

**TECHNISCHE HOCHSCHULE OSTWESTFALEN-LIPPE**

**UNIVERSITY OF APPLIED SCIENCES AND ARTS DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE**

**INDUSTRIAL SOFTWARE ENGINEERING**

**AUTOMATED AQAURIUM USING IOT**

**Submitted By:**

Vamsi Kumar Yerasani-20108353

Gowtham Reddy Sodanapalli-20102603

Sai Kalyani Yerrasani-20069528

Preeti Sharma-20108340

**Table of Contents**

1. **Introduction**  
    1.1 Background and Motivation  
    1.2 Objectives  
    1.3 Tools and Technologies Used
2. **Business Process Design**  
    2.1 Domain Selection   
    2.2 Textual Description of Business Process  
    2.3 BPMN Diagram
3. **Use Cases and User Stories**  
    3.1 Identified Use Cases  
     3.1.1 Use Case 1: Monitor Water Parameters  
     3.1.2 Use Case 2: Feed Fish Automatically  
    3.2 Use Case Specification  
    3.3 User Stories  
     3.3.1 Story 1   
     3.3.2 Story 2   
     3.3.3 Story 3
4. **Domain Modelling**  
    4.1 Manually Designed Domain Model  
    4.2 Prompts and Outputs from GPT-4  
    4.3 Comparison of Manual vs AI Domain Models
5. **System Architecture**  
    5.1 Overview of System Components  
     5.1.1 Hardware Architecture  
     5.1.2 Software Architecture  
    5.2 IoT Patterns Used  
    5.3 Design Decisions
6. **Prototype Implementation**  
    6.1 Use Cases Demonstrated  
    6.2 Tools and Technologies  
    6.3 GitHub Repository and Demo Access  
    6.4 Design and Implementation Overview
7. **Software Development Process**  
    7.1 Chosen Process Model   
    7.2 Justification and Adaptation
8. **AI Support Documentation**  
    8.1 Steps Supported by AI  
    8.2 AI Tools Used  
    8.3 Prompts and Outputs  
    8.4 Evaluation of AI Usefulness and Limitations
9. **Conclusion**
10. **References**
11. **Appendix**

# 1.Introduction

**1.1 Background and Motivation**

Maintaining an aquarium is a delicate and time-intensive task that involves regular monitoring of water parameters, scheduled feeding, and consistent lighting. Failure to perform these tasks accurately can harm aquatic life. With the rise of IoT (Internet of Things) technologies, there is an opportunity to automate and optimize these tasks using sensors, controllers, and cloud-based systems. This project explores how IoT can be applied to simplify aquarium maintenance, reduce human intervention, and enhance the overall reliability of the system.

**1.2 Objectives**

* To design and develop an IoT-based automated aquarium system.
* To monitor water parameters (temperature and pH) in real time.
* To automate fish feeding and lighting based on predefined schedules or user input.
* To notify users in case of abnormal conditions and enable manual override via a mobile application.
* To apply structured software engineering methods for analysis, design, and development.
* To integrate AI tools in modelling and documentation processes.

**1.3 Tools and Technologies Used**

* ESP32 – Central IoT controller with built-in Wi-Fi.
* Temperature and pH Sensors – To monitor water quality.
* Servo Motor – For automatic fish feeding.
* LED Lighting Unit – For lighting control.
* Power Supply – To power the system.
* Arduino IDE (C++) – For microcontroller programming.
* MQTT Protocol – For lightweight real-time communication.
* Firebase / Thing Speak – For cloud data logging and dashboard visualization.
* Mobile Application – For user interaction and alerts.

# 2. Business Process Design

# 2.1 Domain Selection

The chosen domain for this project is **Smart Aquarium Automation**, a subdomain of the Internet of Things (IoT). This domain focuses on applying connected devices and automation technologies to monitor and manage aquariums more efficiently. It involves integrating sensors, actuators, cloud communication, and mobile interfaces to automate tasks such as water quality monitoring, fish feeding, and lighting control. The project aligns with the goals of IoT by enabling real-time data collection, intelligent decision-making, and remote user interaction within a physical environment.

**2.2 Textual Description of Business Process**

The business process for the automated aquarium system is designed around three key actors: User, Sensor System, and Controller. The process begins when the user activates the aquarium system. From that point, the sensor system continuously monitors essential parameters such as temperature and pH levels. If these parameters remain within acceptable ranges, the system continues normal operation by managing lighting and feeding based on set schedules.

If abnormal water conditions are detected, the system triggers an alert. The user is notified via the mobile app, where they can either manually override the automation or allow the system to attempt auto-correction (if programmed). The process includes parallel actions (e.g., lighting and feeding), exclusive decisions (e.g., normal vs. abnormal condition detection), and user intervention points, representing a realistic and dynamic IoT-based control flow.

**2.3 Business Process Model Diagram**

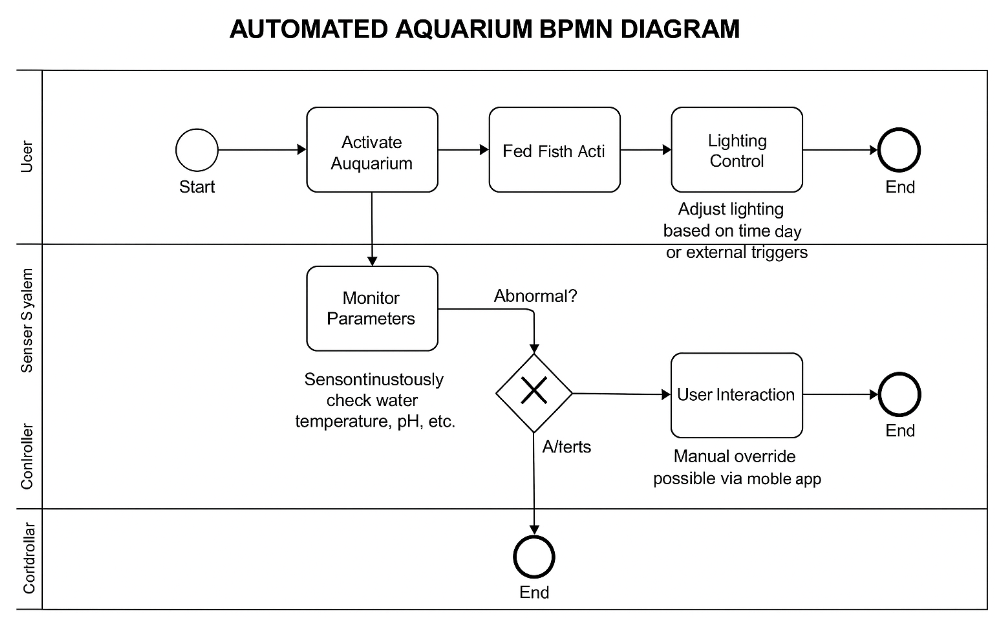
****

Fig. 1: Business Process Model

# 2. Use Cases and User Stories

**3.1 Identified Use Cases**

**Use Case 1:** Monitor Water Parameters

Goal: Continuously monitor the water's temperature and pH level and notify the user if values are abnormal.

**Use Case 2:** Feed Fish Automatically

Goal: Automatically dispense food to the fish at scheduled intervals without manual input.

**3.1.1 Use Case 1: Monitor Water Parameters**

**Description**:  
This use case enables the automated aquarium system to continuously monitor critical water parameters such as temperature and pH level using sensors. The system checks whether the values are within safe thresholds and alerts the user via a mobile application if abnormalities are detected.

**Primary Actor**:  
Sensor System

**Secondary Actor**:  
User

**Goal**:  
To ensure the aquatic environment remains safe and stable by monitoring conditions in real time and notifying the user when attention is needed.

**3.1.2 Use Case 2: Feed Fish Automatically**

**Description**:  
This use case allows the system to **automatically dispense food** to the fish at scheduled intervals using a servo motor-controlled feeding mechanism. The feeding schedule is predefined and can optionally be adjusted by the user.

**Primary Actor**:  
Controller (Feeding Mechanism)

**Secondary Actor**:  
User

**Goal**:  
To maintain a consistent feeding routine for the fish without requiring manual input from the user.

**3.2 Use Case Specification**

**Use Case: Monitor Water Parameters**

|  |  |
| --- | --- |
| **Field** | **Details** |
| Use Case ID | UC1 |
| Actors | Sensor System (Primary), User (Secondary) |
| Goal | Monitor water temperature and pH; alert user if abnormal |
| Preconditions | System is powered on; sensors are active |
| Postconditions | Data is logged; alert sent if needed |
| Main Flow | 1. Read sensor data  2. Check thresholds  3. Send alert if abnormal |
| Alternative | If values are normal, continue monitoring |
| Exceptions | Sensor failure → send error alert |
| Frequency | Continuous |

**Use Case: Feed Fish Automatically**

|  |  |
| --- | --- |
| **Field** | **Details** |
| Use Case ID | UC2 |
| Actors | Controller (Primary), User (Secondary) |
| Goal | Automatically feed fish at scheduled times |
| Preconditions | System is on; feeding schedule is set |
| Postconditions | Fish are fed; action is logged |
| Main Flow | 1. Controller checks schedule  2. Activates feeder at set time  3. Logs action |
| Alternative | Manual feeding if override is triggered |
| Exceptions | Feeder jam/failure → alert sent to user |
| Frequency | Multiple times daily, based on schedule |

**3.3 User Stories (Based on INVEST Principle)**

**User Story 1: Monitor Water Parameters**

Users want the system to continuously monitor the aquarium’s temperature and pH levels,  
So that I can ensure my fish live in a healthy environment.

* **Independent:** Doesn’t rely on other stories.
* **Negotiable**: Threshold values or additional sensors can be added later.
* **Valuable**: Protects aquatic life through early alerts.
* **Estimable**: Easy to estimate effort for implementation.
* **Small**: Focused only on monitoring and alerting.
* **Testable**: Verify with simulated sensor data.

**User Story 2: Feed Fish Automatically**

Users want the system to feed the fish at predefined intervals, So that user doesn’t have to feed them manually every day.

* **Independent**: Can work without other features like alerts.
* **Negotiable**: Feeding frequency can be changed.
* **Valuable**: Ensures consistent feeding schedule.
* **Estimable**: Clear scope and implementation steps.
* **Small**: Only covers automated feeding.
* **Testable**: Check servo motor action on schedule.

**User Story 3: Manual Override via App**

Users want to manually control feeding and lighting via the mobile app,  
So that user can act during unexpected situations.

* **Independent**: Works even if automation fails.
* **Negotiable**: Can extend to other functions later.
* **Valuable**: Gives users control and confidence.
* **Estimable**: Simple to scope for app integration.
* **Small**: Focused only on manual controls.
* **Testable**: Verify manual trigger functionality in app.

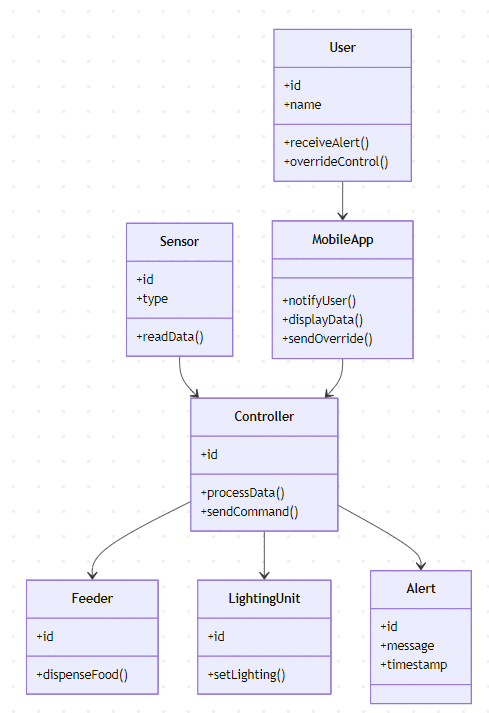
**4. Domain Modelling**

**4.1 Manually Designed Domain Model**

The domain model represents the key entities and their relationships in the system. It was designed based on the identified use cases and user stories (monitoring water parameters, feeding fish automatically, and manual override via app).

**Key Entities:**

* **User**: Interacts with the system via mobile app.
* **Sensor**: Captures water parameters (temperature, pH).
* **Controller**: Core logic component (e.g., ESP32).
* **Feeder**: Dispenses food on schedule or by command.
* **Lighting Unit**: Controls aquarium lighting.
* **Alert**: Notifies user in case of abnormal conditions.
* **Mobile App**: Interface for alerts and manual control.



**4.2 AI-Generated Domain Model (LLM + Mermaid)**

To support the modelling process, we used an AI tool (ChatGPT) with a prompt-based approach to generate an alternative domain model using Mermaid syntax.

**Prompt Used:**

*"Generate a domain model for an IoT-based Automated Aquarium system. It should include entities related to sensors, controller, feeder, lighting, alerts, and user interactions. Use Mermaid syntax to represent the class diagram. Base it on the following use cases: Monitor Water Parameters, Feed Fish Automatically, Notify User if water conditions are abnormal. Follow standard UML class diagram format using Mermaid."*

**4.3 Comparison: Manual vs. AI-Generated Models**

| Aspect | Manual Model | AI-Generated Model |
| --- | --- | --- |
| Clarity | Focused on project-specific design | More generalized IoT structure |
| Detail Level | Custom details like ESP32, MQTT, Firebase included | Abstract components and standard class names |
| Accuracy | Tailored to actual implementation | Correct structure, needs refinement for real-world mapping |
| Usefulness | Ideal for implementation | Useful for quick scaffolding and brainstorming |
| Limitations | Requires more time and tooling | Needs validation and manual editing for relevance |

**5. System Architecture**

**5.1 Overview of System Components**

The automated aquarium system is built using a combination of hardware devices and software services, integrated through an IoT architecture. Each component plays a specific role in monitoring, decision-making, control, and communication.

**5.1.1 Hardware Components**

| Component | Description |
| --- | --- |
| ESP32 Microcontroller | The central controller that connects to all sensors and actuators. Equipped with built-in Wi-Fi, it processes sensor data, executes control logic, and communicates with the cloud via MQTT. |
| Temperature Sensor | Measures the current water temperature. Data is sent to the ESP32 for evaluation. |
| pH Sensor | Continuously monitors the water's pH level, an important parameter for fish health. |
| Servo Motor (Feeder) | Mechanically dispenses fish food at scheduled intervals or on user command. |
| LED Light Unit | Controls aquarium lighting. Can be turned on/off based on time, ambient conditions, or manual override. |
| Power Supply Unit | Provides regulated power to the ESP32, sensors, and actuators. Ensures safe and stable operation. |

**5.1.2 Software Components**

| Component | Description |
| --- | --- |
| Arduino IDE | Used to write and upload the control firmware to the ESP32. The logic for monitoring, scheduling, and actions is coded in C++. |
| MQTT Protocol | A lightweight messaging protocol used for communication between ESP32 and cloud services. Enables publishing sensor data and subscribing to user commands. |
| Firebase / ThingSpeak | Cloud-based database and visualization platform. Used for storing historical sensor data, logging system actions, and optionally presenting a dashboard for the user. |
| Mobile Application / Dashboard | A custom responsive web application built with **Next.js and React**. It serves as both a full-featured desktop dashboard and a mobile app for remote monitoring and control. |

**5.2 IoT Patterns Used**

The architecture applies several IoT design patterns described in the article *"Applying IoT Patterns to Smart Factory Systems”*:

| Pattern | Implementation in Project |
| --- | --- |
| Device Gateway | ESP32 acts as a gateway between sensors/actuators and cloud/database |
| Event Stream | Sensor data is published via MQTT in real time |
| Action Executor | Servo motor and LED lights are triggered based on data or commands |

**5.3 Design Decisions**

* **ESP32** was chosen for its built-in Wi-Fi and low power consumption.
* **MQTT** was used for its lightweight protocol, ideal for IoT applications.
* The system supports **manual override** through a mobile app for flexibility.
* A **modular design** was implemented to allow easy debugging and future enhancements
* **Cloud storage** (Firebase or Thing Speak) ensures data persistence and remote accessibility.

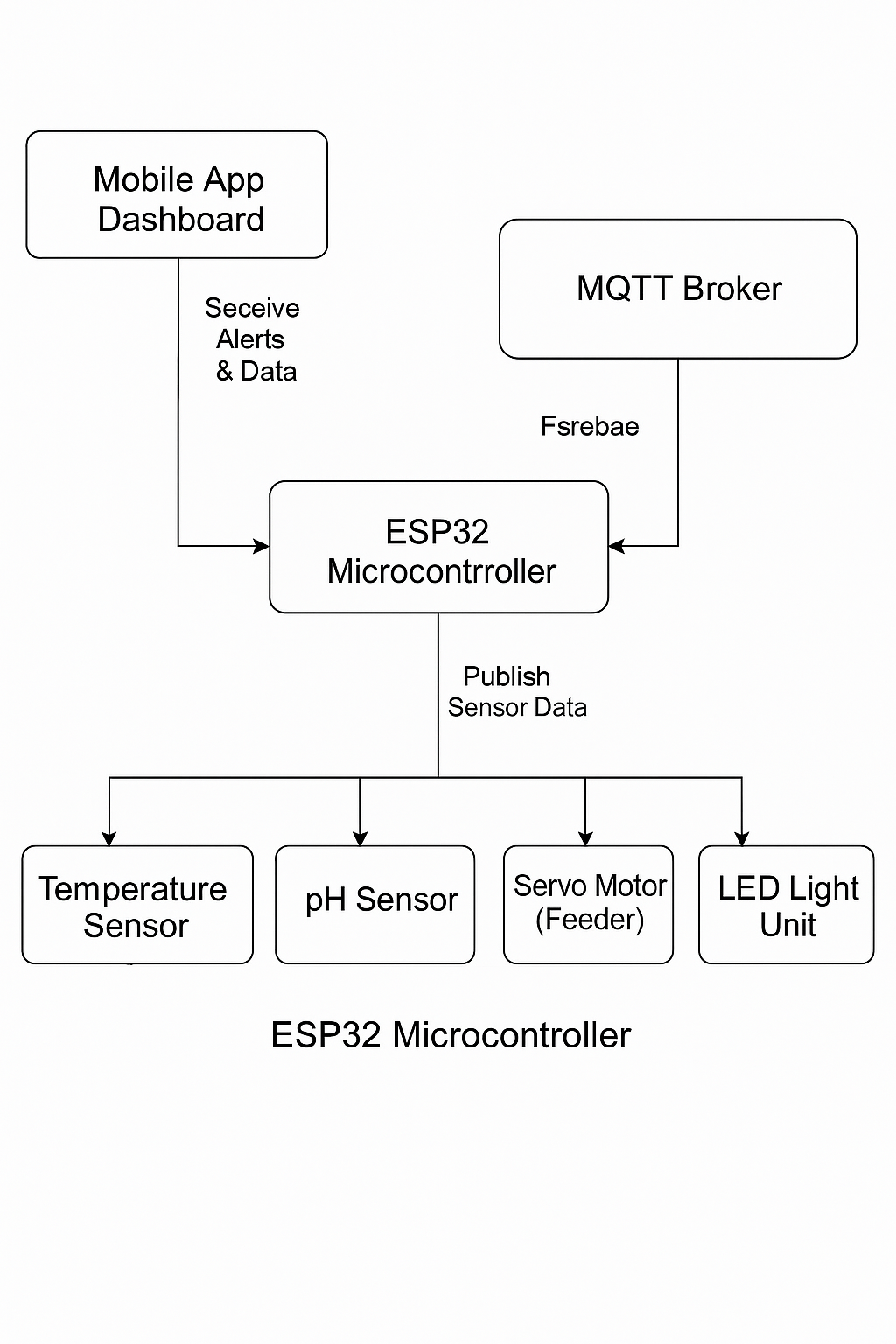


Fig 2: ESP32 Microcontroller

**6. Prototype Implementation**

**6.1 Use Cases Demonstrated**

**Use Case 1: Monitor Water Parameters**

The system continuously monitors the aquarium’s temperature and pH using sensors connected to the ESP32 microcontroller. These values are checked against safe thresholds, and if any abnormal condition is detected, an alert is triggered. The data is also published to the cloud for remote monitoring. Users are notified through the mobile app, allowing them to take manual action if needed. This ensures timely intervention to maintain a healthy aquatic environment and reduce the risk of harm to the fish.

**Use Case 2: Feed Fish Automatically**

A servo motor connected to the ESP32 controls an automatic fish feeder, dispensing food at predefined intervals. Feeding schedules are programmed into the firmware to maintain consistency and avoid overfeeding. Each feeding action is logged, and users can also trigger feeding manually via the mobile app. This automation ensures regular nutrition for the fish, even when the user is away, and helps keep the aquarium environment stable and clean.

**6.2 Tools and Technologies Used**

| **Category** | **Tool / Technology** | **Purpose** |
| --- | --- | --- |
| Microcontroller | ESP32 | Main control unit (Wi-Fi enabled) |
| IDE | Arduino IDE (C++) | Programming the ESP32 |
| Sensors | DS18B20 / DHT11 (Temperature), pH Probe | Real-time environmental monitoring |
| Actuators | Servo Motor, LED Unit | Feeding and lighting control |
| Communication | MQTT (via PubSubClient) | Lightweight message protocol |
| Cloud Platform | Firebase / ThingSpeak | Data storage and remote monitoring |
| User Interface | Tool/Technology | Next.js, React, and Tailwind CSS |

**6.3 GitHub Repository**

**The prototype code, wiring diagrams, and documentation we have made available on GitHub**

**GitHub Repository: [**[**https://github.com/gowthamde24/Automated-Aquarium-Project.git**](https://github.com/gowthamde24/Automated-Aquarium-Project.git)**]**

**A screenshot of a device

AI-generated content may be incorrect.**

**6.4 Design and Implementation Overview**

* The ESP32 is programmed to read sensor data every few seconds.
* The data is published over MQTT and stored in Firebase or ThingSpeak.
* The servo motor rotates at scheduled times to release food.
* LED lighting is controlled based on a fixed schedule or through manual app override.
* A mobile application enables the user to monitor readings and trigger manual commands if needed.
* The system is modular, allowing for future upgrades like water level sensors or automatic cleaning.

**7. Software Development Process**

**7.1 Chosen Process Model**

The development of the automated aquarium system followed a lightweight Agile approach, inspired by Kanban-style workflows. This model was chosen due to its flexibility and suitability for iterative development, especially when dealing with both hardware and software components. The team worked in short, informal cycles, focusing on one or two features at a time such as sensor integration, automated feeding, or cloud connectivity. Tasks were distributed among members, and progress was tracked through direct communication rather than formal tools like sprint boards or issue trackers.

Although no official Agile ceremonies or tooling (e.g., pull requests, GitHub Issues) were used, the commit history and modular implementation reflect Agile principles. Features were developed incrementally, tested frequently, and improved based on practical feedback from real hardware interaction. This approach allowed for rapid adjustments and continuous improvement, making it highly effective for a prototype-driven IoT project.

**7.2 Justification and Observed Practices**

* Development was done directly on a single main branch, without use of pull requests or formal reviews.
* Commits represent discrete progress steps, reflecting the implementation of individual functional components (e.g., sensor reading, feeding mechanism, mobile control).
* There is no formal issue tracking or sprint board, suggesting that team coordination was likely informal (through direct communication or external tools).
* The project evolved in self-contained iterations, each focusing on a feature or system component, aligning with Agile’s principle of incremental delivery.
* The lack of rigid structure made the team more flexible and adaptive, allowing changes to be made quickly based on test results or hardware constraints.

**8. AI Support Documentation**

**8.1 Steps Supported by AI**

AI tools, particularly ChatGPT (GPT-4), were used extensively throughout the project to support planning, modelling, and documentation. In the early stages, AI assisted in refining the business process description, drafting use cases, and rewriting user stories using the INVEST principle. This helped the team organize ideas more clearly and follow industry-standard formats for software engineering artifacts.

In later stages, AI was used to generate Mermaid syntax for domain models and system architecture diagrams, streamlining the creation of visuals without complex modelling tools. Additionally, AI supported writing technical sections of the final report, offering clean, structured explanations of system components, design decisions, and development process rationale. Overall, AI helped accelerate the workflow and improve clarity while still requiring human oversight for accuracy.

**8.2 AI Tools Used**

The primary AI tool used throughout the project was ChatGPT (GPT-4). It supported a wide range of tasks including drafting use cases, refining user stories based on the INVEST principle, and generating structured report content. ChatGPT also played a key role in creating Mermaid syntax for diagrams such as the domain model and system architecture, which simplified the visualization process without the need for dedicated UML tools. While other AI tools like GitHub Copilot were considered for coding assistance, the core AI interaction remained centred around ChatGPT due to its effectiveness in generating and refining technical documentation.

**8.3 Prompts and Outputs**

To support the modelling and documentation tasks in this project, several AI-generated prompts were used. These prompts were designed to guide the AI in producing accurate and structured outputs relevant to each step of the software engineering process. Below are three key examples:

**Prompt 1:**"Generate a domain model for an IoT-based Automated Aquarium system. It should include entities related to sensors, controller, feeder, lighting, alerts, and user interactions. Use Mermaid syntax to represent the class diagram."  
This was used during Step 4 to generate a visual domain model quickly and accurately.

**Prompt 2:**  
"Rewrite the user story for feeding fish automatically using the INVEST principle and explain each part."  
Used in Step 3 to reformat and refine user stories based on best practices.

**Prompt 3:**  
"Write a concise system architecture section for an IoT-based aquarium project, including both hardware and software components, and explain the role of each part clearly in paragraph form."  
This prompt helped generate structured text for Step 5 of the report, covering both technical depth and readability.

**8.4 Evaluation of AI Usefulness**

Using AI tools significantly improved the efficiency and quality of documentation and modelling. It helped the team quickly produce structured content, generate diagrams without needing complex software, and polish the technical report with formal language. However, AI-generated diagrams and code still required manual validation to ensure accuracy and relevance to the actual implementation. In short, AI served as a helpful assistant—not a replacement for critical thinking and hands-on development.

**9. Conclusion**

This project successfully explored the development of an IoT-based automated aquarium system, applying core principles of software engineering across all phases—from business process modelling and use case analysis to domain modeling, architecture design, and prototype implementation. The system, built using an ESP32 microcontroller, sensors, actuators, MQTT, and Firebase, demonstrated the ability to monitor water parameters, automate feeding, and allow manual override via a mobile app.

The project was developed using a lightweight Agile methodology, allowing for incremental progress, rapid testing, and flexible iteration—essential for hardware-software integration. AI tools such as ChatGPT were used to assist in modeling and documentation tasks, improving efficiency and helping maintain a structured approach throughout development. Despite the informal process, the project adhered to the goals of continuous improvement and user-centered functionality.

Looking forward, the system could be enhanced by integrating additional sensors (e.g., water level, ammonia), implementing automated water filtration or cleaning, enabling real-time notifications, and improving the mobile interface with richer controls and analytics. Security features, offline fallback mechanisms, and power optimization could also be considered for real-world deployment. This project provided valuable experience in working with IoT ecosystems, Agile practices, AI-assisted development, and cross-functional collaboration—skills that are directly applicable to real-world smart system design.

**10. References**

1. Espressif Systems. (n.d.). *ESP32 Series Datasheet*. Retrieved July 2025, from https://www.espressif.com/en/products/socs/esp32

2. MQTT.org. (n.d.). *MQTT Version 3.1.1 Protocol Specification*. Retrieved July 2025, from https://mqtt.org/

3. Firebase. (n.d.). *Firebase Realtime Database Documentation*. Google Developers. Retrieved July 2025, from https://firebase.google.com/docs/database

4. Arduino. (n.d.). *Arduino IDE Documentation*. Retrieved July 2025, from https://www.arduino.cc/en/software

5. ThingSpeak. (n.d.). *ThingSpeak IoT Analytics Platform*. MathWorks. Retrieved July 2025, from https://thingspeak.com/

6. Mermaid.js. (n.d.). *Mermaid Diagram Syntax Documentation*. Retrieved July 2025, from https://mermaid.js.org/

7. OpenAI. (2024). *ChatGPT (GPT-4) Technical Overview*. Retrieved July 2025, from <https://openai.com/chatgpt>

8. Beck, K., & Andres, C. (2004). *Extreme Programming Explained: Embrace Change* (2nd ed.). Boston, MA: Addison-Wesley.

**11. Appendices**

**Appendix A: Dashboard Screenshot**

A screenshot of a computer

AI-generated content may be incorrect.

Figure A.1 – Main dashboard showing task overview and user options.

**Appendix B: Mobile View**

**A screenshot of a device

AI-generated content may be incorrect.**

Figure B.1 – Responsive mobile interface for on-the-go access.

**Appendix C: User Story**

**A screenshot of a computer

AI-generated content may be incorrect.**

Figure C.1 – User story diagram showing actor interaction with the system.

**Appendix D: Wiring Diagram**

**A screenshot of a computer program

AI-generated content may be incorrect.**

Figure D.1 – System wiring diagram illustrating architecture flow.