by proving that it provided historical trends, warnings, and real-time data in an intuitive and readily available way. Furthermore, log the outcomes of the tests to guarantee that the system fulfilled the requirements of users.

### **3.9 Deployment**

In Final Year Project 2, the IoT-based crack detection and monitoring system will be deployed on a test platform during the deployment phase. This includes configuring the ultrasonic sensors and ESP32 microcontroller, confirming that they are turned on, and establishing them for transmitting data to the cloud platform. Users will be allowed to gain access to the web interface, which will provide real-time structural crack monitoring. To achieve the best coverage and precision, the positioning of sensors will be thoroughly considered. The system's efficiency and dependability will be examined in real-life situations. In Final Year Project 2, the deployment will be expanded, the results analysed, and modifications conducted according to the results of the initial assessments.

### **3.10 Review**

The whole efficacy of the system and its adherence to the project's objectives will be analysed within the review phase. During this phase, which will be executed as part of Final Year Project 2, crucial variables, including the user experience with the web interface, data exchange dependability, and crack recognition precision, will all be

assessed. To figure out areas for advancement, for instance, the chance of reducing the need for maintenance and minimizing reliance on internet connections, input from users will be acquired and evaluated. The assessment will decide whether the system fulfils both functional and non-functional demands and provides recommendations for further improvements. The system's performance for broad adoption will be strengthened by adjustments and refinements according to the review.

### **3.11 Summary**

Chapter 3 specifies the necessities, layout, and execution strategies for the IoT-based crack detection system. It describes the methodology, incorporating the details collected through questionnaires on users. The hardware and software requirements have been defined, and these involve components like ThingSpeak, Google Firebase, ESP32, and ultrasonic sensors. Remote accessibility, affordability, and real-time monitoring are assured by the system's both functional and non-functional necessities. Context and level diagrams have been employed in the system design to demonstrate the connections between all the different parts. The intention of the interface design is a clear, straightforward online interface. Moreover, Final Year Project 2 will encompass the implementation, deployment, and review phases with a particular focus on efficiency and assessment. This chapter outlines the guidelines to design the system that corresponds with the objectives of the project.

## **CHAPTER 4: IMPLEMENTATION**

### **4.1 Introduction**

This chapter identifies the implementation of the IoT-Based Crack Detection and Monitoring System using ESP32. The chapter explains the composition of a functional system for real-time crack monitoring and structural health assessment, which involves the integration of various hardware and software components. ESP32 microcontroller is used to fetch sensor data, which is then sent over Wi-Fi to cloud services such as Firebase and ThingSpeak for storage, visualization and analysis. These platforms allow monitoring crack growth remotely from a web interface. Automated alert mechanisms are developed using EmailJS and Twilio to notify the users through email and SMS when critical thresholds are reached. This chapter also discusses the detailed implementation of every component and system features, operational flow, and how every component contributes to proactive structural monitoring.

### **4.2 Proposed System Implementation**

#### **4.2.1 Setup of Firebase**

To cater for hosting a simple front-end web interface that presents the system status and appropriate alerts from the crack detection system, Firebase was used in this project. This interface will provide a central dashboard that the users can access from any browser while storage and visualization of data are managed differently on ThingSpeak. The Firebase setup started with the creation of a new project in the Firebase console, which was designated with an appropriate name that would depict the system's purpose, as shown in Figure 4.1.

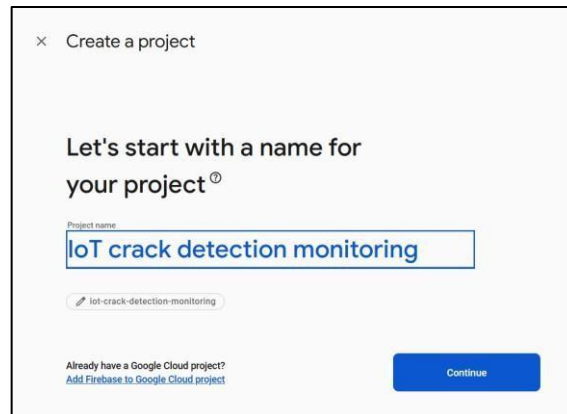


*Figure 4.1: Create a New Project in Firebase Console*

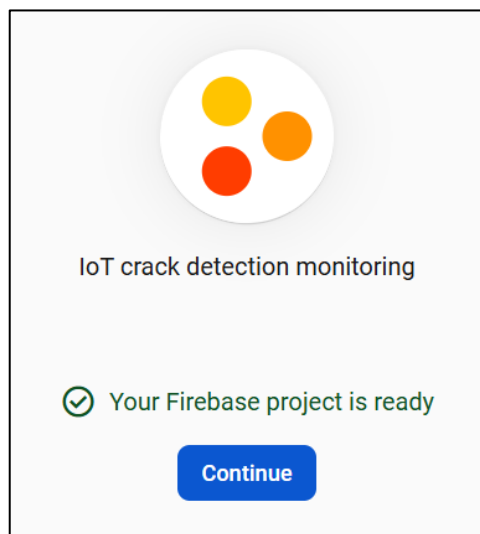


*Figure 4.2: Displays when the Project is Successfully Created*

After the project was formed as illustrates in Figure 4.2, Firebase Hosting was activated by going to “Build > Hosting” in the console, as shown in Figure 4.3. A default hosting site was automatically created with a domain assigned by Firebase.

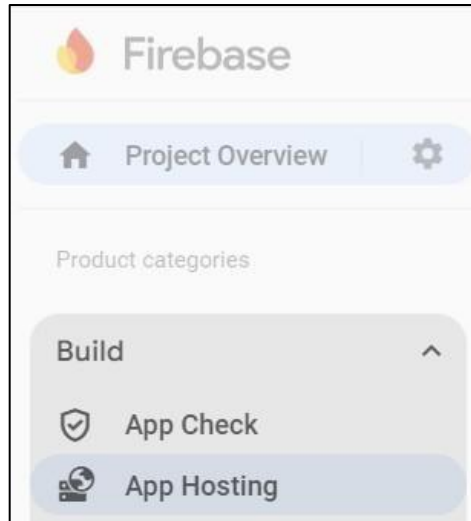


*Figure 4.3: Activating Firebase Hosting for the Project*

In order to control hosting and deployment on a local level, Firebase CLI was installed via the command `npm install -g firebase-tools`, as shown in Figure 4.4.

```
C:\Users\User>npm install -g firebase-tools  
added 82 packages, removed 18 packages, and changed 612 packages in 35s
```

*Figure 4.4: Run a Command to Install Firebase CLI*

The CLI was then authenticated with the help of logging in to the associated Google account using the `firebase login` command after installation. When authenticated, the project folder with the files of HTML, CSS, and JavaScript was initialized by using the `firebase init` command. During the initialisation, the Hosting option was chosen, and the public directory was selected as the directory of the web interface files. It was set so that it does not use Single Page App (SPA) routing and does not override the existing `index.html` file that was previously created manually. Figure 4.5 depicts all the commands to set up the Firebase Hosting.

```

C:\Users\User\OneDrive - UNIMAS\Desktop\iot-fyp2>firebase login
Already logged in as raiyanimal8@gmail.com

C:\Users\User\OneDrive - UNIMAS\Desktop\iot-fyp2>firebase init

#####
#          #          #          #          #          #          #          #          #          #
#          #          #          #          #          #          #          #          #          #
#          #          #          #          #          #          #          #          #          #
#          #          #          #          #          #          #          #          #          #
#          #          #          #          #          #          #          #          #          #
#####

You're about to initialize a Firebase project in this directory:

C:\Users\User\OneDrive - UNIMAS\Desktop\iot-fyp2

/ Are you ready to proceed? Yes
/ Which Firebase features do you want to set up for this directory? Press Space to select features, then Enter to
confirm your choices.
( ) Genkit: Setup a new Genkit project with Firebase
( ) Functions: Configure a Cloud Functions directory and its files
( ) App Hosting: Configure an apphosting.yaml file for App Hosting
> (*) Hosting: Configure files for Firebase Hosting and (optionally) set up GitHub Action deploys
( ) Storage: Configure a security rules file for Cloud Storage
( ) Emulators: Set up local emulators for Firebase products
( ) Remote Config: Configure a template file for Remote Config

```

Figure 4.5: Setup of Firebase Hosting for the Project

Firebase Hosting will automatically enable the HTTPS encryption, meaning no additional configuration had to be done to secure the hosted content. All files that get deployed through Firebase are automatically served with HTTPS secure transport via a free SSL/TLS certificate. This provides secure and encrypted communication between the users and the Firebase server, thereby ensuring that information about the states of the systems cannot be intercepted. Lastly, Figure 4.6 shows that the web interface was published online with the use of the Firebase deploy command. After deployment, a hosting URL was given and tested the live website to see if the front end was accessible and working in a correct manner. The deployed site acts as a public part of the system, which communicates the system with the current data from ThingSpeak and provides the users with a trusting demonstration of the results of crack monitoring.

```

C:\Users\User\Music\iot-crack-detection\public>firebase deploy

=== Deploying to 'crack-detection-monitoring-iot'...

i deploying hosting
i hosting[crack-detection-monitoring-iot]: beginning deploy...
i hosting[crack-detection-monitoring-iot]: found 12 files in public
+ hosting[crack-detection-monitoring-iot]: file upload complete
i hosting[crack-detection-monitoring-iot]: finalizing version...
+ hosting[crack-detection-monitoring-iot]: version finalized
i hosting[crack-detection-monitoring-iot]: releasing new version...
+ hosting[crack-detection-monitoring-iot]: release complete

+ Deploy complete!

Project Console: https://console.firebase.google.com/project/crack-detection-monitoring-iot/overview
Hosting URL: https://crack-detection-monitoring-iot.web.app

```

Figure 4.6: Deploying the Firebase Hosting Web Application

## 4.2.2 Setup of ThingSpeak

ThingSpeak was employed in this project as the primary IoT system on which sensor data

accumulated by the ESP32-based crack detection system gets stored, visualized, and analysed. ThingSpeak is an IoT analytics platform used for real-time data collection from microcontrollers and allows users to study interactive charts and conduct data analysis with the help of MATLAB. The setup started with the creation of a free MathWorks account, which is needed to access ThingSpeak services as seen in Figure 4.7.

Figure 4.7: Creating a Free MathWorks Account to Access ThingSpeak

After the verification of the account and its creation, a new channel was checked in in the ThingSpeak dashboard, which is depicted in Figure 4.8. As shown in Figure 4.9, the channel was personalized as it was given a meaningful name, for example, “IoT-Based Crack Detection and Monitoring System,” and meaningful labels were selected for each field derived from the sensor data, for example, “Crack Depth,” depending on the sensor used.

| Name                                            | Created    | Updated          |
|-------------------------------------------------|------------|------------------|
| IoT-Based Crack Detection and Monitoring System | 2025-04-21 | 2025-05-08 10:13 |

Figure 4.8: New Channel Created in ThingSpeak

**Channel Settings**

Percentage Complete 50%

Channel ID 2927927

Name IoT-Based Crack Detection and Monitoring System

Description This channel stores data from an IoT-based crack detection system using ESP32. It monitors crack depth.

Field 1 Crack depth (mm) ☒

*Figure 4.9: Setup of Channel Settings*

After the creation of the channel, the Write API Key, as displayed in Figure 4.10, was generated and is used to authenticate and send data from the ESP32 device to ThingSpeak. This key was then stored and used later to be put into an Arduino sketch in order to make communication success.

Private View Public View Channel Settings Sharing API Keys

**Write API Key**

Key PPOXU8C6IE1T33MR

Generate New Write API Key

*Figure 4.10: Write API Key for ThingSpeak*

The channel's sharing settings were set up then, private prior to testing and made public to let the Firebase-hosted dashboard show the charts with the use of embedded iframes or shared URLs, as shown in Figure 4.11.

Private View Public View Channel Settings Sharing API Keys

**Channel Sharing Settings**

☐ Keep channel view private

☒ Share channel view with everyone

☐ Share channel view only with the following users:

*Figure 4.11: Update the Channel Sharing Settings to Public*

Figure 4.12 shows that the visualization was done by using ThingSpeak’s built-in charting interface to generate real-time line graphs that automatically updated when new sensor data was pushed to the channel. The interval for update was defined based on the estimated frequency of data transmission from ESP32.

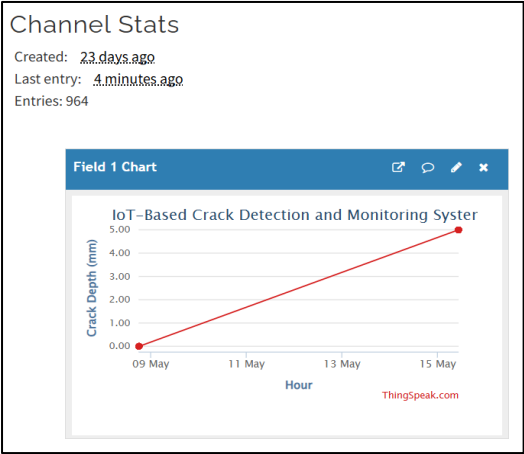


Figure 4.12: Line Graph of Real-Time Monitoring Data

In addition to this, the options to change the display setting in the chart were also altered by changing the colour, labels, gridlines, and axis titles for easy reading, as illustrated in Figure 4.13. No extra plugins and integration were needed, as most of the required features are native to ThingSpeak.

Field 1 Chart Options

Title: IoT-Based Crack Timescale:

X-Axis: Hour Average:

Y-Axis: Crack Depth (mm) Median:

Color: #d62020 Sum:

Background: #ffffff Rounding:

Type: line  Data Min:

Dynamic?: true  Data Max:

Days:  Y-Axis Min:

Results: 60 Y-Axis Max:

Save Cancel

Figure 4.13: Update the Display Setting for the Line Graph

After a channel was completely configured as well as tested, its public URL or

iframe was embedded into the Firebase interface, which hosted the web, and users were able to follow structural adjustments without requiring a login to ThingSpeak, as shown in Figure 4.14 below. This neat integration turned ThingSpeak into an effective and trustworthy solution to the data storage and visualization needs of the system.

```
<h2>Real-Time Data</h2>
<div class="iframe-container" style="text-align: center;">
  <iframe
    width="450"
    height="260"
    src="https://thingspeak.com/channels/2927927/charts/1?bgcolor=%23ffffff&color=%23d62020&dynamic=true&type=line&update=15"
    frameborder="0">
  </iframe>
</div>
```

*Figure 4.14: Block of Code to Display the Graph in the Web Interface*

### 4.2.3 Installation and Setup of Arduino IDE

The Arduino IDE was used in this project for programming as well as uploading the source code into the ESP32 microcontroller. Arduino IDE has a clean and easy-to-manage interface for writing and compiling codes to be uploaded in many microcontroller boards, such as the ESP32. The setup started with a download of the latest version of Arduino IDE from the official Arduino website, one that was compatible with Windows, Mac, and Linux, as depicted in Figure 4.15.

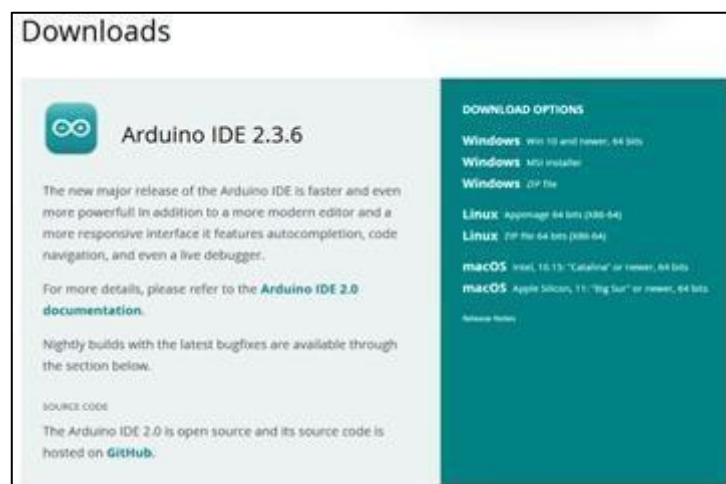


*Figure 4.15: The Latest Version of Arduino IDE*

When downloaded, the IDE was installed using the default installation

procedures, as seen in Figure 4.16. Once the installation process was over, the next thing was to download the files to configure IDE to work with the ESP32 board because it is not employed by default in Arduino IDE.

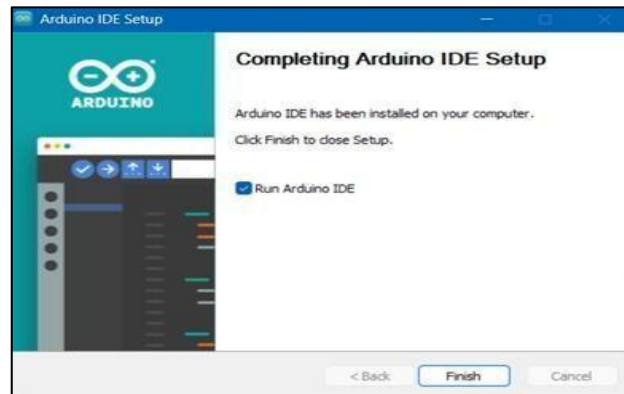


*Figure 4.16: Complete Installation of Arduino IDE*

Figure 4.17 illustrates that the ESP32 platform was installed through Tools > Board > Boards Manager by searching “ESP32” and installing the package created by the Espressif Systems.

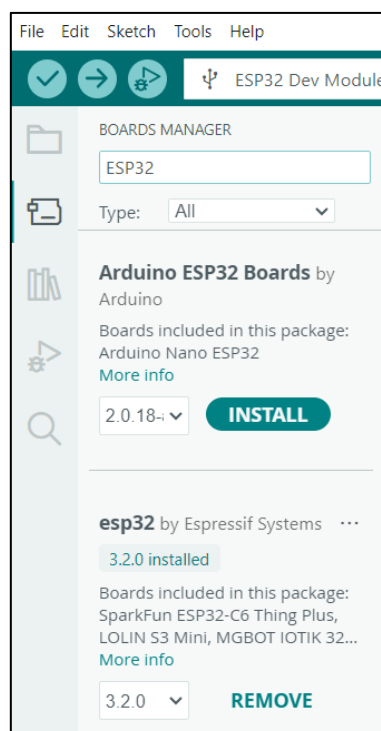


*Figure 4.17: Installation of ESP32 board by Espressif Systems*

The proper ESP32 board, once installed, was then chosen from the Tools > Board menu. For this project, the board that was utilized was the ESP32 Dev Module, which is one of the most commonly available variants that does support many ESP32 modules, as shown in Figure 4.18.

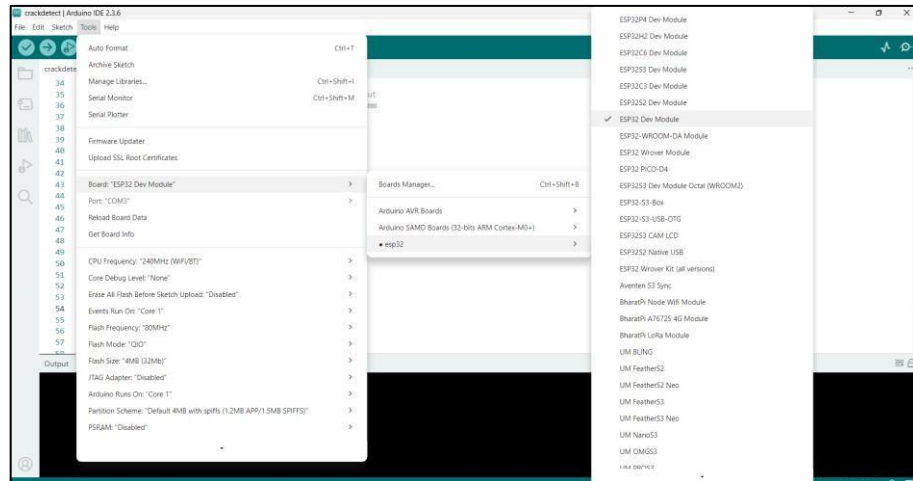


Figure 4.18: Choosing the Accurate Tools that are used for the Project

The necessary libraries after setting up the board to have successful communication with ThingSpeak and Wi-Fi networks. Figure 4.19 displays the comprised libraries that were installed by the Library Manager, like WiFi.h, HTTPClient.h, and ThingSpeak.h, among others. By using the ThingSpeak.h library, the ESP32 can post HTTP POST requests with sensor data to the ThingSpeak cloud platform. The source code was then written to retrieve values from the connected crack detection sensor, connect to the Wi-Fi network, and send the data over to the specified channel in ThingSpeak using the generated Write API Key.

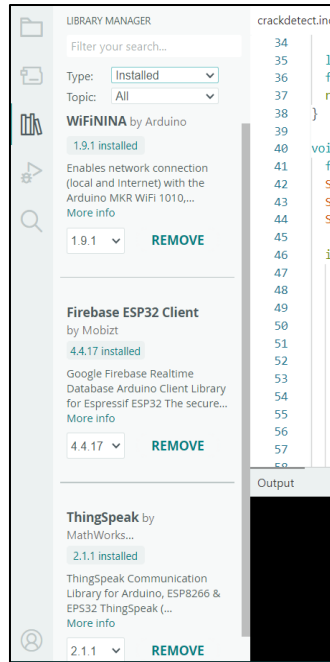


*Figure 4.19: Libraries that were Installed for the Project*

The right port and board settings were confirmed to be correct under the Tools menu before uploading the code to ESP32. The ESP32 was wired with the computer using a USB cable, and the COM port was used correspondingly. After all the configurations were checked, the sketch was compiled and uploaded to ESP32 by hitting the Upload button of the IDE as depicted in Figure 4.20. Once the upload was done successfully, Figure 4.21 shows the serial monitor was used to verify the status of the Wi-Fi connection, the status of the ThingSpeak update, and the debug messages to confirm that all was in order. Such a configuration enabled the continuous and automatic collection and transfer of data, which was the main logic of the crack monitoring system.

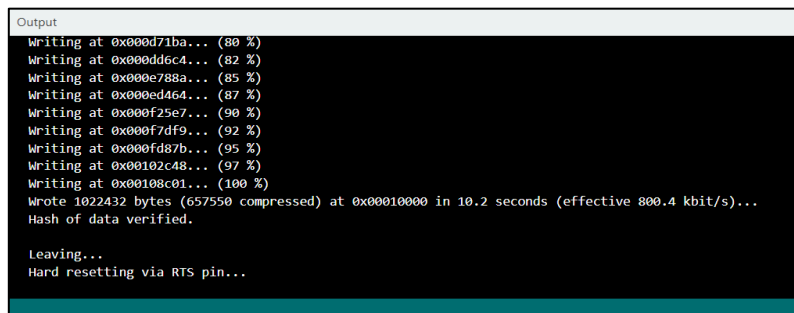


*Figure 4.20: Compiling and Uploading the Code to ESP32*

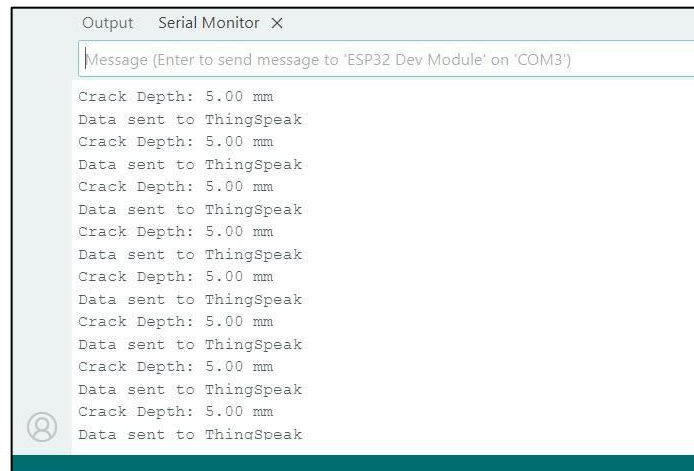


*Figure 4.21: Messages Displayed by Serial Monitor*

#### 4.2.4 Installation and Setup of Visual Studio Code

For developing the front-end interface of the crack monitoring system, Visual Studio Code (VS Code) was the primary code editor that was employed. It was the environment to develop and modify the HTML, CSS, and JavaScript files, which were later hosted in Firebase. The installation process started by downloading the latest version of VS Code from the official website of Visual Studio Code and then installing the latest version of VS Code by default in a Windows operating system using the default installation wizard, which is displayed in Figure 4.22.



*Figure 4.22: Installation of Visual Studio Code (VS Code)*

Once the editor was installed correctly, a new project folder was created locally, which contained all the necessary front-end files. In this folder, HTML, CSS, and JavaScript files were formed in the same directory.

#### 4.2.5 Setup of EmailJS and Twilio

To increase the functionality of the crack monitoring system, EmailJS and Twilio were integrated into the web interface's system to allow real-time email and SMS alerts as the monitoring values go over the threshold set. These services made the system able to promptly notify the user about possible structural concerns without having to continuously check the dashboard. Both processes were solved via the front end, using JavaScript.

The configuration started with the setup of EmailJS, which is a platform that can be used for email sending directly from the client-side, for which we shall not need anything other than JavaScript. To begin with, an account was made on EmailJS, and a new email service was chosen, Gmail in the current case, as shown in Figure 4.23 below.



*Figure 4.23: Created New Email Service using Gmail*

Figure 4.24 displays a custom email template that was created from the EmailJS dashboard, where placeholders like `{{message}}`, `{{from_name}}`, and `{{to_email}}` were declared to insert alert messages and recipient information from JavaScript code dynamically.

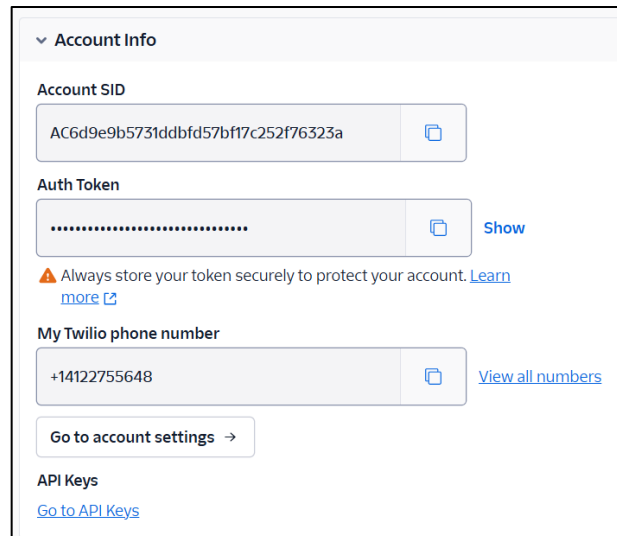
Figure 4.24: Customized an Email Template for Crack Alerts Message

Upon the building of the template, the service ID, template ID, and user public key were obtained and were later added in the JavaScript file. To make it possible to integrate, the EmailJS SDK was imported to the HTML file using a `<script>` tag referring to the EmailJS CDN. In the JavaScript file, there was a function written to initiate the email alert by catching relevant sensor data and sending it through the email template through the `emailjs.send()` method, as seen in Figure 4.25 below.

```
emailjs.send("service_u2gevis", "template_7b2lqho", emailParams)
  .then(() => {
    | console.log("✅ Alert email sent.");
  })
  .catch((err) => {
    | console.error("❌ Failed to send email:", err);
  });
```

Figure 4.25: Code to Initiate the Email Alert in JavaScript

For the SMS alert feature in the proposed system, there was an integration of Twilio through a straightforward backend configuration. The first step to the process was creating a Twilio account by visiting its signup page. This was followed by upgrading the account from the free trial to the full paid version to remove the messaging constraints. A verified phone number was attached, and the dashboard gave the right credentials: the Account SID, Auth Token, and a Twilio phone number retrieved through the “Get a Trial Number” option, which is displayed in Figure 4.26.



Account Info

Account SID  
AC6d9e9b5731ddb57b17c252f76323a

Auth Token  
..... Show

Always store your token securely to protect your account. [Learn more](#)

My Twilio phone number  
+14122755648 View all numbers

Go to account settings →

API Keys  
[Go to API Keys](#)

Figure 4.26: Twilio Account Information

For secure purposes of SMS sending, we have developed the backend server using Node.js and Express. An initiation of a project folder was carried out and named sms-server. Inside this project folder, a server.js file was created to serve POST requests to a /send-sms endpoint as shown in Figure 4.27. This script had the Twilio API enabled for sending SMS texts through the use of the given credentials.

```
// POST endpoint to send SMS
app.post('/send-sms', (req, res) => {
  const { to, message } = req.body;

  client.messages.create({
    body: message,
    from: '+14122755648', // Twilio trial number
    to: '+601119390387'
  })
  .then(message => {
    console.log('✅ SMS sent:', message.sid);
    res.send("SMS sent successfully");
  })
  .catch(err => {
    console.error("❌ Error:", err);
    res.status(500).send("Failed to send SMS");
  });
});

// Start server
const PORT = 3001;
app.listen(PORT, () => {
  console.log('🔥 Server running on http://localhost:${PORT}');
});
```

Figure 4.27: Code to Initiate the SMS Alert in JavaScript

### 4.3 Workflow of the Proposed System

The process of workflow of the proposed crack detection and monitoring system based on IoT is systematic and continuous to deliver accurate acquisition, processing, and real-time alerts based on acquired data. The system starts with the deployment of a sensor, which is an ultrasonic sensor, and it is linked to an ESP32 microcontroller. This sensor provides continuous monitoring of crack expansion or movement and sends data to the ESP32 for processing. The ESP32 pulls values from the sensor at a specified interval in the source code and prepares the data to send it to the cloud.

After the data is captured, the ESP32 goes on to connect to a Wi-Fi network that has been previously set in the Arduino code. After a successful connection, the ESP32 passes on the measured values to the ThingSpeak cloud platform through the HTTP POST requests. The data is then saved and displayed in real-time on the ThingSpeak dashboard by a private channel setup with charts and widgets. This dashboard helps users to view trends and historical data on crack activity outside the range of typical views through any web browser.

At the same time, ESP32 determines whether the sensor data passes the threshold value. If the data goes above a critical point, the ESP32 invokes a JavaScript function, which is embedded in the front-end interface, which is hosted on the Firebase Hosting service. This front-end dashboard created using HTML, CSS, and JavaScript and hosted in Firebase, offers an intuitive interface and shows the current sensor readings, ThingSpeak plotting, as well as alert messages.

When JavaScript realizes that a critical value has been passed, the system triggers an email notification using EmailJS and an SMS alert using Twilio, which are both integrated directly into the web application. These alerts will inform the user of the abnormal condition instantaneously without necessarily having to look at the dashboard.

This entire process runs autonomously after deployment, thus providing continuous monitoring, real-time logging of data, and timely notification whenever there is a case of structural anomalies. The architecture reduces the need for manual supervision not only but also provides a reliable cloud-integrated solution for the use of structural health monitoring in buildings and other infrastructure.

#### 4.4 System Feature and Operation

The proposed IoT-based building crack detection and monitoring system has several important characteristics that allow for real-time monitoring of data, cloud-based visualization, and notification emails and SMS alerts to the users. Each feature is realized with the help of hardware blocks and software tools.

##### 4.4.2 Real-time crack data monitoring

Monitoring crack depth in real-time with the use of an ultrasonic sensor attached to the ESP32 microcontroller is one of the key capabilities of the proposed IoT-based crack monitoring system. The deployment is carried out via the Arduino IDE, in which the sensor reads the distance of the sensor to the surface, and this indicates the depth of the crack. This is done in the `loop()` function, through which the `readDistanceMM()` method sends the ultrasonic pulse through `trigPin` as well as reads the echo response from the `echoPin`, which is depicted in Figure 4.28. The result, in millimeter units, reflects the present crack depth.

```
void loop() {  
  float distance = readDistanceMM();  
  Serial.print("Crack Depth: ");  
  Serial.print(distance);  
  Serial.println(" mm");  
}
```

*Figure 4.28: loop() Function to Read the Crack Depth*

As shown in Figure 4.29, when the data is obtained, the system tests for a stable Wi-Fi connection. When connected, the ESP32 builds an HTTP GET request, which includes the present value of the crack depth, and sends it to ThingSpeak through the HTTPClient library. The line “String url = thingSpeakServer + “?api\_key=” + apiKey + “&field1=”+ String(distance);” constructs the request URL dynamically, adding the latest reading that needs to be updated within the cloud. This whole loop executes every 15 seconds, which is also the minimum data update interval in ThingSpeak for free accounts to have near real-time monitoring.

```
if (WiFi.status() == WL_CONNECTED) {  
    HTTPClient http;  
    String url = thingSpeakServer + "?api_key=" + apiKey + "&field1=" + String(distance);  
  
    http.begin(url);  
    int httpResponseCode = http.GET();  
    if (httpResponseCode > 0) {  
        Serial.println("Data sent to ThingSpeak");  
    } else {  
        Serial.print("Failed to send data. Error code: ");  
        Serial.println(httpResponseCode);  
    }  
    http.end();  
}  
  
delay(15000); // 15 seconds delay (ThingSpeak limit)  
}
```

*Figure 4.29: Code for the Real-Time Crack Data Monitoring*

Figure 4.30 displays the user interface side where the web application allows fetching data from ThingSpeak and presents it in a live graph. This allows the users to see the changes in the crack depth at the same time as new readings are posted to the cloud. The graph gets updated dynamically to let the users be aware of any changes in the structure, which is crucial for safety and preventive maintenance.

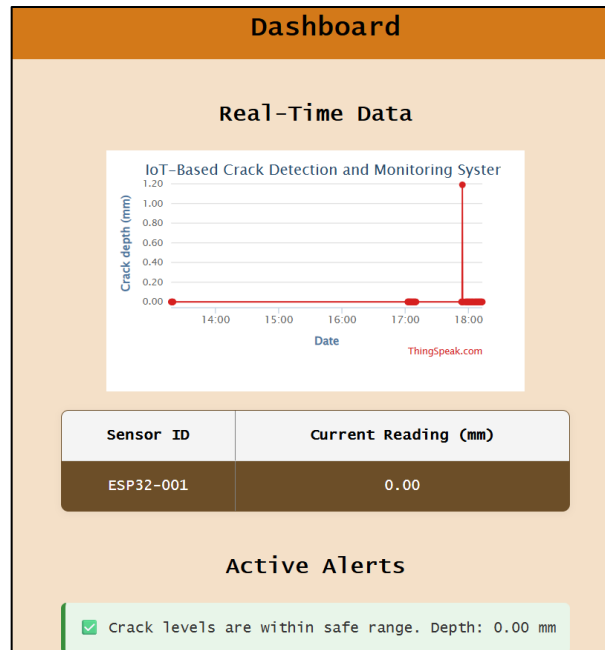


Figure 4.30: Dashboard Page that Displays the Real-Time Data

#### 4.4.3 Cloud-based data visualization

To obtain real-time observation about the structural crack depth, the system includes cloud-based data visualization with the help of ThingSpeak's embedded charting functions. Every 15 seconds, the ESP32 updates the crack measurement to the ThingSpeak cloud, and these values are represented on the dynamic line charts in real time. This enables the users, such as engineers or building managers, to see the changes in crack depth in a non-intrusive and continuous manner, without engaging with the hardware directly.

The visualization is inserted into the web interface of the monitoring system with the help of a `<iframe>` tag embedded into the HTML file, as depicted in Figure 4.31. This allows us to provide the live sensor data directly on the web application's dashboard, as shown in Figure 4.32 below. The iframe pulls the chart from the ThingSpeak platform that automatically displays the information that is stored in the corresponding channel.

```

<iframe
width="450"
height="260"
src="https://thingspeak.com/channels/2927927/charts/1?bgcolor=%23ffffff&color=%23d62020&dynamic=true&type=line&update=15"
frameborder="0">
</iframe>

```

*Figure 4.31: Code that Displays the ThingSpeak Line Graph*

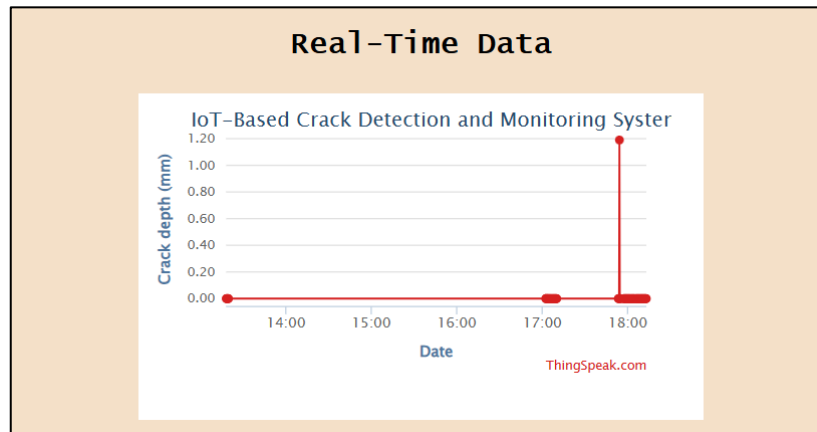


*Figure 4.32: The Live Sensor Data Being Displayed on the Dashboard Page*

#### 4.4.4 Threshold detection and alert trigger

To further improve safety and allow for early warning mechanisms, the system has a threshold-based detection mechanism. Users can set a threshold for crack depth in the settings of the web application. If the most recent crack readings drawn from the ThingSpeak channel go above this threshold, then the system will automatically send alert emails or text messages to the users, depending on the method they choose.

The feature of threshold detection is present in the `checkCrackDepth()` function of the JavaScript code, as illustrated in Figure 4.33. End-users can enter a crack depth limit value, which they can save on the web interface of the application, and this value is stored in the browser's `localStorage`. Upon loading the dashboard or updating the threshold settings by the user, the application pulls the current reading on crack depth through the ThingSpeak API. This reading is compared with the user-defined threshold value by the system. If the present crack depth goes above the threshold, then the next step is to trigger alerts depending on the user's chosen method of notification.

```
function checkCrackDepth() {
  const threshold = parseFloat(localStorage.getItem("crackThreshold")) || 0;
  const notificationMethod = localStorage.getItem("notificationMethod");

  fetch(`https://api.thingspeak.com/channels/${channelID}/fields/${field}/last.json?api_key=${apiKey}`)
    .then(response => response.json())
    .then(data => {
      const crackDepth = parseFloat(data[`field${field}`]);

      if (crackDepth > threshold) {
        console.log(` ⚠ Crack depth exceeded: ${crackDepth} mm`);
      }
    });
}
```

Figure 4.33: Code for checkCrackDepth() Function for the Threshold Detection Feature

```
if (notificationMethod === "email") {
  const userEmail = auth.currentUser?.email || "babymya78@gmail.com"; // fallback email

  const emailParams = {
    to_email: userEmail,
    threshold: threshold,
    current_depth: crackDepth
  };

  emailjs.send("service_u2gevis", "template_7b2lqho", emailParams)
    .then(() => {
      console.log("✅ Alert email sent.");
    })
    .catch((err) => {
      console.error("❌ Failed to send email:", err);
    });
}
```

Figure 4.34: Code for Email Alert Notification

```
if (notificationMethod === "sms") {
  const phone = localStorage.getItem("notificationPhone");

  fetch("http://localhost:3001/send-sms", {
    method: "POST",
    headers: {
      "Content-Type": "application/json"
    },
    body: JSON.stringify({
      to: phone,
      message: ` ⚠ Crack depth exceeded!\n\nThe latest crack depth reading is ${crackDepth} mm, which has exceeded your set threshold of ${threshold} mm.\n\nPlease check the monitoring system for further action.`
    })
  })
    .then(res => res.text())
    .then(result => {
      console.log("✅ SMS Alert Sent:", result);
    })
    .catch(err => {
      console.error("❌ SMS Error:", err);
    });
}
```

Figure 4.35: Code for SMS Alert Notification

If the user had chosen email for notification, the system would have used EmailJS to alert the user at the email address, as shown in Figure 3.34 above. On the other hand, when SMS is picked, the code for the SMS method is initiated, which is depicted in

Figure 4.35. The system sends a POST request to a local backend server, and the alert message content, as seen in Figure 4.36. Figure 4.37 and Figure 4.38 show the email and SMS alert notifications that were sent to the user respectively.

```
C:\Users\User\Music\iot-crack-detection\sms-server>node server.js
🚀 Server running on http://localhost:3001
✅ SMS sent: SM205e3aa92033d576ea0d6717d5a8218c
```

*Figure 4.36: SMS Server Console Output*

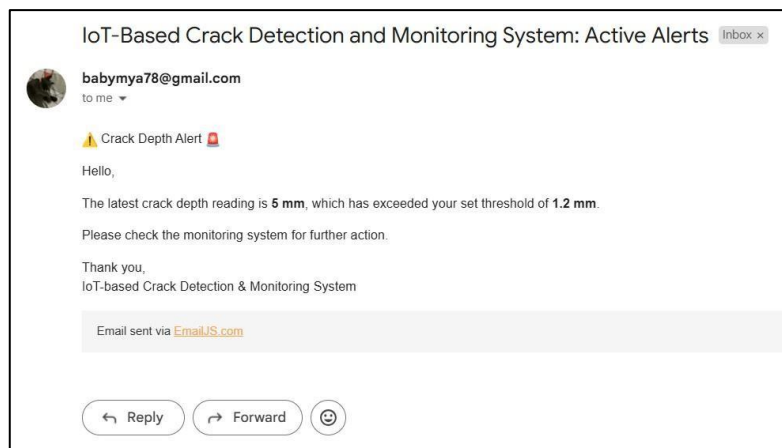


*Figure 4.37: Email Alert Notification Message*

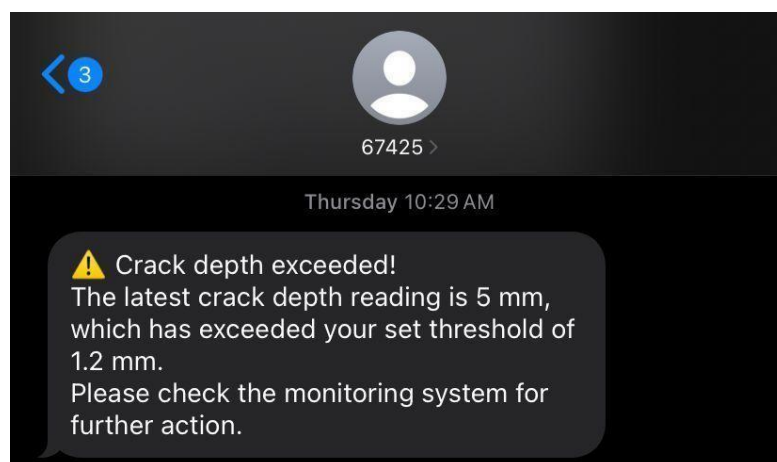
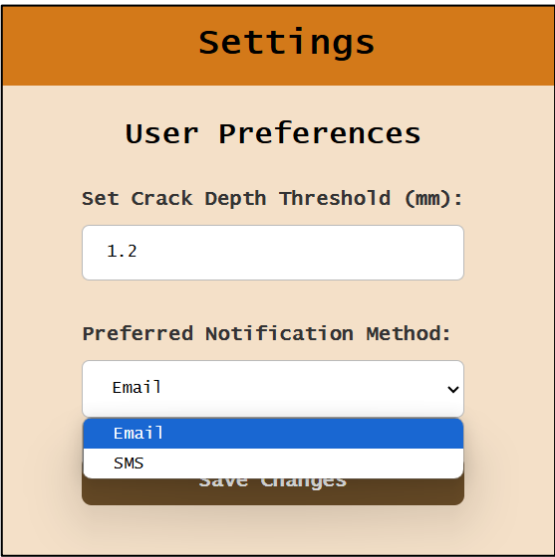


*Figure 4.38: SMS Alert Notification Message*

In the corresponding web application, the interface provides a means through which the user can customize the threshold value and select a favourite notification method

among the options available, as displayed in Figure 4.39. Email or SMS and enter a phone number that only appears if the user chooses SMS, as shown in Figure 4.40. Once the settings are saved, the system monitors the crack depth values in real-time thereafter using the saved threshold and automatically issues alerts as it should. This makes sure that users get information about any prospective structural problems that need their attention in a timely manner.



**Settings**

**User Preferences**

Set Crack Depth Threshold (mm):

1.2

Preferred Notification Method:

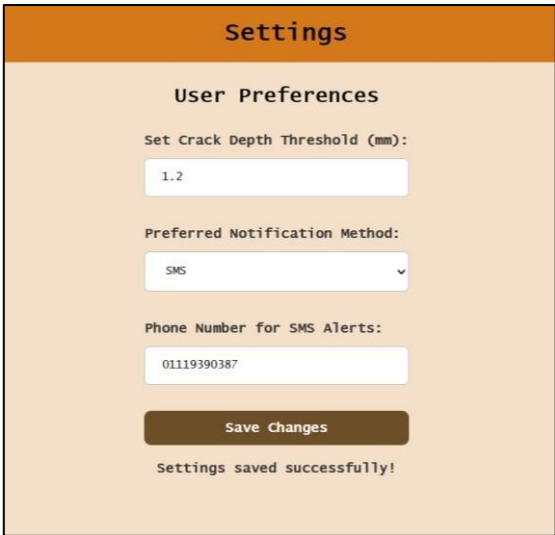
Email

Email

SMS

Save Changes

Figure 4.39: Customization of Threshold Value and Notification Method



**Settings**

**User Preferences**

Set Crack Depth Threshold (mm):

1.2

Preferred Notification Method:

SMS

Phone Number for SMS Alerts:

01119390387

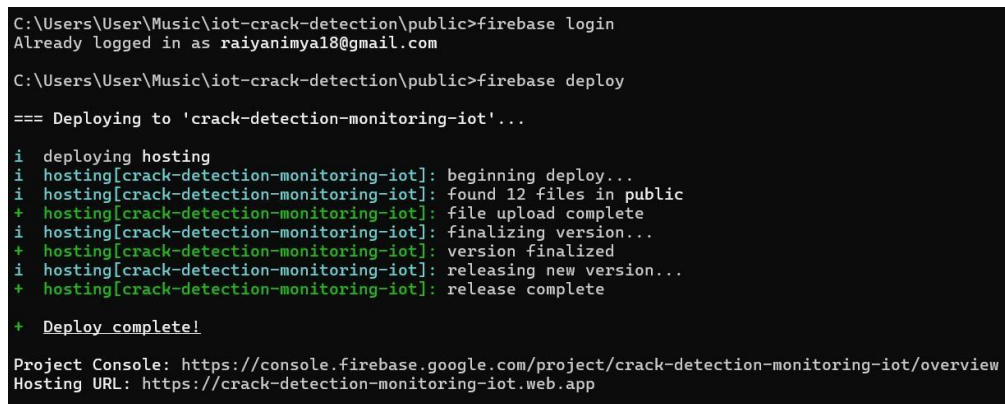
Save Changes

Settings saved successfully!

Figure 4.40: SMS Chosen as Notification Method

#### 4.4.5 Firebase hosting and web application deployment

Every HTML, CSS, and JavaScript file is sent to Firebase Hosting to make the dashboard public. After completion of the design in Visual Studio Code, the files were added in the /public directory, which was deployed using the firebase deploy command, as shown in Figure 4.41. This enabled one to get to the real-time dashboard through a custom Firebase URL, meaning that the system could be viewed on any internet-connected device.

A terminal window showing the execution of Firebase commands. The user is in the directory C:\Users\User\Music\iot-crack-detection\public. They run 'firebase login' and are already logged in as raiyanimyal8@gmail.com. Then they run 'firebase deploy'. The output shows the deployment to 'crack-detection-monitoring-iot' with various status messages: 'i deploying hosting', 'i hosting[crack-detection-monitoring-iot]: beginning deploy...', 'i hosting[crack-detection-monitoring-iot]: found 12 files in public', '+ hosting[crack-detection-monitoring-iot]: file upload complete', 'i hosting[crack-detection-monitoring-iot]: finalizing version...', '+ hosting[crack-detection-monitoring-iot]: version finalized', 'i hosting[crack-detection-monitoring-iot]: releasing new version...', '+ hosting[crack-detection-monitoring-iot]: release complete', and '+ Deploy complete!'. At the bottom, it provides the Project Console URL (https://console.firebase.google.com/project/crack-detection-monitoring-iot/overview) and the Hosting URL (https://crack-detection-monitoring-iot.web.app).

```
C:\Users\User\Music\iot-crack-detection\public>firebase login
Already logged in as raiyanimyal8@gmail.com

C:\Users\User\Music\iot-crack-detection\public>firebase deploy

=== Deploying to 'crack-detection-monitoring-iot'...

i  deploying hosting
i  hosting[crack-detection-monitoring-iot]: beginning deploy...
i  hosting[crack-detection-monitoring-iot]: found 12 files in public
+  hosting[crack-detection-monitoring-iot]: file upload complete
i  hosting[crack-detection-monitoring-iot]: finalizing version...
+  hosting[crack-detection-monitoring-iot]: version finalized
i  hosting[crack-detection-monitoring-iot]: releasing new version...
+  hosting[crack-detection-monitoring-iot]: release complete

+  Deploy complete!

Project Console: https://console.firebase.google.com/project/crack-detection-monitoring-iot/overview
Hosting URL: https://crack-detection-monitoring-iot.web.app
```

*Figure 4.41: Deployment of Firebase Hosting*

#### 4.5 Summary

In this chapter, a detailed presentation of the full implementation of the proposed IoT-based crack detection and monitoring system has been discussed. The implementation started with step-by-step setting up of the fundamental tools and platforms like Firebase Hosting, ThingSpeak, Arduino IDE, Visual Studio Code, EmailJS, and Twilio. All platforms took a significant role in the system's functionality and integration. The front-end dashboard was deployed using Firebase Hosting. The ThingSpeak cloud platform was used to store and visualize real-time cloud data; Arduino IDE was used to program the ESP32 microcontroller to read sensor data and send it to the cloud.

Visual Studio Code was the main development environment used to create the web interface with the help of HTML, CSS, and JavaScript. EmailJS and Twilio were effectively deployed on the front end to give a real-time alert to the users through email

and SMS correspondingly. The system features, like real-time data monitoring, cloud visualization, threshold detection, and automatic notifications, were described with the exact implementation of codes used in them for explaining how each of them was implemented and how they worked.

It was illustrated through the workflow of the system how sensor data fed in from the physical world down into the ESP32 and through the cloud to reach the user interface and alert mechanisms. This meant that the emergence of critical developments in crack could be identified and effectively dealt with in time. Overall, this chapter described the practical implementation of the system and its performance for real-time monitoring and alerts to lay down the grounds for a reliable and cost-effective structural health monitoring solution.

## **CHAPTER 5: TESTING**

### **5.1 Introduction**

The proposed IoT-Based Crack Detection and Monitoring System needs testing to verify its dependability, correctness, and performance. This chapter discusses how the system's performance and operation are checked against the requirements set by the project team. Testing is classified as functional testing and non-functional testing. Functional testing checks that all the system modules work correctly and are properly integrated, but non-functional testing examines how reliable, how fast, and how usable the system is. They help make sure the system is strong, easy to use, and everything users want it to be.

### **5.2 Functional Testing**

Functional testing ensures that all components and elements of the system act the way they should. This makes sure each piece of the system is working right and is suitable for the user. The main purpose is to identify any problems in the system's key functions, including data from sensors, exchanged messages over networks, information being sent, and how the user interacts with the system.

Testing was done manually using prepared test cases, and the real outcomes were compared to the predicted outcomes. Within this project, functional testing consisted of two types, including unit testing to validate the basic sensor modules and APIs and integration testing to confirm the connections among these parts. The system was analysed for its responses to legal and illegal inputs to see how strong it was and how it managed errors. Functional testing allows the project to verify that each component in the IoT crack detection system, ranging from hardware and back to the frontend, is working properly all the time.

### 5.2.1 Unit Testing

A unit test checks whether each module does what it is supposed to do when running by itself. One of the most important parts of this project was conducting unit tests to confirm the sensor input module, ESP32 Wi-Fi communications, data sending via ThingSpeak, and displaying in Firebase worked correctly. Every test case was designed with set aims, data to operate on, and the outputs expected to ensure testing all kinds of situations. White-box testing was selected to test how each module worked, especially when their code read from sensors or sent information to the cloud for processing.

As part of unit testing, the module testing process was also performed on the web interface for admin login, dashboard display, settings options, and historical data management components. With these tests, it was confirmed that users could benefit from accurate data, and their actions in logging in, logging out, searching through, and exporting data were all functioning as intended. The positive results from unit testing provided a basis for putting together the entire system, and all modules showed they had successfully met their test case requirements.

The test cases and results of unit tests are shown in Table 5.1 to 5.8.

Table 5.1 shows the test case for the unit testing of sensor reading.

Table 5.1: Unit Test Case for Sensor Reading Module

|               |                                                    |                                                                            |                |                           |               |        |
|---------------|----------------------------------------------------|----------------------------------------------------------------------------|----------------|---------------------------|---------------|--------|
| ID            | 1                                                  | Module Name                                                                | Sensor Reading |                           |               |        |
| Objective     |                                                    | To test the accuracy of sensor reading from ultrasonic sensor              |                |                           |               |        |
| Name          | Nur Raiyani                                        |                                                                            | Date           | 25 <sup>th</sup> May 2025 | Status        | Pass   |
| No            | Prerequisites                                      |                                                                            |                |                           |               |        |
| 1             | ESP32 is powered and connected.                    |                                                                            |                |                           |               |        |
| 2             | Ultrasonic sensor is correctly connected to ESP32. |                                                                            |                |                           |               |        |
| 3             | Serial Monitor is open for data monitoring         |                                                                            |                |                           |               |        |
| Test Scenario |                                                    | Sensor reads crack data accurately when placed in front of a test surface. |                |                           |               |        |
| ID            | Test Case                                          |                                                                            | Input Data     | Expected Result           | Actual Result | Status |

|   |                       |                                                  |                                         |             |      |
|---|-----------------------|--------------------------------------------------|-----------------------------------------|-------------|------|
| 1 | Simulate 0.5 mm crack | Place object with a crack in front of the sensor | Serial Monitor prints crack depth in mm | As Expected | Pass |
| 2 | Simulate no crack     | Place object with no crack                       | Serial Monitor prints 0 mm              | As Expected | Pass |

Table 5.2 shows the test case for the unit testing of Wi-Fi connection.

Table 5.2: Unit Test Case for Wi-Fi Connection

|               |                                               |                                                                   |                  |                           |        |      |
|---------------|-----------------------------------------------|-------------------------------------------------------------------|------------------|---------------------------|--------|------|
| ID            | 2                                             | Module Name                                                       | Wi-Fi Connection |                           |        |      |
| Objective     |                                               | To test the ability of ESP32 to connect to a Wi-Fi network        |                  |                           |        |      |
| Name          | Nur Raiyani                                   |                                                                   | Date             | 25 <sup>th</sup> May 2025 | Status | Pass |
| No            | Prerequisites                                 |                                                                   |                  |                           |        |      |
| 1             | ESP32 is powered.                             |                                                                   |                  |                           |        |      |
| 2             | Correct SSID and password configured in code. |                                                                   |                  |                           |        |      |
| 3             | Wi-Fi is active and within range.             |                                                                   |                  |                           |        |      |
| Test Scenario |                                               | ESP32 connects to Wi-Fi within 5 seconds and prints confirmation. |                  |                           |        |      |

| ID | Test Case         | Input Data            | Expected Result                            | Actual Result | Status |
|----|-------------------|-----------------------|--------------------------------------------|---------------|--------|
| 1  | Connect to Wi-Fi  | Correct credentials   | "WiFi Connected" printed on Serial Monitor | As Expected   | Pass   |
| 2  | Wrong credentials | Incorrect credentials | Connection failed                          | As Expected   | Pass   |

Table 5.3 shows the test case for the unit testing of ThingSpeak data transmission.

Table 5.3: Unit Test Case for ThingSpeak Transmission

|               |                                   |                                                                |                                   |               |        |
|---------------|-----------------------------------|----------------------------------------------------------------|-----------------------------------|---------------|--------|
| ID            | 3                                 | Module Name                                                    | ThingSpeak Data Transmission      |               |        |
| Objective     |                                   | To verify that sensor data is uploaded to ThingSpeak correctly |                                   |               |        |
| Name          | Nur Raiyani                       | Date                                                           | 25 <sup>th</sup> May 2025         | Status        | Pass   |
| No            | Prerequisites                     |                                                                |                                   |               |        |
| 1             | Wi-Fi is connected.               |                                                                |                                   |               |        |
| 2             | ThingSpeak API key is configured. |                                                                |                                   |               |        |
| 3             | ThingSpeak channel is active.     |                                                                |                                   |               |        |
| Test Scenario |                                   | ESP32 sends sensor data to ThingSpeak every 15 seconds.        |                                   |               |        |
| ID            | Test Case                         | Input Data                                                     | Expected Result                   | Actual Result | Status |
| 1             | Send value to ThingSpeak          | Sensor = 0.5 mm                                                | ThingSpeak graph updates with 0.5 | As Expected   | Pass   |
| 2             | No input data                     | Sensor disconnected                                            | Last known value shown            | As Expected   | Pass   |

Table 5.4 shows the test case for the unit testing of user login in the web app.

Table 5.4: Unit Test Case for User Login

|               |                                                             |                                                                  |                                     |               |        |
|---------------|-------------------------------------------------------------|------------------------------------------------------------------|-------------------------------------|---------------|--------|
| ID            | 4                                                           | Module Name                                                      | User Login                          |               |        |
| Objective     |                                                             | To test the login functionality of the Firebase web interface    |                                     |               |        |
| Name          | Nur Raiyani                                                 | Date                                                             | 25 <sup>th</sup> May 2025           | Status        | Pass   |
| No            | Prerequisites                                               |                                                                  |                                     |               |        |
| 1             | Web application is hosted and accessible.                   |                                                                  |                                     |               |        |
| 2             | User credentials are configured in Firebase Authentication. |                                                                  |                                     |               |        |
| 3             | Internet connection is active.                              |                                                                  |                                     |               |        |
| Test Scenario |                                                             | User attempts to log in using correct and incorrect credentials. |                                     |               |        |
| ID            | Test Case                                                   | Input Data                                                       | Expected Result                     | Actual Result | Status |
| 1             | Login with valid credentials                                | Correct credentials                                              | Redirect to dashboard               | As Expected   | Pass   |
| 2             | Login with wrong password                                   | Incorrect credentials                                            | Error: (auth/invalid - credential)  | As Expected   | Pass   |
| 3             | Login with invalid email                                    | Incorrect credentials                                            | Error: (auth/invalid - credential)  | As Expected   | Pass   |
| 4             | Login with empty fields                                     | Email: (empty)<br>Password: (empty)                              | Error: "Please fill out this field" | As Expected   | Pass   |

Table 5.5 shows the test case for the unit testing of Firebase web display.

Table 5.5: Unit Test Case for Firebase Web Display

|               |                                                             |                                                                |                                |                           |        |      |
|---------------|-------------------------------------------------------------|----------------------------------------------------------------|--------------------------------|---------------------------|--------|------|
| ID            | 5                                                           | Module Name                                                    | Firebase Web Display           |                           |        |      |
| Objective     |                                                             | To test the display of sensor data in Firebase web dashboard   |                                |                           |        |      |
| Name          | Nur Raiyani                                                 |                                                                | Date                           | 25 <sup>th</sup> May 2025 | Status | Pass |
| No            | Prerequisites                                               |                                                                |                                |                           |        |      |
| 1             | Firebase web interface is connected to ESP32 and ThingSpeak |                                                                |                                |                           |        |      |
| 2             | Internet is available.                                      |                                                                |                                |                           |        |      |
| 3             | User must be logged in.                                     |                                                                |                                |                           |        |      |
| Test Scenario |                                                             | Web interface retrieves and displays latest data in real-time. |                                |                           |        |      |
| ID            | Test Case                                                   | Input Data                                                     | Expected Result                | Actual Result             | Status |      |
| 1             | Display crack data                                          | Firebase receives 0.5 mm                                       | UI shows "Crack Depth: 0.5 mm" | As Expected               | Pass   |      |
| 2             | No new data                                                 | Firebase not updated                                           | Old data remains visible       | As Expected               | Pass   |      |

|   |                            |                    |                    |             |      |
|---|----------------------------|--------------------|--------------------|-------------|------|
| 3 | Data updates in real- time | New data = 1.50 mm | Chart auto-updates | As Expected | Pass |
|---|----------------------------|--------------------|--------------------|-------------|------|

Table 5.6 shows the test case for the unit testing of the setting page in the web interface.

Table 5.6: Unit Test Case for Settings Page

|               |                                                            |                                                                                                                            |                           |        |      |
|---------------|------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|---------------------------|--------|------|
| ID            | 6                                                          | Module Name                                                                                                                | Settings Page             |        |      |
| Objective     |                                                            | To test the ability to configure threshold values and alert preferences                                                    |                           |        |      |
| Name          | Nur Raiyani                                                | Date                                                                                                                       | 25 <sup>th</sup> May 2025 | Status | Pass |
| No            | Prerequisites                                              |                                                                                                                            |                           |        |      |
| 1             | User is logged in.                                         |                                                                                                                            |                           |        |      |
| 2             | Local storage is configured to store alert settings.       |                                                                                                                            |                           |        |      |
| 3             | Email and SMS APIs are integrated and functioning.         |                                                                                                                            |                           |        |      |
| 4             | Threshold alert logic is implemented on backend and ESP32. |                                                                                                                            |                           |        |      |
| Test Scenario |                                                            | User updates the crack threshold limit and chooses notification channels to receive alerts when the threshold is exceeded. |                           |        |      |

| ID | Test Case              | Input Data                           | Expected Result                                          | Actual Result | Status |
|----|------------------------|--------------------------------------|----------------------------------------------------------|---------------|--------|
| 1  | View existing settings | Open Settings page                   | Current threshold and notification preferences displayed | As Expected   | Pass   |
| 2  | Update threshold value | Enter: 2.0 mm                        | New value saved to local storage                         | As Expected   | Pass   |
| 3  | Input invalid value    | “abc”                                | Cannot enter other format than number                    | As Expected   | Pass   |
| 4  | Enable email alert     | Choose Email                         | Email setting saved                                      | As Expected   | Pass   |
| 5  | Enable SMS alert       | Choose SMS, Enter valid phone number | SMS setting saved                                        | As Expected   | Pass   |
| 6  | Enter invalid phone    | abcd1234                             | Cannot enter other format than number                    | As Expected   | Pass   |
| 7  | Trigger threshold      | Simulate crack width = 6.16 mm       | Alert sent to configured email/SMS                       | As Expected   | Pass   |

Table 5.7 shows the test case for the unit testing of the data history page.

Table 5.7: Unit Test Case for Data History Page

|               |                                    |                                                                  |                                                               |                           |        |      |
|---------------|------------------------------------|------------------------------------------------------------------|---------------------------------------------------------------|---------------------------|--------|------|
| ID            | 7                                  | Module Name                                                      | Data History Page                                             |                           |        |      |
| Objective     |                                    | To test the ability to view and manage historical crack readings |                                                               |                           |        |      |
| Name          | Nur Raiyani                        |                                                                  | Date                                                          | 25 <sup>th</sup> May 2025 | Status | Pass |
| No            | Prerequisites                      |                                                                  |                                                               |                           |        |      |
| 1             | User is logged in.                 |                                                                  |                                                               |                           |        |      |
| 2             | Data is available in ThingSpeak.   |                                                                  |                                                               |                           |        |      |
| 3             | Web interface can query past data. |                                                                  |                                                               |                           |        |      |
| Test Scenario |                                    | User views and searches through past crack monitoring data.      |                                                               |                           |        |      |
| ID            | Test Case                          | Input Data                                                       | Expected Result                                               | Actual Result             | Status |      |
| 1             | Search by date                     | Enter:<br>"2025-04-01"                                           | Readings from that date displayed                             | As Expected               | Pass   |      |
| 2             | No history available               | Empty ThingSpeak data                                            | "No data available for the selected date range" message shown | As Expected               | Pass   |      |

Table 5.8 shows the test case for the unit testing of logging out.

Table 5.8: Unit Test Case for Logout

|               |                                      |                                                  |                           |               |        |
|---------------|--------------------------------------|--------------------------------------------------|---------------------------|---------------|--------|
| ID            | 8                                    | Module Name                                      | Logout                    |               |        |
| Objective     |                                      | To test the logout functionality for admin users |                           |               |        |
| Name          | Nur Raiyani                          | Date                                             | 25 <sup>th</sup> May 2025 | Status        | Pass   |
| No            | Prerequisites                        |                                                  |                           |               |        |
| 1             | User is logged in.                   |                                                  |                           |               |        |
| 2             | Logout button is visible and active. |                                                  |                           |               |        |
| Test Scenario |                                      | User clicks logout button.                       |                           |               |        |
| ID            | Test Case                            | Input Data                                       | Expected Result           | Actual Result | Status |
| 1             | Normal logout                        | Click “Logout”                                   | Redirect to login page    | As Expected   | Pass   |
| 2             | Reload after logout                  | Refresh browser                                  | Stays on login page       | As Expected   | Pass   |

### 5.2.2 Integration Testing

Integration testing ensures that every part connected in the IoT-Based Crack Detection and Monitoring System acts in sync along the whole system. It ensures that communication and transfer of information are occurring properly from the hardware to

the cloud platform and into the web application created with Firebase. The ESP32 gets the crack data and posts this information directly to ThingSpeak, which stores all the data. The web app gets this information from ThingSpeak's API and shows it as both current and past statistics to users. Tests were carried out on the Firebase login system to confirm that only people who have signed in can see the settings, the status of alerts, and their past data. The tests included checking if users' threshold values trigger an alert either via email or SMS after their crack data goes past the set limit on ThingSpeak. It checks if hardware, backend, and frontend elements consistently work as a group to provide accurate and timely monitoring of structural cracks. The test cases and results of the integration test are shown in Table 5.9 and Table 5.10.

Table 5.9 shows the test case for the integration testing of hardware to cloud platform.

Table 5.9: Integration Test Case for Hardware to Cloud Platform

|               |                                                    |                                                                                                  |                                   |                           |        |      |
|---------------|----------------------------------------------------|--------------------------------------------------------------------------------------------------|-----------------------------------|---------------------------|--------|------|
| ID            | 9                                                  | Module Name                                                                                      | Hardware to Cloud Platform        |                           |        |      |
| Objective     |                                                    | To test if sensor data is correctly read, processed, and uploaded to ThingSpeak                  |                                   |                           |        |      |
| Name          | Nur Raiyani                                        |                                                                                                  | Date                              | 25 <sup>th</sup> May 2025 | Status | Pass |
| No            | Prerequisites                                      |                                                                                                  |                                   |                           |        |      |
| 1             | Sensor is connected and calibrated.                |                                                                                                  |                                   |                           |        |      |
| 2             | ESP32 is powered and connected to Wi-Fi.           |                                                                                                  |                                   |                           |        |      |
| 3             | ThingSpeak channel is active with correct API key. |                                                                                                  |                                   |                           |        |      |
| Test Scenario |                                                    | Simulated crack values are sent from the sensor, processed by ESP32, and uploaded to ThingSpeak. |                                   |                           |        |      |
| ID            | Test Case                                          | Input Data                                                                                       | Expected Result                   | Actual Result             | Status |      |
| 1             | Simulate 6.16 mm crack                             | Crack depth = 6.16 mm                                                                            | Value appears on ThingSpeak chart | As Expected               | Pass   |      |
| 2             | Sensor disconnected                                | No input                                                                                         | ESP32 handles error, no upload    | As Expected               | Pass   |      |

Table 5.10 shows the test case for the integration testing of the web application of the IoT crack detection and monitoring system.

Table 5.10: Integration Test Case for IoT Crack Detection and Monitoring System Web App

|           |    |                     |                                           |
|-----------|----|---------------------|-------------------------------------------|
| <b>ID</b> | 10 | <b>Web App Name</b> | IoT Crack Detection and Monitoring System |
|-----------|----|---------------------|-------------------------------------------|

|                  |                                                                                                                                                                                                                                                         |             |                           |               |      |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|---------------------------|---------------|------|
| <b>Objective</b> | To verify the seamless integration between the Firebase-based web application and the ThingSpeak backend, ensuring that user authentication, navigation, data retrieval, alert display, and system configurations function correctly as a whole system. |             |                           |               |      |
| <b>Name</b>      | Nur Raiyani                                                                                                                                                                                                                                             | <b>Date</b> | 25 <sup>th</sup> May 2025 | <b>Status</b> | Pass |

|                      |                                                                              |                   |                                  |                      |               |
|----------------------|------------------------------------------------------------------------------|-------------------|----------------------------------|----------------------|---------------|
| <b>Test Scenario</b> | User logs in through Firebase Auth, and the dashboard loads ThingSpeak data. |                   |                                  |                      |               |
| <b>ID</b>            | <b>Test Case</b>                                                             | <b>Input Data</b> | <b>Expected Result</b>           | <b>Actual Result</b> | <b>Status</b> |
| 1                    | Valid login                                                                  | Valid credentials | Redirected to dashboard          | As Expected          | Pass          |
| 2                    | Invalid login                                                                | Wrong password    | Error: (auth/invalid-credential) | As Expected          | Pass          |

|                      |                                                                                          |                       |                                      |                      |               |
|----------------------|------------------------------------------------------------------------------------------|-----------------------|--------------------------------------|----------------------|---------------|
| <b>Test Scenario</b> | Dashboard fetches crack width data via ThingSpeak REST API and displays it in real time. |                       |                                      |                      |               |
| <b>ID</b>            | <b>Test Case</b>                                                                         | <b>Input Data</b>     | <b>Expected Result</b>               | <b>Actual Result</b> | <b>Status</b> |
| 1                    | Load dashboard                                                                           | API returns 5.99 mm   | "Current Reading: 5.99 mm" displayed | As Expected          | Pass          |
| 2                    | ThingSpeak API unreachable                                                               | Simulate network loss | Message: "Unable to load data"       | As Expected          | Pass          |

|                      |                                                                                                                               |                                      |                               |                      |               |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------|--------------------------------------|-------------------------------|----------------------|---------------|
| <b>Test Scenario</b> | User configures a new threshold and enables alerts. When sensor data from ThingSpeak exceeds the value, alerts are triggered. |                                      |                               |                      |               |
| <b>ID</b>            | <b>Test Case</b>                                                                                                              | <b>Input Data</b>                    | <b>Expected Result</b>        | <b>Actual Result</b> | <b>Status</b> |
| 1                    | Set threshold to 2.0 mm                                                                                                       | Enter: 2.0, Save                     | Saved to config/used by ESP32 | As Expected          | Pass          |
| 2                    | Trigger email alert                                                                                                           | Threshold = 2.0 mm, Sensor = 6.16 mm | Email sent to user            | As Expected          | Pass          |
| 3                    | Trigger SMS alert                                                                                                             | Threshold = 2.0 mm, Sensor = 6.16 mm | SMS sent to user              | As Expected          | Pass          |
| 4                    | No alert scenario                                                                                                             | Sensor = 0.8 mm                      | No alert triggered            | As Expected          | Pass          |

|                      |                                                                   |                   |                        |                      |               |
|----------------------|-------------------------------------------------------------------|-------------------|------------------------|----------------------|---------------|
| <b>Test Scenario</b> | User visits Data History page and views or filters past readings. |                   |                        |                      |               |
| <b>ID</b>            | <b>Test Case</b>                                                  | <b>Input Data</b> | <b>Expected Result</b> | <b>Actual Result</b> | <b>Status</b> |

|   |                |                |                                                          |             |      |
|---|----------------|----------------|----------------------------------------------------------|-------------|------|
| 1 | Filter by date | "08/05/2025"   | Only records from that date shown                        | As Expected | Pass |
| 2 | No data found  | Empty response | Message: "No data available for the selected date range" | As Expected | Pass |

| Test Scenario |                          | An alert condition is detected by ESP32 and published to ThingSpeak. The UI reacts accordingly. |                                  |               |        |
|---------------|--------------------------|-------------------------------------------------------------------------------------------------|----------------------------------|---------------|--------|
| ID            | Test Case                | Input Data                                                                                      | Expected Result                  | Actual Result | Status |
| 1             | Alert condition active   | "High crack severity detected! Depth: 57.50 mm"                                                 | Alert card shown in red          | As Expected   | Pass   |
| 2             | Moderate alert condition | "Moderate crack severity. Depth: 6.16 mm"                                                       | Yellow colour alert card display | As Expected   | Pass   |
| 3             | No alert condition       | "Crack levels are within safe range. Depth: 1.91 mm"                                            | Green colour alert card display  | As Expected   | Pass   |

| Test Scenario |                                                       | User navigates through all pages and logs out. |                         |               |        |
|---------------|-------------------------------------------------------|------------------------------------------------|-------------------------|---------------|--------|
| ID            | Test Case                                             | Input Data                                     | Expected Result         | Actual Result | Status |
| 1             | Verify navigation from Dashboard to Data History page | Click "Data History" button                    | Data History page opens | As Expected   | Pass   |
| 2             | Verify navigation from Dashboard to Settings page     | Click "Settings" button                        | Settings page opens     | As Expected   | Pass   |
| 3             | Verify navigation from Dashboard to Login page        | Click "Logout" button                          | Login page opens        | As Expected   | Pass   |
| 4             | Verify navigation from Data History to Dashboard page | Click "Dashboard" button                       | Dashboard page opens    | As Expected   | Pass   |
| 5             | Verify navigation from Data History to Settings page  | Click "Settings" button                        | Settings page opens     | As Expected   | Pass   |
| 6             | Verify navigation from Data History to Login page     | Click "Logout" button                          | Login page opens        | As Expected   | Pass   |

|    |                                                      |                             |                         |             |      |
|----|------------------------------------------------------|-----------------------------|-------------------------|-------------|------|
| 7  | Verify navigation from Settings to Dashboard page    | Click “Dashboard” button    | Dashboard page opens    | As Expected | Pass |
| 8  | Verify navigation from Settings to Data History page | Click “Data History” button | Data History page opens | As Expected | Pass |
| 9  | Verify navigation from Settings to Login page        | Click “Logout” button       | Login page opens        | As Expected | Pass |
| 10 | Refresh on protected page (after logout)             | Refresh URL                 | Redirect to login page  | As Expected | Pass |

### 5.3 Non-Functional Testing

Non-functional testing reviews the quality characteristics of a system that are unrelated to specific tasks the system needs to carry out. In functional testing, the goal is to find out what a system does, but in non-functional testing, it is to identify how well the system works under several conditions. For this reason, the reliability, performance and usability of the system are tested to check that it remains user-friendly and efficient throughout its use.

To measure the total system quality of the IoT-Based Crack Detection and Monitoring System, non-functional testing was performed. The tests were mostly done to ensure the system was stable, observe how quickly data could be transmitted, and learn if users were pleased with the interface.

#### 5.3.1 Reliability Testing

The goal of reliability testing is to check if the system can function correctly and regularly for a long period. It guarantees that the system can run regularly without any disruptions, missing data, or behaviour that is not expected. Testing this way is crucial for systems that always operate and handle data in real time, such as those used for monitoring. The purpose of this project was to check the performance of IoT-based crack detection and

monitoring by allowing it to run uninterrupted for a whole day. The purpose was to test if ESP32 could regularly gather sensor readings and transmit them to ThingSpeak and check if Firebase's web interface would consistently bring the latest readings. During the testing period, the system was stable and sent and displayed data as was supposed, showing it would work well for real-time monitoring. Test cases for reliability testing of the IoT-Based Crack Detection and Monitoring System are presented in Table 5.11 to prove the system's reliability by finding and fixing errors that cause it to fail, cease sending data, or present inaccuracies.

Table 5.11: Test Cases for Reliability Testing

|               |                                                                    |                                                                                             |                                                                                     |                           |        |      |
|---------------|--------------------------------------------------------------------|---------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|---------------------------|--------|------|
| ID            | 11                                                                 | Web App Name                                                                                | IoT-Based Crack Detection and Monitoring System                                     |                           |        |      |
| Objective     |                                                                    | To test the reliability and stability of the system under long-term and unstable conditions |                                                                                     |                           |        |      |
| Name          | Nur Raiyani                                                        |                                                                                             | Date                                                                                | 25 <sup>th</sup> May 2025 | Status | Pass |
| No            | Prerequisites                                                      |                                                                                             |                                                                                     |                           |        |      |
| 1             | ESP32 is correctly flashed and connected to Wi-Fi.                 |                                                                                             |                                                                                     |                           |        |      |
| 2             | Sensors are properly wired and calibrated.                         |                                                                                             |                                                                                     |                           |        |      |
| 3             | ThingSpeak channel is configured, and API key is active.           |                                                                                             |                                                                                     |                           |        |      |
| 4             | Firebase interface is connected to ThingSpeak or relevant backend. |                                                                                             |                                                                                     |                           |        |      |
| 5             | Stable power source is used for continuous operation.              |                                                                                             |                                                                                     |                           |        |      |
| Test Scenario |                                                                    | Verify the reliability and stability of the web app under various conditions.               |                                                                                     |                           |        |      |
| ID            | Test Case                                                          | Input Data                                                                                  | Expected Result                                                                     | Actual Result             | Status |      |
| 1             | Continuous Operation Test                                          | Run system for 24 hours with constant sensor data updates                                   | System remains stable, data sent every 15s to ThingSpeak and shown in web interface | As Expected               | Pass   |      |
| 2             | Wi-Fi Disconnection and Reconnection                               | Disconnect Wi-Fi for 1 min, then reconnect                                                  | System resumes data upload without crashing or losing previous data                 | As Expected               | Pass   |      |
| 3             | ThingSpeak Downtime Simulation                                     | Disable ThingSpeak temporarily                                                              | ESP32 handles upload errors gracefully and retries upon reconnection                | As Expected               | Pass   |      |
| 4             | Sensor Disconnection Test                                          | Unplug sensor mid-operation                                                                 | System continues running and logs error or sends                                    | As Expected               | Pass   |      |

|   |                                |                                                       |                                                                       |             |      |
|---|--------------------------------|-------------------------------------------------------|-----------------------------------------------------------------------|-------------|------|
|   |                                |                                                       | null/zero data appropriately                                          |             |      |
| 5 | Power Interruption Test        | Restart ESP32 during operation                        | System reboots and resumes data transmission automatically            | As Expected | Pass |
| 6 | Data integrity test            | Save and retrieve data multiple times from ThingSpeak | All recorded values are intact and display correctly on web interface | As Expected | Pass |
| 7 | API Key Failure                | Use invalid API key                                   | System displays/logs error and prevents incorrect data upload         | As Expected | Pass |
| 8 | High Frequency Data Input Test | Send sensor data every 5s for 30 mins                 | System handles faster data input without lag or crash                 | As Expected | Pass |
| 9 | Browser Access Stress Test     | Open Firebase web app on multiple devices             | Interface loads data without sync issues or performance degradation   | As Expected | Pass |

### 5.3.2 Performance Testing

The goal of performance testing is to find out how efficiently a system handles actual use conditions, especially when important duties like monitoring and alerting take place. When testing this crack detection and monitoring system, the accuracy of detecting different crack depths and the system's response must be checked against the set thresholds. It is important for the system's real-time monitoring applications, since timely alerts may keep structures safe from hazards. The microcontroller should detect any changes in the crack depth, process them immediately, and send correct data to Firebase and ThingSpeak for visualization and triggering alerts. Table 5.12 summarizes the tests done to ensure that the system works properly, consistently, and effectively on different crack inputs.

Table 5.12: Performance Test Cases for Crack Depth Detection Accuracy

|               |                                          |                                                                                                      |                                |                                             |               |        |
|---------------|------------------------------------------|------------------------------------------------------------------------------------------------------|--------------------------------|---------------------------------------------|---------------|--------|
| ID            | 12                                       | Module Name                                                                                          | Crack Depth Detection Accuracy |                                             |               |        |
| Objective     |                                          | To ensure that the system detects crack widths accurately and triggers alerts at correct thresholds. |                                |                                             |               |        |
| Name          | Nur Raiyani                              |                                                                                                      | Date                           | 25 <sup>th</sup> May 2025                   | Status        | Pass   |
| No            | Prerequisites                            |                                                                                                      |                                |                                             |               |        |
| 1             | Sensor calibrated.                       |                                                                                                      |                                |                                             |               |        |
| 2             | Threshold defined.                       |                                                                                                      |                                |                                             |               |        |
| 3             | System connected to Firebase/ThingSpeak. |                                                                                                      |                                |                                             |               |        |
| Test Scenario |                                          | Check if the system identifies crack width correctly and responds based on threshold.                |                                |                                             |               |        |
| ID            | Test Case                                |                                                                                                      | Sensor Input (mm)              | Expected Result                             | Actual Result | Status |
| 1             | Crack width below threshold              |                                                                                                      | 0.8 mm                         | No alert, data stored and displayed         | As Expected   | Pass   |
| 2             | Crack width at threshold                 |                                                                                                      | 2.0 mm                         | Warning active alert displayed on dashboard | As Expected   | Pass   |
| 3             | Crack width above threshold              |                                                                                                      | 6.16 mm                        | Emergency alert sent                        | As Expected   | Pass   |
| 4             | Sensor input fluctuates (noise)          |                                                                                                      | 1.3 – 3.13 mm                  | Warning only when exceeds threshold         | As Expected   | Pass   |
| 5             | Sudden crack width jump                  |                                                                                                      | 2.3 mm                         | Emergency alert sent immediately            | As Expected   | Pass   |
| 6             | Gradual increase over time               |                                                                                                      | 1.2 → 2.4 mm                   | Alert when exceeds threshold                | As Expected   | Pass   |
| 7             | Faulty sensor spike                      |                                                                                                      | 9.0 mm                         | Emergency alert and warning triggered       | As Expected   | Pass   |
| 8             | Sensor stuck                             |                                                                                                      | 5.99 mm                        | Alert once, no duplicates                   | As Expected   | Pass   |
| 9             | No sensor input                          |                                                                                                      | -                              | “No data” and disconnection alert           | As Expected   | Pass   |
| 10            | Matches manual measurement               |                                                                                                      | 0.8 mm                         | Data matches, no alert                      | As Expected   | Pass   |

The results from the performance test showed that the system works well in detecting and dealing with different crack depth situations. All the test cases worked properly, indicating the system can identify and respond to minor, moderate, or severe cracks correctly. Strong measurements were produced, resisting changes in input,

varying crack depths, and damaged or missing information, which helped maintain reliable performance while the system monitored. The measurements taken manually were almost identical to those made by the readings, which confirmed the system's accuracy. It is shown from the results that the system for detecting cracks works equally well and satisfies all the intended performance requirements.

### **5.3.3 Usability Testing**

User Acceptance Testing (UAT) is essential during non-functional testing to ensure the requirements from users and the project are met. During this project, UAT made sure the IoT- based crack detection and monitoring system is effective, well-functioning, and reliable for use by our planned target users. Testing focused on ensuring users could work easily with the web application, see current crack data in real time, and comprehend its key functions of visualizing data, receiving alerts, and checking performance. The testing phase made it possible to find out about problems with usability or limitations caused by technology that were not seen in previous phases.

During UAT, some students and people with basic knowledge about technology were involved in interacting with the system using the web interface and the ThingSpeak cloud. A user manual and sample test cases were used to direct these testers to review sensor values, handle threshold warnings, and access the interface easily. Participant feedback was gathered by distributing the questionnaire through Google Forms. The form featured questions that asked for opinions on using the system, how data was presented, how quickly the system reacts, and what can be improved. The results from the survey given to 15 participants were analysed to detect possible opportunities to enhance the system's user experience before it was deployed.

#### **5.3.3.1 Analysis on User Acceptance Survey**

As part of the UAT, a questionnaire was sent to 15 participants to get their insights on

the system's usability, abilities, performance, and satisfaction. There were four different sections in the questionnaire: Section A covered user background, Section B was about the usability interface, Section C reviewed functionality and performance, and Section D asked about user satisfaction and feedback.

The details about the users in Section A made it easier to see who the participants were. Most respondents (46.7%) were in the 20–29 years age group, whereas 33.3% were in the 30– 39 years group. That left about 20% who were younger than 20 years of age. The group included students as the largest group (53.3%), followed by engineers (26.7%) and architects (20%). Both academics and professionals were well represented in the sample, so the information was balanced. Only 26.7% of respondents reported using IoT-based systems before. Most of those surveyed (73.3%) reported they had not worked with IoT systems before. Therefore, the system must be easy for users who are not familiar with IoT, so a simple design that does not need technological expertise is very important. The detailed responses can be found in **Appendix B**.

### Section B: Usability Interface

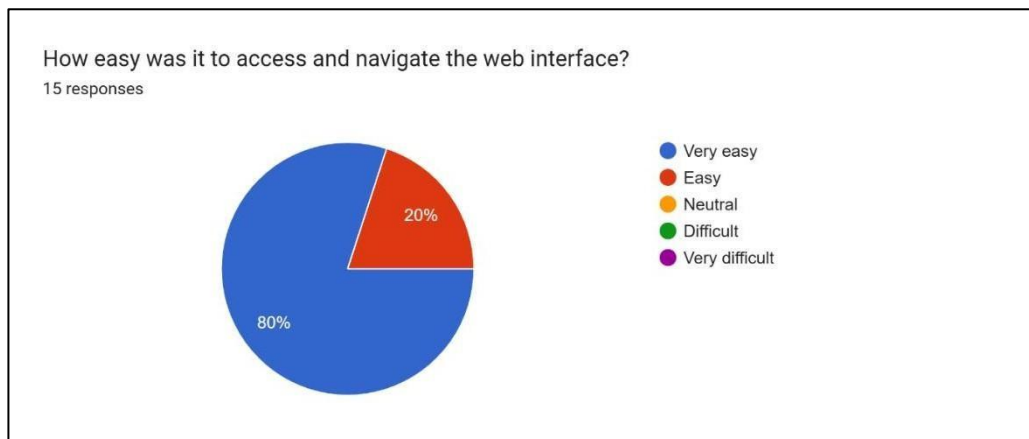


*Figure 5.1: The First Question for Section B*

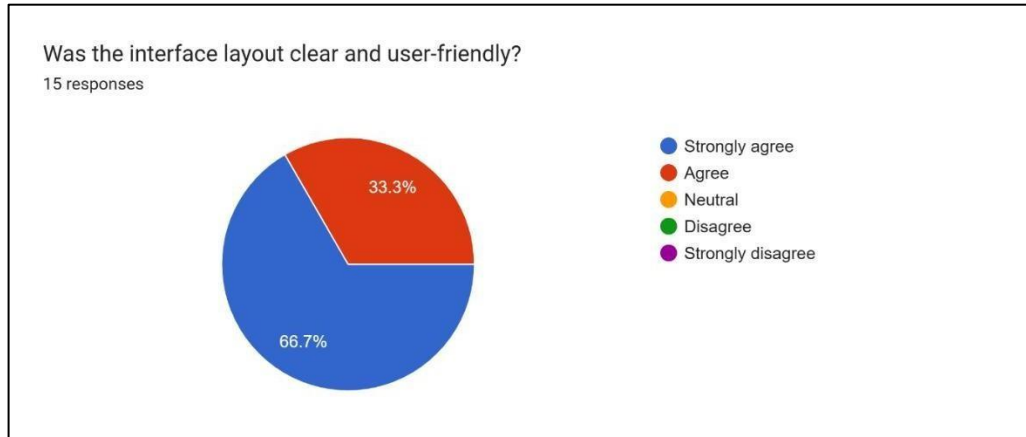


*Figure 5.2: The Second Question for Section B*

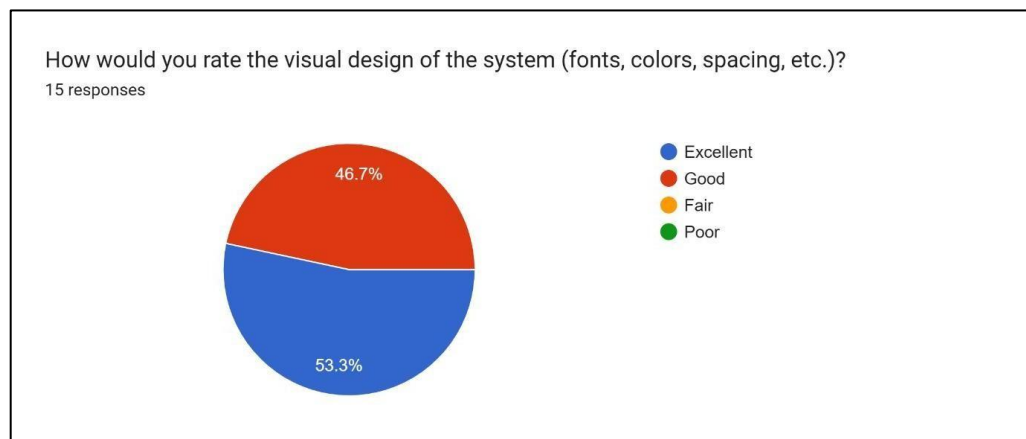


*Figure 5.3: The Third Question for Section B*

Section B was intended to evaluate how easy and convenient the interface of the IoT- Based Crack Detection and Monitoring System’s web platform is to use. From the pie chart shown in Figure 5.1 above, it appears that the majority of participants (80%) indicated that the website was very easy to use, while the other 20% found it easy, which suggests the website is user-friendly for everyone. A large number of respondents strongly agreed that the interface was straightforward and user-friendly, while only a small number said they just agreed, based on Figure 5.2. In relation to the visual design of the system using fonts, colours and spacing, most people (53.3%) described it as “excellent,” and fewer (46.7%) described it as “good,” as shown in Figure 5.3. The feedback shows that people like how the system looks and how easy it is to use, meaning the interface

effectively connects appearance and practicality. All in all, users believe that the web interface is user-friendly, attractive, and organized in a way that supports easy use.

**Section C: Functionality and Performance**

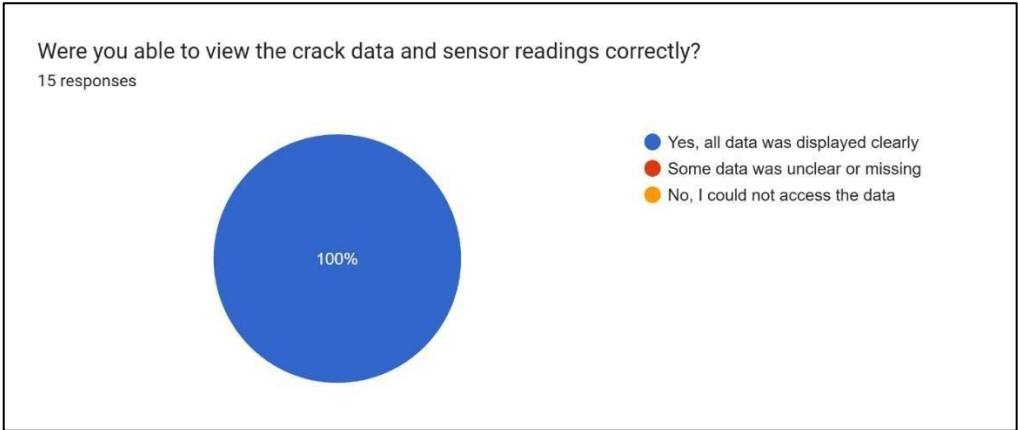


*Figure 5.4: The First Question for Section C\**

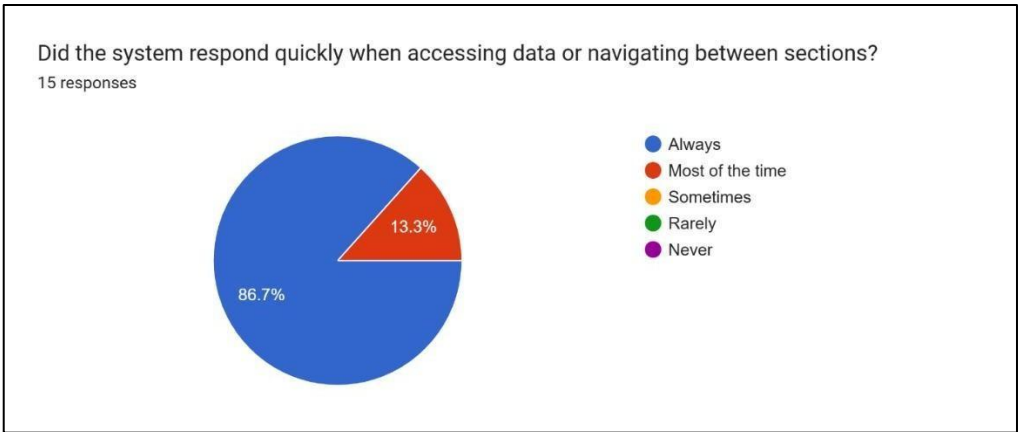


*Figure 5.5: The Second Question for Section C*

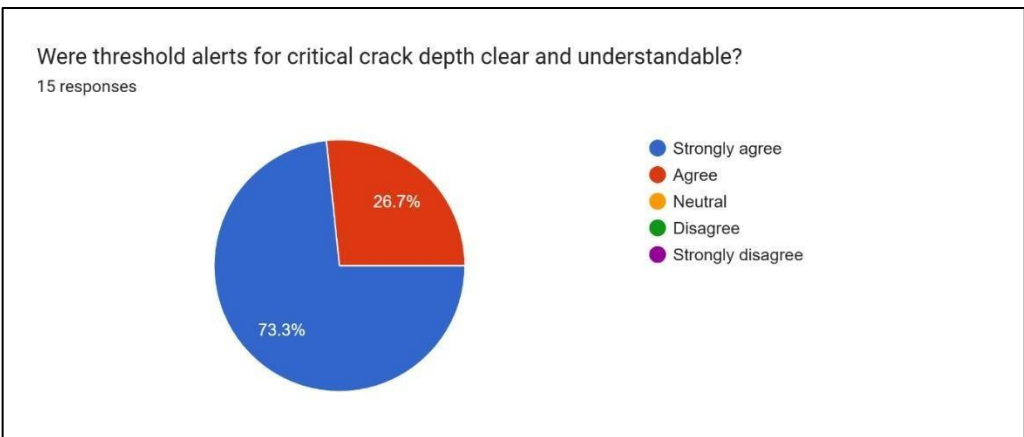


*Figure 5.6: The Third Question for Section C*

In section C, the review focused on whether data was presented accurately, if the system responded quickly, and if the alert messages were easy to understand. Figure 5.4 indicates that all respondents could see the crack data and sensor readings clearly, meaning the system lets users read and access vital information efficiently. Figure 5.5 shows that 86.7% of users said the system always responded fast to their actions, and 13.3% indicated that the system performed well most of the time with little to no lag. According to Figure 5.6, concerning the clarity of threshold alerts that indicate where cracks become critical, 73.3% of users strongly agreed they were clear and understood them well, and the other 26.7% said they agreed to some extent. It means that the system does a good job of alerting users with key information that is easy to use and accessible when needed. All things considered, it is evident that the system operates smoothly, manages information in a timely way, and gives quick alerts, which is important for monitoring buildings in real time.

#### **Section D: User Satisfaction and Feedback**

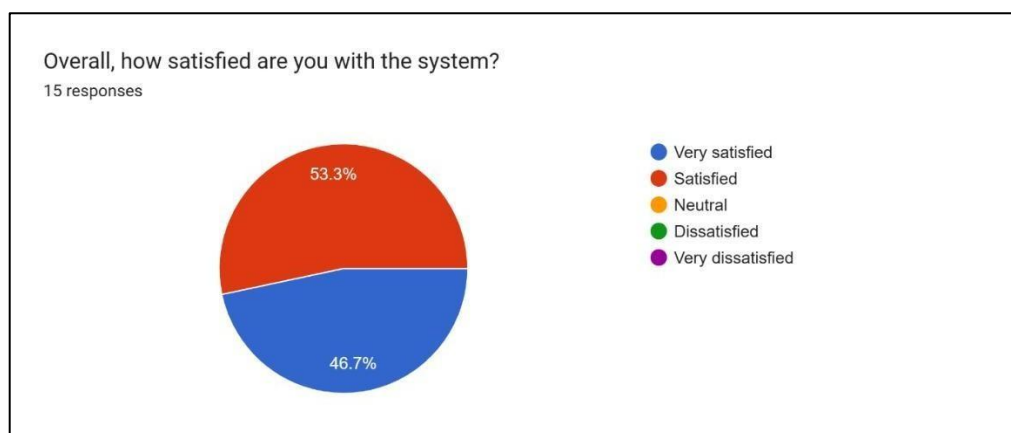


*Figure 5.7: The First Question for Section D*

What did you like most about the system?

15 responses

The system gives alerts when the crack level is serious, which is really helpful.

The design is clean and not too technical, which makes it beginner-friendly.

Very responsive interface, no lagging or crashes during testing.

It's great that the readings are stored and I can refer to past data.

I liked how it shows the date and time of each reading accurately.

Everything is labelled clearly.

Really helpful for engineers like me for our building projects.

The real-time updates are very useful, especially for monitoring without having to go to the site.

It's simple and straightforward. I could easily understand the data being shown.

The system gives alerts when the crack is severe and I can set the threshold myself.

It worked well on my browser without needing to install anything extra.

The real-time updates

Having access to sensor readings anytime gives peace of mind.

The alert system is simple but effective.

I liked that it shows both the graph and current reading in table.

*Figure 5.8: The Second Question for Section D*

What challenges or difficulties did you encounter while using the system?

15 responses

I don't have any issues.

I couldn't tell which readings were the latest unless I looked at the timestamp carefully.

There is no challenges that I encountered.

Nothing.

Okay.

Nope.

No major challenges at all.

Didn't face major issues, just took time to understand the graph.

No challenges and difficulties.

Cannot sort the history data and long list of data in a day make me confused.

All good.

Nothing

No challenges at all.

The readings seemed delayed at times, not sure if it was my internet.

No issues.

*Figure 5.9: The Third Question for Section D*

Do you have any suggestions for improvement or additional features?

15 responses

- A mobile app version would make it easier to check updates on the go.
- Add a zoom feature on the graph to look at specific time periods.
- Adding camera or image to visually capture the crack will be very helpful too.
- Offline viewing of last saved data would be helpful.
- Make the active alerts more visible, maybe like red flashing haha.
- Nothing, I like it already.
- Maybe add a feature to export the data to Excel.
- Would be nice if the system include a feature to export the data to PDF or Excel.
- It would be nice to have a refresh button instead of reloading the page.
- The active alerts part should add checkboxes with it so that we know which one has been settled or still an open case.
- Consider adding camera support to visually capture the crack as well.
- Need bigger graph!
- Include historical trend comparisons, something like "this month vs last month."
- Include an option to add notes or comments for each reading.
- The filter part in history page need to have feature to filter out all the zero reading bcs its a lot.

*Figure 5.10: The Fourth Question for Section D*

In the last section, the focus was on getting user satisfaction ratings and feedback on what works well, what could be better, and what needs fixing. As can be seen in Figure 5.7, many respondents (46.7%) were very satisfied, and another 53.3% were satisfied, giving the system a high level of overall satisfaction. There were several positive qualities mentioned by the respondents, as shown in Figure 5.8. A lot of users appreciated how convenient and easy-to-follow the labels were, along with the well-organized information. Other users enjoyed getting rapid real-time updates, a smooth interface, and instant access to sensor information. In addition, the alert feature was appreciated by a lot of users for warning about significant cracks.

As can be seen in Figure 5.9, most individuals stated there were no major issues, however, a few noted that it was sometimes hard to see the latest information, the graph layout took some getting used to, and there were occasional brief delays in updates that might have been caused by internet quality. A user mentioned that having so many history records with zero readings made it hard to overcome the problem of not being able

to sort or filter them. According to Figure 5.10, the proposed improvements were adding a refresh button, transferring readings to Excel or PDF, and adding a way to include images of cracks using a camera on the web app. Some users suggested adding a mobile app, making active alerts more visible with effects like flashing icons or highlights and providing a way to make notes for each data entry. Some suggested including options for zooming in on graphs, comparing past trends, and looking at data you have saved without an internet connection. The overall results indicated that users were satisfied and suggested some useful ways to make the system better in the future.

#### **5.4 Discussion on Results**

It was confirmed by functional tests, including unit and integration testing, that all IoT-based Crack Detection and Monitoring System parts operated correctly and interacted well. Unit tests found that the sensor readings and processing modules functioned as designed, and integration tests confirmed normal data transmission from the ESP32 to ThingSpeak and Firebase, along with the operation of the alert-generating system. These findings indicate that the important functions of the system involving data collection, transmission, processing, and warning are strong enough to support monitoring cracks in real time.

In addition to functional tests, the non-functional tests proved how the system could perform in real-use situations. In reliability testing, it was found that the system could run continuously for 24 hours, facing network disconnections, problems in the sensors, or loss of power without crashing or losing any information. After performance testing, it was found that detection of crack depth was accurate no matter when or how much the results fluctuated and that it matched the results achieved by manual measurements. This means that the system is consistently accurate in its monitoring, which is essential for important systems where safety matters.

Moreover, usability testing discovered that the system is simple to use and meets users' expectations, including those who are not familiar with IoT. Many respondents stated that the interface was easy to work with, looked good, and responded quickly, which is necessary for successful monitoring and choices while events unfold. Although users were generally satisfied, they wanted to see features such as refreshing, exporting data, and having access on mobile devices included. The insights lead to necessary changes that ensure both the system's technology and user experience are up to standard for it to be widely adopted.

## **5.5 Summary**

This chapter discussed and reviewed the outcomes from carrying out both functional and non- functional tests of the IoT-based crack detection and monitoring system. Both unit and integration testing were used in functional testing to confirm that individual modules and their interactions met the defined requirements. The test cases for the system were satisfied, verifying that data, transmitting that data, and displaying the information functioned appropriately.

The system was analysed during stages of non-functional testing, considering qualities such as reliability, performance, and usability. According to the results of reliability testing, the system kept working steadily for a significant time, and the performance tests showed that the system can accurately check for cracks and rapidly alert about them in multiple situations, showing it is well suited to real-time structural monitoring. A user acceptance survey showed that target users found the system easy to use, quick, and provided helpful features. The results of both evaluations show that the system is working properly, is user-friendly, and can be deployed in the real world.

## CHAPTER 6: CONCLUSION AND FUTURE WORKS

### 6.1 Introduction

The chapter wraps up the development process of the IoT-Based Crack Detection and Monitoring System by using ESP32 to assess structures. It evaluates the entire development of the project, from the first planning and analysis through to implementing and evaluating the system. The aim of this chapter is to evaluate if the project's goals were achieved and to point out the main outcomes and contributions from the development process. Moreover, this chapter addresses the challenges and difficulties that often occur with hardware, software, and environmental elements throughout the implementation phase. This section also delves into future improvements that could enhance the system's functions, scalability, and user- friendliness. This chapter outlines the key value of the project and how it could be used in real-life situations, mainly for enhancing safety and routine maintenance in infrastructure.

### 6.2 Objectives and Achievements

In the beginning of the project, the key objectives were defined in Chapter 1. The objectives led the development and implementation of the IoT-Based Crack Detection and Monitoring System using ESP32. The main aim was to create a wireless system that can track cracks in real time and store data through IoT technology and reasonable components. Table 6.1 below outlines the main objectives and the achievements of the project:

Table 6.1: Summary of Objectives and Achievements

| Objective                                                                                                                                            | Achievement                                                                                                                                                                                                                                                                                                         |
|------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| To design a wireless crack detection system that utilizes sensors to constantly monitor for and detect structural faults, ensuring immediate action. | The system features an ESP32 microcontroller and ultrasonic sensor, and it records the crack depth and wirelessly sends it to ThingSpeak. It allows real-time checking and alerts users when it detects structural problems. Data on the web application is easily accessed through Firebase's basic web interface. |

|                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| To achieve a 90% accuracy rate in tracking and identifying structural cracks in buildings for accurate data and increased safety.                                                               | In experimental circumstances, the sensor was able to give the same correct values each time. Even so, experiments on other materials and in various environments would be needed to ensure that the 90% benchmark could be reached. Yet, the prototype manages to fulfil its purpose for early alerts and real-time monitoring.                                                                        |
| To implement affordable sensors and IoT technology that continuously monitor cracks and provide real-time alerts, which will make it a feasible choice for broad building maintenance adoption. | Components such as ESP32, ultrasonic sensor and software built on open-source IoT platforms are used, which keeps the costs down. These factors enable users to afford, scale, and apply the system in small to medium-sized building maintenance systems. Sending alerts instantly and monitoring them in real-time is in line with the project's goal of improving safety with economical technology. |

### 6.2.1 Discussion on Achievements

The development of the IoT-Based Crack Detection and Monitoring System has shown that it meets most of the standards and is both technically and practically effective in the field of structural health monitoring. The device uses the ESP32 module to communicate, and an ultrasonic sensor collects real-time data on cracks, permitting constant and off-site oversight of buildings. With the successful addition of ThingSpeak and Firebase, the system can make data accessible at any time and any place, which supports the goal of taking quick actions and helping early on.

Furthermore, the test prototype showed positive results when detecting cracks in a simulated setting. When the system continuously measures shortening or lengthening of surfaces within a specified distance, it can identify the possibility of structural issues. This is particularly helpful in the first stages of crack detection because inspections at those times may not be frequent or practical. The use of real-time alerting helps stakeholders react quickly to small faults and prevent these from becoming serious structural issues.

Even though the system was not tested with large data sets due to a lack of large-scale experimentation, it always showed reliable and steady results in different tests.

Fixing issues like setting up sensors, cutting down noise, and making specific material settings could allow the system to move closer to or even surpass the target accuracy. Clear data on the web interface and strong Wi-Fi transmission confirm that the chosen IoT components are effective.

Additionally, it is also significant that the system is built in a cost-effective way. Since the system uses low-cost tools such as the ESP32 and ultrasonic sensors and relies on open-source software, it is a cheaper option than commonly used structural monitoring solutions. Being relatively low-cost helps the system become scalable for different users, such as building managers, maintenance staff, and individual property owners who need to address safety issues in risky locations.

Besides meeting technical needs, the project showed that Agile methodology can be effectively used to handle the development process. Constantly making improvements in each stage of development and testing made sure the project goals were met. All things considered, this project has reached its original goals, giving room for even better improvements, and created a solid base for advancing the technology and applying it in the real world.

### **6.3 Project Limitations and Constraints**

While the IoT-Based Crack Detection and Monitoring System seemed to work effectively during its testing, some problems and limitations were spotted. Even though these limits do not take away from the system's main features, they can improve the product in future versions. For the system to be reliable, scalable, and effective in the real world, it is important to understand these issues.

#### **1. Limited Real-World Testing**

The system was tested under set conditions where temperature, moisture, and different surfaces did not change much. With such a setup, the experiments were easier to manage

and measure, yet they were not designed to simulate all the different conditions met in real buildings or infrastructures. Each type of common material used in construction, such as concrete, steel, wood, and composites, can cause cracks and those cracks may make sensors more or less accurate. Also, when structures are exposed outdoors, they could be hit by wind, rain, sunlight, and vibration, which might influence how the monitoring system functions. Because limited time and resources prevented extensive field studies, the system's ability to perform well in different and unpredictable settings could not be checked.

## 2. Sensor Accuracy and Range Limitations

Using the selected sensor, which is an ultrasonic sensor, allows for easy installation, and it is unlikely in this case to run into major physical involvements, but their design does have some real limits. Problems such as variations in reflections, loud surrounding background noises, temperature changes, and mounting angle issues can lead to their degraded performance. Unevenness in surfaces can cause trouble for distance readings because ultrasonic waves get scattered or absorbed. It becomes difficult to spot cracks developing accurately because tiny changes are hard to note. Besides, the useful range of these sensors could be insufficient to discover serious cracks or large structural flaws in bigger things. The system's accuracy for close monitoring decreases because of these limits.

## 3. Dependence on Internet Connectivity

A main benefit of the system is that it continually sends information to ThingSpeak and Firebase, so anyone can remotely check and supervise the data. Nonetheless, it works well only when the internet is available regularly. In situations where connectivity is weak or unpredictable, such as at construction sites far from town or in remote rural buildings that are old or underground, there is a risk of delays or completely lost data. As a result, the system may fail to deliver warnings on time due to problems with its alerts.

In case of a sudden loss of internet connection, the lack of monitoring can be crucial, certainly if the cracks grow faster during that period.

#### 4. Power Supply Considerations

A stabilized power supply is needed for the ESP32 microcontroller and its connected sensor to continue working. Even though the components are energy-efficient, they still need to be connected to a power source via either a cable or a battery. Maintaining a reliable electricity supply in situations where it is hard to reach or outside can be troublesome. Power failures or low battery can prevent data monitoring, which leads to problem areas in the collected information and a drop in the system's reliability. Since the system depends entirely on being plugged in for power, it cannot be used in necessary critical infrastructure where continuous monitoring matters.

#### 5. Absence of Automated Crack Classification

Although it can spot and monitor cracks using sensors, the system can't determine on its own how serious or what type of cracks they are. Currently, the system can monitor crack depth and surface changes, yet it is unable to tell if a crack is structural or non-structural and whether it is active or inactive. This means that the system does not give a strong diagnosis since users still have to make sense of the readings or ask for expert help to determine a detected anomaly's level of concern.

The integration of machine learning models allows the system to automatically classify events and give early warnings with special descriptive severity tags. Still, creating this feature calls for many resources, including a vast dataset, skilled training models, and modern processing, all of which exceeded the abilities of this project. Given that crack classification is not present, the system cannot easily provide information that would make it easier for building managers, engineers, or emergency services to prioritize actions.

## 6.4 Future Works

Although the IoT-based system for detecting and monitoring cracks has shown its core functions, some enhancements can still be added to make it more accurate, widely usable, and scalable.

### 1. Integration of Automated Crack Classification

Upcoming versions might benefit from including machine learning methods or AI-based classifiers to make the system perform better. Such devices could tell apart structural from non-structural faults, measure their significance, and warn about possible problems ahead. Studying lots of crack variations and corresponding sensor readings helps the system give reliable results and reduce the need for manual checking.

### 2. Mobile App Development

Having a mobile application would make it possible for users to get alerts in real time, review dashboards for monitoring cracks, and control data using their smartphones. A mobile app could make it easier for managers or maintenance workers to solve emergency situations.

### 3. Multi-Sensor Support for Broader Monitoring

Currently, the system mostly depends on monitoring with just one sensor. More sensors could be used in future versions in different zones of the structure, which would allow for a full check of all parts of the building or bridge. With this approach, users could observe and monitor several cracks and see the development of cracks in the material.

### 4. Offline Data Logging and Backup

Offline storing of data could help the system in places where the internet is not always available or is slow. With this arrangement, the system will remain active and will store data in the cloud when the connection is restored. Adding backups locally will ensure historical data is safe if the server fails or data is lost.

## 5. Battery Optimization and Power Management

Improving power efficiency will result in less maintenance and longer life for the machines. Some options are to tune the sensor operating times, use power-saving wireless technologies such as LoRa or Zigbee, and connect the devices using solar-powered solutions.

## 6. User Role Management and Multi-Level Access

It would be useful to incorporate a safe login system with varying user roles such as admin, maintenance team, and engineers so that everyone gets the right access. This option can handle needed permissions and shield critical data across big, complex systems such as the ones used by businesses and governments.

## 7. Integration with Predictive Maintenance System

If the system combines crack data with inputs from vibration, temperature, and load, it could become a part of a comprehensive predictive maintenance framework. It would allow for identifying upcoming failure issues and scheduling fixes in advance, promoting safety and saving money.

## 8. Support for Visual Data (Camera or Image Processing)

Using a camera system to confirm the presence of cracks together with the sensor data is useful. Regular image capture or snapshots upon a change in readings allow users to check the condition of the building from a distance. Higher versions may employ computer vision tasks to make crack detection more accurate. Improving these features will guarantee the system is more dependable, can be used in a wider range of situations, and becomes a smarter device for monitoring the condition of buildings at any moment.

## 6.5 Conclusion

All in all, it is clear from the results that the IoT-Based Crack Detection and Monitoring System can successfully monitor structural cracks in real time by using reasonably priced

technology. Nevertheless, several changes are necessary to turn the system into a dependable solution for real-world situations. Future efforts should aim to make the system more accurate, add automated classification for cracks, enhance sensor performance, and improve both the system's scalability and user interface. With these enhancements, the system will overcome present problems and become stronger, more accessible, and more effective. These advancements eventually help by making it possible to locate issues in infrastructure at an early stage and keep structures well maintained.

## REFERENCES

- Abdul Razak, A. H., Abdullah, N. S., Al Junid, S. A., Halim, A. K., Idros, M. F., Osman, F. N., & Nazamuddin, F. (2022a). Structural crack detection system using internet of things (IOT) for Structural Health Monitoring (SHM): A Review. *Jurnal Kejuruteraan*, 34(6), 983–998. [https://doi.org/10.17576/jkukm-2022-34\(6\)-01](https://doi.org/10.17576/jkukm-2022-34(6)-01)
- Babu, J. C., Kumar, M. S., Jayagopal, P., Sathishkumar, V. E., Rajendran, S., Kumar, S., Karthick, A., & Mahseena, A. M. (2022). IOT-based intelligent system for internal crack detection in building blocks. *Journal of Nanomaterials*, 2022(1). <https://doi.org/10.1155/2022/3947760>
- Butler, R. (2022, March 7). *Benefits of IOT-enabled predictive maintenance for buildings*. Leading adaptive buildings company. <https://www.buildingsiot.com/blog/benefits-of-iot-enabled-predictive-maintenance-for-buildings-bd>
- Cantero-Chinchilla, S., Aranguren, G., Royo, J. M., Chiachío, M., Etxaniz, J., & Calvo-Echenique, A. (2021). Structural health monitoring using ultrasonic guided-waves and the degree of health index. *Sensors*, 21(3), 993. <https://doi.org/10.3390/s21030993>
- Golding, V. P., Gharineiat, Z., Munawar, H. S., & Ullah, F. (2022). Crack detection in concrete structures using Deep Learning. *Sustainability*, 14(13), 8117. <https://doi.org/10.3390/su14138117>
- Jung, Y.-J., & Jang, S.-H. (2024). Crack detection of reinforced concrete structure using smart skin. *Nanomaterials*, 14(7), 632. <https://doi.org/10.3390/nano14070632>

- Khan, Md. A.-M., Kee, S.-H., Pathan, A.-S. K., & Nahid, A.-A. (2023). Image processing techniques for Concrete Crack Detection: A Scientometrics Literature Review. *Remote Sensing*, 15(9), 2400. <https://doi.org/10.3390/rs15092400>
- Li, J., Bennet, C., & Collins, W. (2019). *Strain-Based Fatigue Crack Monitoring of Steel Bridges Using Wireless Elastomeric Skin Sensors*.
- Liu, H., Laflamme, S., Li, J., Downey, A., Bennett, C., Collins, W., Ziehl, P., Jo, H., & Todsen, M. (2024). Sensing Skin Technology for fatigue crack monitoring of Steel Bridges: Laboratory Development, field validation, and Future Directions. *International Journal of Bridge Engineering, Management and Research*, 1(1). <https://doi.org/10.70465/ber.v1i1.8>
- Mathé, C. (2023, December 12). *IOT in smart buildings: Benefits, use cases, and tips*. Wattsense. <https://www.wattsense.com/blog/building-management/iot-in-smart-buildings-benefits-use-cases-and-tips/>
- Rahita, A. C., Zaki, A., Nugroho, G., & Yadi, S. (2024). Internet of things (IOT) in Structural Health Monitoring: A decade of research trends. *Instrumentation Measure Métrologie*, 23(2), 123–139. <https://doi.org/10.18280/i2m.230205>
- Revolutionizing structural health monitoring with AI, ML, IOT, and Sensor Technology*. Encardio Rite. (2024). <https://www.encardio.com/blog/ai-ml-structural-health-monitoring>
- Sarkom, Y., Abdul Ghani, M. Z., Muhamad Ariff, N. R., Anuar Jalil, M. K., & Kenawy, I. (2023). A comparative review of the statutory periodic building inspection implementation in the Klang Valley Region, Malaysia and Hong Kong, China.

*International Journal of Sustainable Construction Engineering and Technology*,  
14(5). <https://doi.org/10.30880/ijscet.2023.14.05.022>

Simpson, D. (2024, October 10). 5 ways hardware development is just... different. Altium.  
<https://resources.altium365.com/p/why-hardware-development-is-different>

Sun, C., & Zhang, Z. (2020). *An Automatic Deep Learning-Based Crack Identification Methodology for Bridges Using UAV Images*.

Tanveer, M., Elahi, M. U., Jung, J., Azad, M. M., Khalid, S., & Kim, H. S. (2024). Recent advancements in guided ultrasonic waves for structural health monitoring of composite structures. *Applied Sciences*, 14(23), 11091.  
<https://doi.org/10.3390/app142311091>

*The role of IOT technology in Optimizing Facilities Management: Features and benefits across buildings*. Tanand Technology. (2024, June 16).  
<https://www.tanand.com.my/the-role-of-iot-technology-in-optimizing-facilities-management-features-and-benefits-across-buildings/>

Yoon, J., Lee, J., Kim, G., Ryu, S., & Park, J. (2022). Deep neural network-based structural health monitoring technique for real-time crack detection and localization using strain gauge sensors. *Scientific Reports*, 12(1). <https://doi.org/10.1038/s41598-022-24269-4>

## APPENDICES

### Appendix A

#### Sample Questionnaire

Link of the Survey Form: <https://forms.gle/CmEdhjpLo5WYnUZW8>

#### Final Year Project: Survey on IoT-based Crack Detection and Monitoring System

Assalamualaikum and hi! My name is Nur Raiyani, and I am a fourth-year student in the Network Computing course at Universiti Malaysia Sarawak (UNIMAS). As part of my final year project (FYP), I am working on a project titled **"IoT-Based Crack Detection and Monitoring System."** This project aims to develop a wireless, real-time crack detection and monitoring system that uses IoT technology to help identify and assess structural cracks, providing early alerts for maintenance and ensuring building safety.

To gather valuable insights for this project, I am conducting this survey to understand the challenges, benefits, and perceptions of current crack detection systems, as well as evaluate how an IoT-based system could improve existing processes.

Your participation in this survey is crucial for the success of my project, as it will help me understand the needs and expectations of potential users. The survey should take only a few minutes to complete, and your responses will be kept confidential.

If you have any questions or need further information, feel free to contact me at:  
Email: [80685@siswa.unimas.my](mailto:80685@siswa.unimas.my)  
Phone: 01119390387

Thank you for your time and support!

#### Section A: Respondent Information

1. What is your age group? \*

- ☐ Below 20 years
- ☐ 20–29 years
- ☐ 30–39 years
- ☐ 40–49 years
- ☐ 50 years and above

2. What is your gender? \*

- ☐ Male
- ☐ Female
- ☐ Prefer not to say

3. What is your occupation? \*

- ☐ Student
- ☐ Engineer
- ☐ Architect
- ☐ Other: \_\_\_\_\_

### Section B: Familiarity with Crack Detection Systems

1. Are you familiar with crack detection and monitoring systems? \*

- ☐ Yes, I have used them before
- ☐ Yes, but I have never used them
- ☐ No, I am not familiar with them

2. What types of crack detection systems have you used before? (Select all that apply) \*

- ☐ Manual visual inspections
- ☐ Non-destructive testing methods (e.g., ultrasonic, radiographic)
- ☐ Automated systems (e.g., sensors, monitoring devices)
- ☐ None

3. How satisfied are you with the existing crack detection systems? \*

- ☐ Very satisfied
- ☐ Satisfied
- ☐ Neutral
- ☐ Dissatisfied
- ☐ Very dissatisfied

4. What are the biggest challenges you face with existing systems? (Select all that apply) \*

- ☐ High cost
- ☐ Time-consuming processes
- ☐ Lack of real-time monitoring
- ☐ Limited accessibility
- ☐ Complexity of operation
- ☐ Other: \_\_\_\_\_

5. How often do you think cracks in structures go unnoticed or undetected? \*

- ☐ Very often
- ☐ Often
- ☐ Sometimes
- ☐ Rarely
- ☐ Never

### Section C: Proposed IoT-Based Crack Detection and Monitoring System

1. How important is it to have real-time crack monitoring for structural safety? \*

- ☐ Very important
- ☐ Important
- ☐ Neutral
- ☐ Not important

2. Do you think IoT-based crack detection systems could address challenges faced by current systems? \*

- ☐ Strongly agree
- ☐ Agree
- ☐ Neutral
- ☐ Disagree
- ☐ Strongly disagree

3. What features would you find most useful in an IoT-based crack detection system? (Select all that apply) \*

- ☐ Real-time monitoring and alerts
- ☐ Historical data for analysis
- ☐ Remote accessibility
- ☐ Cost-effectiveness
- ☐ Scalability to different types of structures
- ☐ Other: \_\_\_\_\_

4. What concerns do you have about using an IoT-based crack detection system? (Select all that apply) \*

- ☐ Accuracy and reliability of detection
- ☐ High cost
- ☐ Maintenance requirements
- ☐ Complexity of installation and use
- ☐ Dependence on internet connectivity
- ☐ Other: \_\_\_\_\_

5. How do you think the proposed IoT-based crack detection system compares to existing systems in terms of overall effectiveness? \*

- ☐ More effective
- ☐ About the same
- ☐ Less effective

6. Would you consider using an IoT-based crack detection and monitoring system if it proves to be more accurate and effective than existing systems? \*

- ☐ Yes
- ☐ No
- ☐ Maybe

## Appendix B

### Sample Questionnaire

Link of the Survey Form: <https://forms.gle/Rb7VxEkie13qToZ9A>

#### User Acceptance Testing (UAT) for IoT-Based Crack Detection and Monitoring System

Assalamualaikum and hi! I am Nur Raiyani, a final-year student of the Network Computing program at Universiti Malaysia Sarawak (UNIMAS). As part of my Final Year Project (FYP), I have developed an "IoT-Based Crack Detection and Monitoring System" aimed at monitoring structural cracks in real-time using wireless sensor technology. This system is designed to assist in early detection of cracks for preventive maintenance, helping to enhance safety in buildings and infrastructure.

To ensure the system is functional, user-friendly, and meets its intended goals, I am conducting this **User Acceptance Testing (UAT)** to gather feedback from potential users. This feedback is essential in evaluating the usability, performance, and satisfaction level of the system from a user's perspective. Your honest responses will help me improve the system and validate its effectiveness before final implementation.

The UAT questionnaire will only take a few minutes of your time, and all responses will be kept strictly confidential. If you have any questions or concerns regarding the testing or this form, please feel free to contact me:

Email: [80685@siswa.unimas.my](mailto:80685@siswa.unimas.my)  
Phone: 011-1939 0387

##### Section A: User Background

What is your age group? \*

- ☐ Below 20 years
- ☐ 20-29 years
- ☐ 30-39 years
- ☐ 40-49 years
- ☐ 50 years and above

What is your occupation? \*

- ☐ Student
- ☐ Engineer
- ☐ Architect
- ☐ Other: \_\_\_\_\_

Have you used any IoT-based system before? \*

- ☐ Yes
- ☐ No

##### Section B: Usability and Interface

How easy was it to access and navigate the web interface? \*

- ☐ Very easy
- ☐ Easy
- ☐ Neutral
- ☐ Difficult
- ☐ Very difficult

Was the interface layout clear and user-friendly? \*

- ☐ Strongly agree
- ☐ Agree
- ☐ Neutral
- ☐ Disagree
- ☐ Strongly disagree

How would you rate the visual design of the system (fonts, colors, spacing, etc.)? \*

- ☐ Excellent
- ☐ Good
- ☐ Fair
- ☐ Poor

| Section C: Functionality and Performance                                                                                                                                                                                                                                                    | Section D: User Satisfaction and Feedback                                                                                                                                                                                                                                 |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Were you able to view the crack data and sensor readings correctly? *</p> <p><input type="radio"/> Yes, all data was displayed clearly</p> <p><input type="radio"/> Some data was unclear or missing</p> <p><input type="radio"/> No, I could not access the data</p>                    | <p>Overall, how satisfied are you with the system? *</p> <p><input type="radio"/> Very satisfied</p> <p><input type="radio"/> Satisfied</p> <p><input type="radio"/> Neutral</p> <p><input type="radio"/> Dissatisfied</p> <p><input type="radio"/> Very dissatisfied</p> |
| <p>Did the system respond quickly when accessing data or navigating between sections? *</p> <p><input type="radio"/> Always</p> <p><input type="radio"/> Most of the time</p> <p><input type="radio"/> Sometimes</p> <p><input type="radio"/> Rarely</p> <p><input type="radio"/> Never</p> | <p>What did you like most about the system? *</p> <p>Your answer</p>                                                                                                                                                                                                      |
| <p>Were threshold alerts for critical crack depth clear and understandable? *</p> <p><input type="radio"/> Strongly agree</p> <p><input type="radio"/> Agree</p> <p><input type="radio"/> Neutral</p> <p><input type="radio"/> Disagree</p> <p><input type="radio"/> Strongly disagree</p>  | <p>What challenges or difficulties did you encounter while using the system? *</p> <p>Your answer</p>                                                                                                                                                                     |
|                                                                                                                                                                                                                                                                                             | <p>Do you have any suggestions for improvement or additional features? *</p> <p>Your answer</p>                                                                                                                                                                           |
|                                                                                                                                                                                                                                                                                             | <div><div>Back</div><div>Submit</div><div>Clear form</div></div>                                                                                                                                                                                                          |