IOT Enabled Hydroponic Farm Monitoring Using Arduino & Cloud

Jason Alexander Tan
Business Information Systems
Binus International
Jakarta, Indonesia
jason.tan001@binus.ac.id

***Abstract*—This case study focuses on the development of an Arduino and PHP based integrated hydroponic farm monitoring system by leveraging IoT and cloud technologies. The system's objective is to deliver relevant environmental parameter data, such as parts per million (PPM), pH levels, CO_2 levels, humidity, and temperature, to greatly optimize the routine hydroponic monitoring activity performed by JUST HYDROPONICS.**

Keywords—hydroponic; farm; arduino; php; monitoring; iot; cloud; system

I. INTRODUCTION (*Heading 1*)

A. Introduction

Agriculture, vital to sustaining the growing global population, faces the challenge of meeting increased food demands. Hydroponics, an alternative to traditional agriculture, offers advantages such as faster growth, higher yields, and reduced water usage [1]. However, it introduces complexity, necessitating precise monitoring of environmental factors. Leveraging IoT technology, particularly Arduino and cloud computing, presents an opportunity to create an efficient hydroponic farm monitoring system.

B. Background

Hydroponics, while not a new concept, has gained popularity due to technological advancements and a need for sustainable practices [1]. IoT technology, exemplified by Arduino, and cloud computing platforms like Google Cloud and AWS, enable real-time monitoring and data storage, enhancing the viability of hydroponics.

C. Object of Study

The focus is on a hydroponic farm, "Just Hydroponics," a family-owned business on the outskirts of Bogor City, with expansion plans.

D. Case Study Objectives

The case study aims to develop an Arduino-based hydroponic farm monitoring system. It comprises an Arduino sensor device collecting data and a web server housing a database and visualization tools. The system serves as a proof of concept, emphasizing technological feasibility and providing a foundation for future studies.

E. Research Questions

1. How does the proposed Arduino-based and cloud-powered system compare to manual methods in tracking hydroponic environmental changes?
2. What components constitute the solution, and what roles do they play?
3. What is required for building these components?
4. How does the proposed solution differ from manual methods, and what are the advantages and drawbacks?
5. Is the solution ready for immediate commercialization post-case study?

F. Hypothesis

1. Cost-effectiveness of the proposed system depends on farm scale, with an expected efficiency increase for larger farms.
2. The solution comprises Arduino sensors and a web server, each with specific components and functions.
3. Building components involves Arduino, sensors, Wi-Fi module, PHP, MySQL, and basic hardware.
4. Advantages include scalability, reduced losses, real-time data, and consistency, while drawbacks involve costs and initial reliability concerns.
5. The solution, as-is, is not ready for immediate commercialization due to cost, reliability, and usability considerations.

G. Scope & Limitations

The project focuses on an Arduino-powered hydroponic monitoring device and a PHP-based web application. The device monitors six environmental variables, while the application provides data visualization. The study excludes direct actions influencing the farm, complex security features, and individual user accounts for simplicity.

II. THEORETICAL FOUNDATION

A. Arduino Uno

Arduino Uno, an open-source hardware and software solution from Italy, serves as the cornerstone for the hydroponic sensor project [2]. Its adaptability to project-specific needs,

owing to its open-source nature, and user-friendly design make it an ideal platform [2].

The selected board, Arduino Uno R3 16U2, boasts specifications including an ATMEGA328P microcontroller, 5V operating voltage, and various input and output parameters [3]. Notable features encompass 14 digital I/O pins (6 PWM), 6 analog input pins, flash memory of 32KB, and a clock speed of 16 MHz [3].

Breaking down the Arduino Uno components, digital pins, a built-in LED, Power LED, ATMEGA328P microcontroller, Analog Input pins, Power Pins, Power Connector, RX and TX LEDs, USB jack, and Reset Button are identified and detailed [3].

Controlled through the Arduino IDE, version 2.0.4 at the time of the study, using C++ language with a subset of the standard C library due to RAM constraints [2] [4]. The IDE facilitates seamless communication with the Arduino Uno, crucial for project development.

B. Wi-Fi Module

For wireless communication, the ESP-01 Wi-Fi module, featuring the ESP-8266 chip, was integrated with the Arduino Uno [5]. Recognized for its cost-effectiveness and robust capabilities, the ESP-01, in contrast to alternatives like Wi-Fi Shield and Arduino Yun, presents an economical solution [5].

The ESP-8266 chip, utilizing 3.3V, demands over 300mA at peak draw. It houses 64 KiB of instruction RAM, 96 KiB of data RAM, and 4 MiB of QIO FLASH [6]. To harness its full potential with an Arduino Uno, an external 3.3V power supply is imperative due to the board's limited 3V3 port capabilities [6].

Operable by default with AT commands, the ESP-01 firmware permits critical operations like resetting the module, configuring Wi-Fi modes, connecting to access points, and interacting with web servers [7]. Each command, such as retrieving firmware version (AT+GMR) or setting Wi-Fi mode (AT+CWMODE=<mode>), adheres to specific syntax and produces anticipated responses [7].

Two communication modes were explored: USB bypass mode and SoftwareSerial mode [7]. The former treats the Arduino Uno as a TTL to USB adapter for direct communication with the computer. In contrast, the latter leverages SoftwareSerial for Arduino-ESP-01 communication, allowing independent operation without a direct computer connection [7].

In USB bypass mode, the ESP-01 is configured to connect to the computer directly through the Arduino Uno [7]. Alternatively, in SoftwareSerial mode, the ESP-01 communicates with the Arduino Uno's microcontroller, emphasizing independent operation without a computer connection [7]. However, challenges arise due to baud rate disparities, leading to garbled responses [7].

C. Water Temperature Sensor

For water temperature measurement, the DS18B20 sensor, a third-party wired long probe variant, was employed. Despite being based on Maxim Integrated's documentation, the acquired model differed in physical configuration [8].

The DS18B20, a digital thermometer, offers 9 to 12-bit Celsius temperature readings and operates in external or parasitic power mode. Each unit possesses a unique 64-bit serial code, enabling multiple DS18B20s on a single wire bus [8].

The DS18B20 is capable of operating in two power modes. In external power mode, the sensor requires three wire connections, drawing power from the VDD pin [9]. In parasitic power mode, utilizing only two pins (data and ground), this mode draws power from the data line, impractical for Arduino Uno due to insufficient power from the data pin [9].

Regardless of the power mode, successful sensor interfacing demands the installation of the OneWire and DallasTemperature libraries, facilitated through the Arduino IDE's "Manage Libraries" feature [9].

D. Humidity Sensor

For humidity measurement, the DHT11 sensor was employed, capable of measuring temperature and humidity with a calibrated digital signal output. The sensor comprises a resistive-type humidity measurement component and an NTC temperature measurement component, with 4 pins, where pin 3 is not connected [10].

The DHT11 operates on 3-5.5V DC, requiring a minimum 1-second delay for stable status after powering up. Operating beyond specifications may shift calibration, and exposure to chemical vapors can interfere with sensitive elements. If calibration shifts occur within operating boundaries, the DHT11 can self-calibrate over time [10].

The DHT11 measures humidity by gauging electrical resistance between two electrodes. Higher humidity lowers resistance, while lower humidity increases it [11].

To interface with the DHT11, the DHTLib library is required, downloadable from various internet sources. Installation instructions may be found on GitHub.

E. CO2 Sensor

For CO2 measurement, the MH-Z19B sensor by Winsen Electronics, utilizing non-dispersive infrared (NDIR) technology, was employed. It features built-in temperature compensation, UART output, and PWM input, offering high sensitivity, resolution, low power consumption, and resistance to water vapor interference [12].

The sensor has nine pins, with Vin, GND, PWM, Vo, RX, TX, and HD (calibration) being utilized. The technical specifications include a measurement range of 0-2000 ppm (0-5000 ppm), accuracy of ±(50ppm + 3% measured value), and operates at 4.5-5.5V DC. Interface settings recommend a baud rate of 9600, 8 data bits, 1 stop bit, and no parity [12].

F. pH Sensor

To measure pH levels, a PH-4502C sensor kit was employed, consisting of the PH-4502C module and a pH probe.

According to the figure above, the PH-4502C module features six pins on its left side, each serving specific functions (13):

1. The TO pin is for temperature output.
2. The DO pin provides a 3.3V output for pH limit.
3. The PO pin is the analog pH output.

4. The first GND is for the pH probe.
5. The second GND is for the board itself.
6. The VCC pin is the 5V power pin.

The PH-4502C board communicates pH values via voltage. The default is pH 7 at 0V, indicating it will go negative for acidic pH values, which Arduino cannot read. Therefore, it needs to be offset, with a pH value of 0 represented by 0V and a pH of 14 by 5V.

A pH electrode measures pH levels by the voltage potential it measures, dependent on the liquid's acidity or alkalinity. It's crucial to note that pH probes are inherently unstable, requiring regular cleaning and calibration.

To set up the sensor, connect 4 of the 6 pins to the Arduino board. The PO pin should link to the Arduino A0 analog port, both GND pins to the Arduino's GND ports, and the VCC pin to the Arduino's 5V port.

Calibration involves turning the potentiometer near the BNC connector to the correct offset. Short the BNC connector's inner wire with its outer shell to simulate a neutral pH value of 7. Measure the PO pin's value using a multimeter, adjusting the potentiometer until it reads 2.5V.

According to CimpleO [13], it's crucial to note that a pH probe takes some time to reach the correct value (at least two minutes). Readings may fluctuate if the temperature is above 30°C or below 10°C.

G. PPM Sensor

To measure PPM (parts per million), an Analog TDS Sensor by DFRobot was utilized. This TDS sensor provides an Arduino-compatible analog output, operates within a wide 3.3V ~ 5.5V input range, and outputs 0V ~ 2.3V analog. The TDS probe is waterproof and can be submerged in water for extended periods.

The PH2.0-3P connection interface is divided into three, with GND connecting to GND, VCC connecting to 5V, and Analog connecting to A0 ~ A5 on the Arduino board (DFRobot, n.d.). The probe is also connected to the other side of the board and immersed in the medium intended for measurement (DFRobot, n.d.).

Calculating TDS PPM involves measuring electrical conductivity. DFRobot's sensor uses the NaCl conversion factor, which is 0.5 (DFRobot, n.d.). This factor is multiplied by the electrical conductivity in microsiemens to produce PPM (DFRobot, n.d.).

H. PHP

1) Overview

Originally "Personal Home Page Tools," PHP is a versatile scripting language crucial for web development. Created in 1994, it evolved into a powerful language with cross-platform compatibility. PHP's strengths include database interaction, seamless HTML embedding, and extensive protocol support (14). Its open-source nature, established community, and numerous frameworks like Laravel make it a cost-effective and widely accepted choice.

2) MVC Design Pattern

The Model-View-Controller (MVC) design pattern divides applications into Model, View, and Controller for modularity. The Model manages logic and data, the View displays information, and the Controller handles application flow. Despite advantages like modularity and scalability, MVC has drawbacks, including boilerplate verbosity and complexity, making it more suitable for medium to large applications (15).

I. HTML & CSS

HTML (Hypertext Markup Language) and CSS (Cascading Style Sheet) form the foundation of modern web page development. HTML serves as the structure, defining elements like paragraphs and headings. It structures documents into a Document Object Model (DOM) to render web pages (16).

CSS, on the other hand, acts as the stylist, determining the visual appearance of the website, covering aspects such as fonts, colors, and margins (17). CSS provides precise control over layout and display characteristics, allowing the creation of visually appealing and user-friendly websites. Combining HTML and CSS has become the global standard for frontend web development.

Following the cascading principle, CSS styles can be inherited, overridden, or combined. It allows developers to create responsive designs through "media queries," adapting styles based on the user's device, screen resolution, or orientation (17).

J. JavaScript

JavaScript, a dynamically typed and interpreted programming language, plays a crucial role in web development, providing the interactivity and responsiveness essential for modern websites (18). Alongside HTML and CSS, it forms a core technology layer in the majority of contemporary front-end code.

Key features of JavaScript include its Turing completeness, enabling it to perform any algorithmically expressible computation. It encompasses fundamental programming elements like variables, functions, control flow structures, objects, and asynchronous programming. With a C-like syntax resembling languages such as C++, C#, and Java, JavaScript is accessible to developers familiar with these languages.

JavaScript's capabilities extend to event handling, detecting user actions like clicks and keystrokes. It also excels in Document Object Model (DOM) manipulation, dynamically altering static HTML code to enhance webpage dynamism. Originally designed for front-end development, JavaScript has expanded its reach into server-side programming through frameworks like Node.js, allowing developers to create full-stack web applications without relying on a dedicated server-side scripting language (18).

K. XAMPP

XAMPP, a cross-platform open-source web development environment, streamlines the creation and management of local web development environments. It encompasses Apache Web Server, MySQL, PHP, and Perl. The pre-configured components facilitate immediate development and testing. XAMPP's inclusion of PHP, a MariaDB server, and a Web Server aligns with the requirements of this case study.

1) MySQL

MySQL, an open-source relational database management system, adheres to the relational database model, employing tables for data organization. It features schemas, utilizes SQL for querying, and supports diverse data types. Known for its simplicity and ease of use, MySQL was chosen for its compatibility with the project's needs and the author's proficiency.

2) Apache Web Server

The Apache Web Server, an open-source web server and deployment platform, offers a user-friendly, one-click local hosting solution. Maintained by the Apache Software Foundation, it boasts a robust community. Supporting virtual hosts, Apache enables hosting multiple web applications on a single server. Available across major operating systems, it serves as the test deployment platform for this case study, providing simplicity and pre-configuration.

3) phpMyAdmin

Packaged with XAMPP, phpMyAdmin is a PHP web-based database management application tailored for MySQL and MariaDB databases. Particularly beneficial for developers lacking a dedicated database engine, phpMyAdmin enhances efficiency in database management during the development process (19).

L. Amazon Web Services

Amazon Web Services (AWS) stands out as a premier cloud computing platform, offering a comprehensive suite of over 200 services globally. From foundational infrastructure to cutting-edge technologies, AWS ensures swift, efficient, and cost-effective solutions, facilitating seamless migration and enabling novel possibilities.

1) Amazon RDS

Amazon RDS simplifies relational database management with scalability and cost-efficiency. Automated tasks, including backups and patching, enhance ease of use. Supporting MySQL and PostgreSQL, RDS prioritizes security through IAM and VPC. Customization is achievable with RDS Custom, while AWS Outposts extends managed databases on-premises. Varied DB instances provide capacities and storage options, ensuring high availability in AWS Regions with Availability Zones. Security groups manage access, and Amazon CloudWatch facilitates monitoring, following a shared responsibility model.

2) Amazon EC2

Amazon EC2 offers flexible and scalable computing power, streamlining server management for dynamic capacity adjustments. Instances operate within secure Virtual Private Clouds (VPCs), each fortified by firewalls. Key features include diverse instance types, secure login with key pairs, and versatile storage and networking capabilities. EC2 empowers users with robust computing resources, contributing to AWS's excellence in cloud infrastructure.

III. Problem Analysis

A. Current Processes

Just Hydroponics currently relies on manual data collection using handheld tools, primarily the HI98301 TDS Meter. Limited by its high cost, the farm supplements with cheaper, less accurate models. Data is collected once a day, posing challenges in responding to sudden environmental changes. Lack of standardization in measuring tools compromises data quality, focusing solely on nutrient ppm and neglecting crucial variables like temperature and CO_2 saturation.

Data storage involves whiteboards, creating vulnerabilities. Historical data is at risk of being wiped, and manual transcription into notebooks is impractical.

B. Problem Analysis

Storing data on whiteboards introduces extreme vulnerability, with potential for data loss due to accidents or weather. Migrating to a cloud-based solution enhances data security, availability, and resilience.

Whiteboard storage makes data management laborious. A cloud-based web application simplifies data management, requiring minimal man-hours.

Manual data collection and management demand significant manpower. Introducing an Arduino-based system automates processes, reducing the need for manual data collection and streamlining data maintenance.

Manual data collection introduces inaccuracies and inconsistencies, driven by interpretation variations and human errors. Electronic sensors, calibrated to a standard, can eliminate these factors, ensuring data accuracy and consistency.

Manual data collection schedules lead to delayed responses to hydroponic system changes. An automated system allows for higher-frequency data collection, immediate alerts, and efficient responses without incurring additional costs.

Manual practices result in an inefficient allocation of financial resources. Implementing automation reduces maintenance costs, increases reliability, decreases workload, ensures consistency, and facilitates continuous monitoring.

IV. Solution Design

A. Arduino

1) Architecture Overview

The Arduino-based sensor comprises five sensors (DS18B20, DHT11, MH-Z19B, PH-4502C, DFRobot TDS Meter) collecting temperature, humidity, CO2, pH, and nutrient ppm. Data is compiled into a URL pattern adhering to PHP $_GET, sent via the ESP8266 Wi-Fi module in a continuous "Loop Cycle."

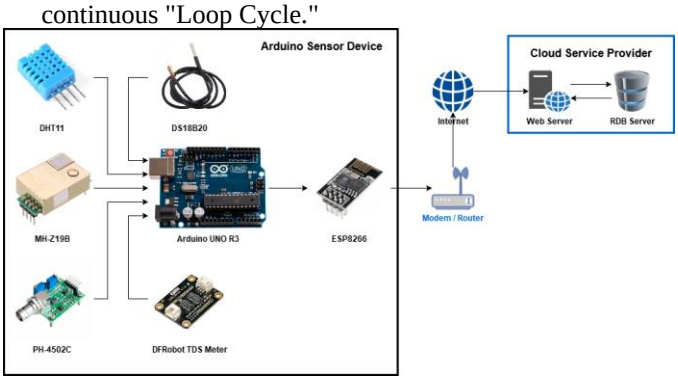

Fig. 1. The Arduino Architecture Overview

2) Loop Cycle

A continuous process involves data collection, compilation, and transmission via the ESP8266 Wi-Fi module. The "Loop Cycle" ensures consistent operation, allowing the Arduino to transmit data as long as it's powered and operational.

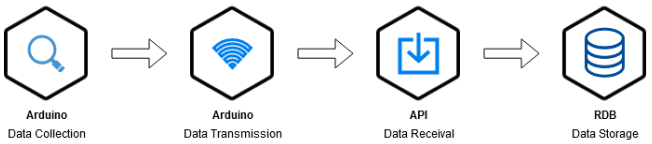

Fig. 2. The System Process Cycle

3) Data Collection Frequency

Configurable data collection frequency, advised between once every half hour to once every hour. Balancing frequency with storage costs is essential, aligning with General Hydroponics' recommendations for optimal hydroponic system monitoring.

4) Unit Cost

Detailed unit cost breakdown for Arduino components, totaling Rp 1,039,275.

TABLE I. UNIT COST BREAKDOWN

Item	Unit Cost	Total Cost
Arduino Uno R3	Rp 120.500	Rp 120.500
Jumper Cables	Rp 8.800 (x4)	Rp 35.200
ESP-01 Wi-Fi Module	Rp 15.000	Rp 15.000
Small Breadboard	Rp 7.100	Rp 7.100
DHT11	Rp 10.800	Rp 10.800
DS18B20	Rp 9.775	Rp 9.775
Resistor 4.7k Ohm 1/2W	Rp 400 (x2)	Rp 800
MH-Z19B	Rp 574.500	Rp 574.500
DFRobot Analog TDS Sensor	Rp 227.400	Rp 227.400
PH-4502C + pH Probe	Rp 265.600	Rp 265.600
	Final Total	**Rp 1,039,275**

B. *Web Application*

1) Architecture Overview

The web application requires an active internet connection and is designed for deployment on various platforms, including cloud services. The MVC design pattern is implemented with eight classes, including controllers, database connections, and classes for managing sensor data.

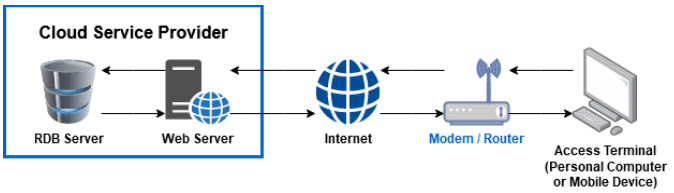

Fig. 3. The Web Application Architecture Overview

2) Class Diagram

An eight-class diagram representing the MVC design pattern. Classes include controllers (DashboardController, LoginController), database connections (DatabaseConnection), and classes for handling sensor data (SensorData, SensorDataArray, Sensor, SensorArray).

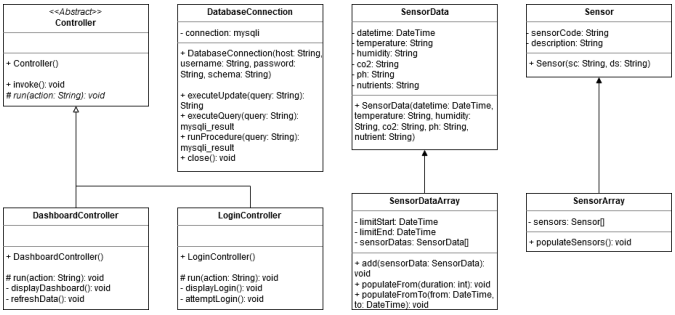

Fig. 4. The Web Application Class Diagram

3) Design Pattern

The Model View Controller (MVC) design pattern is the foundation, involving four Model classes, two Controller classes, and two View pages (dashboard.php and login.php). The program flow starts at index.php, invoking the router, which, in turn, calls the desired controller.

4) Use Case

A use case diagram illustrates user interactions with the system, featuring functionalities like selecting sensors, date range, and ordering data based on various parameters.

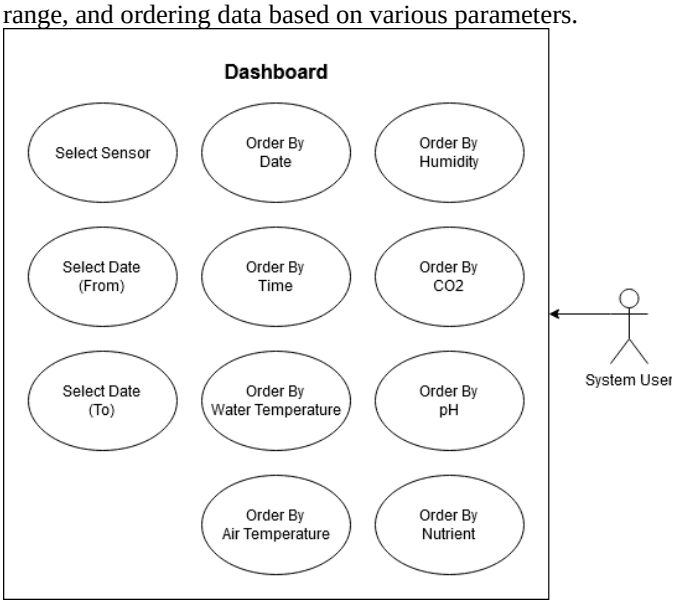

Fig. 5. The Web Application Use Case

5) User Interface

The user interface includes a dashboard displaying eight columns of data: Date, Time, Air Temp, Water Temp, Humidity, CO2, pH, and Nutrient. A login page is a placeholder with a simple password entry for demonstration purposes.

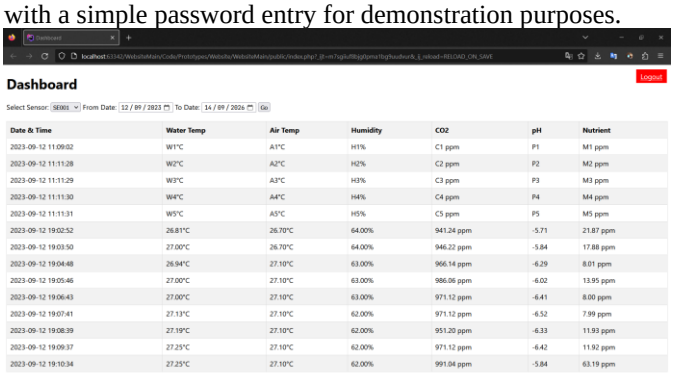

Fig. 6. The Simplistic & Utilitarian UI Design

6) Database

A simple database structure with two tables: sensor and master. The sensor table stores information about physical sensors, and the master table holds all collected data, referencing sensors through a foreign key.

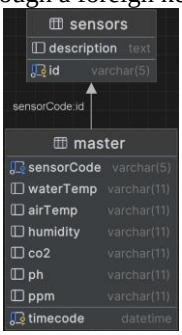

Fig. 7. The Database Structure

7) Arduino API

The API, integrated into the web application, resides in the /arduino directory. It follows a superglobal URL pattern, receiving data from the Arduino through PHP's standard $_GET format.

Fig. 8. An illustration of the $_GET Superglobal Used

C. Cloud Deployment

A cloud deployment plan for AWS, featuring an EC2 server, RDS database, and internet connections from the Arduino and user devices. While the database server should ideally have no direct contact with the Arduino, for debugging purposes, public internet access is enabled temporarily.

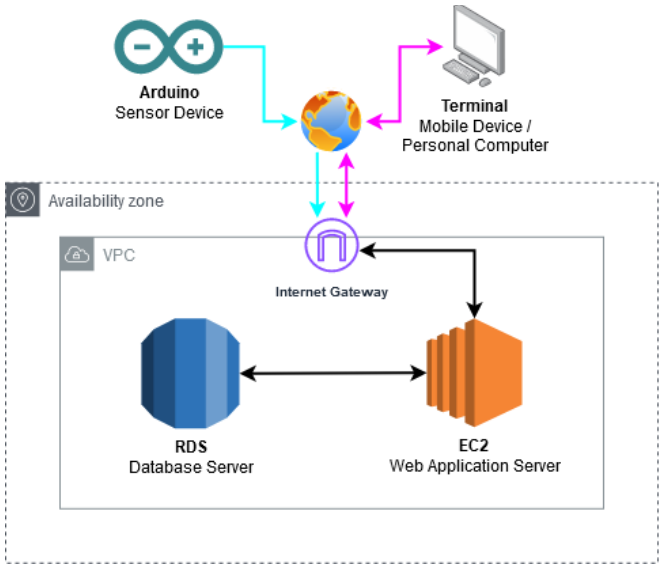

Fig. 9. The Cloud Architecture Plan

D. Testing Plan

A manual testing approach is adopted, dividing testing into unit and integration testing for both the Arduino and web application components. End-to-end testing includes basic acceptance testing, multi-device testing, and overall reliability testing, emphasizing data collection interval adherence during prolonged operational periods.

V. Solution Implementation

A. Arduino

1) Architecture

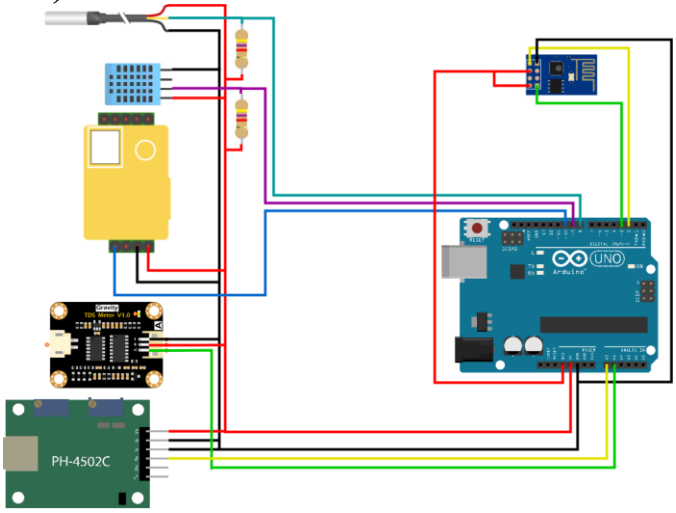

Fig. 10. A Schematic of the Arduino Device

a) The figure above illustrates the Arduino prototype architecture.

b) Power details: 3.3V for Wi-Fi, 5V for five sensor modules.

c) Sensor connections: Analog, digital, and PWM ports.

d) Specific pin connections for each sensor module outlined.

2) Prototype

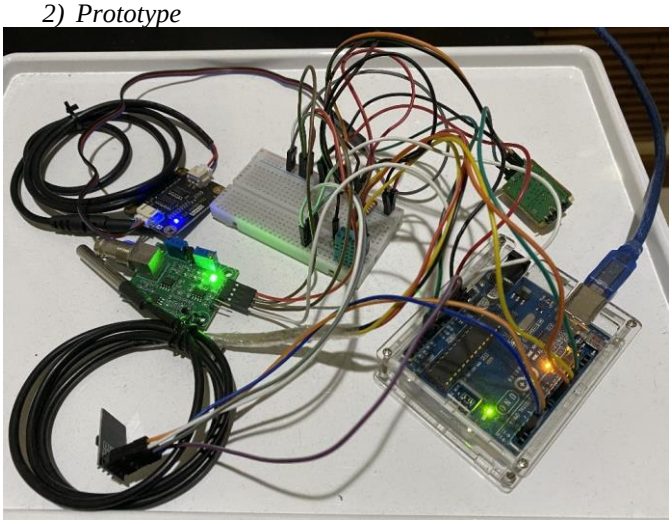

Fig. 11. An Early Prototype of the Arduino Sensor Device

The figure above depicts the physical prototype, emphasizing its proof-of-concept nature.

3) File Structure

Seven files structure the code for readability and maintenance. Roles include main execution, Wi-Fi operations, and individual sensor modules.

Fig. 12. The Arduino Code's File Structure

a) ArduinoMain: Main execution code with setup and loop functions. Responsible for importing libraries and calling functions from other files.

b) 00_WiFi: Wi-Fi module operations, including setup and command transmission.

c) 01_WaterTemperature: Code for water temperature probe setup and data retrieval.

d) 02_Humidity: Code for air temperature and humidity sensor setup and data retrieval.

e) 03_CO2: Code for CO2 sensor setup and data retrieval using PWM.

f) 04_PH: Code for pH sensor data retrieval through analog pin.

g) 05_Nutrient: Code for TDS meter setup and data retrieval, considering water temperature.

4) Program Logic & Loop

Arduino IDE compiles code and combines files. Execution starts with definitions, followed by setup functions and the continuous loop. Data from sensors is retrieved and transmitted via Wi-Fi in sequence.

5) The Code

For the sake of brevity, code details have been omitted in this technical report. It will focus only on the structure and key functionalities. For full details, please refer to the main report.

a) Main: Code imports libraries, defines sensor code, and initializes modules. Continuous loop retrieves sensor data and transmits it via Wi-Fi.

b) Wi-Fi (ESP-01): Commands for Wi-Fi module setup and data transmission. Functions to send data, run commands, and handle unreliable response.

c) Water Temperature (DS18B20): Setup and data retrieval functions for water temperature probe.

d) Air Temperature & Humidity (DHT11): Setup and data retrieval functions for air temperature and humidity sensor.

e) CO2 (MH-Z19B): Setup and data retrieval functions for CO2 sensor using PWM.

f) pH (PH-4502C): Data retrieval function for pH sensor using analog pin.

g) Nutrient (TDS Meter): Setup and data retrieval functions for TDS meter, considering water temperature.

B. Web Application

1) File Structure

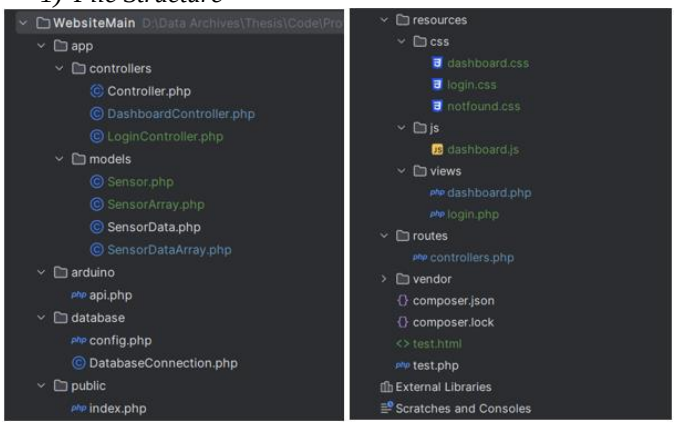

Fig. 13. The Web Application's File Structure

The file structure, depicted in the figure above, is a snapshot from the PHPStorm IDE, illustrating the organization inspired by Laravel. Key components include:

a) app folder: Hosts the application's logic. Notably, the controllers and models reside here, with the database connection class residing in the database folder.

b) arduino folder: Contains the API responsible for receiving data from the Arduino device via $_GET URL.

c) database folder: Includes the database configuration file and the DatabaseConnection class file. While the latter might be more suitably placed in the app folder, restructuring isn't deemed worthwhile at this point.

d) public folder: Houses the index file, serving as the application's "landing page" and calling the router.

e) resources folder: Contains view pages and supporting elements such as JavaScript and CSS files.

f) routes folder: Hosts the router file (controllers.php), determining active controllers and, consequently, displayed pages.

g) vendor folder: Primarily unused, containing files for Composer, though Composer isn't employed as the server is locally hosted using Apache on XAMPPs

2) Program Logic

a) The Index File: The program initiates in index.php within the public subdirectory. Here, it imports controllers.php, acquires the controller to-be-invoked, and proceeds to invoke it using the invoke() function.

b) Router, Check Function: The router (controllers.php) incorporates two functions - check() and load(). While load() loads a controller based on the provided parameter, check() acts as a "wrapper function," adding controller retention functionality via cookies to remember pages.

c) Controllers Abstraction Layer: Controllers implement an abstraction layer in the form of the abstract Controller class. With invoke() and run() as primary functions, each controller inherits the former, maintaining consistency, while run() is controller-specific, employing a switch for possible actions.

d) DashboardController's run() Function: Although run() differs across controllers, it follows a consistent design pattern, featuring a switch and a list of possible actions, facilitating data flow via GET or POST superglobals between view pages, controllers, and models.

3) Database & API

The database is comprised of two tables, which are the master table and sensor table.

a) Master Table: The master table, designed with 8 columns, stores timecode, sensor code, water temperature, air temperature, humidity, CO2, pH, and nutrient data collected from the Arduino sensor device.

b) Sensors Table: The sensors table is simpler, storing sensor information with a sensor code in the description column.

4) API Logic

Upon invocation, the API retrieves data from $_GET super global variables, defaulting to null if unspecified. It then checks if all variables are null; if so, data insertion is rejected. Otherwise, it builds a query string, calls *executeUpdate()* to send the SQL command to the database, and closes the connection.

5) Cloud Deployment

There is a full written guide for the cloud deployment implementation within the full report. The guide is too lengthy to be written here. Please refer to the full report.

VI. EVALUATION

A. Methodology

The evaluation methodology used is Case Study Evaluation, focusing on identifying common themes and patterns to understand challenges and successes during the solution's implementation. While the structure adheres to the formal Case Study Evaluation, practical flexibility is applied.

For the sake of brevity and to shorten the length of this technical report to meet the requirements as defined in the guidelines given by BINUS International, most of the detailed result diagrams are omitted.

B. Arduino

1) Unit Testing

a) Wi-Fi Module

The Wi-Fi module successfully inserted data into the database. However, a limitation was noted due to baud rate differences affecting message readability.

b) Water Temperature Module

The sensor module performed as expected, successfully retrieving environmental data.

c) Humidity Module

The sensor module successfully retrieved environmental data for both air temperature and humidity.

d) CO2 Module

The sensor module successfully retrieved environmental data. Manual calibration is required.

e) pH Module

The sensor module successfully retrieved environmental data, with a note about voltage skew if the pH probe is not properly connected.

f) TDS Module

The sensor module successfully retrieved environmental data. A minor discrepancy was noted when the electrodes were dried and held in the air.

2) Integration Testing

All sensor modules and the Wi-Fi module, when combined, worked as expected. The Arduino prototype successfully inserted environmental data into the database.

C. Web Application

1) Unit Testing

a) Website

Various test cases were performed, including displaying data, sensor selection, data table column sorting, logout button functionality, applying time filters, and functionality to remember time filters.

b) Database & API

Test cases involved normal data insert through API, all null data insert through API, and partial null data insert through API.

2) Integration Testing

Integration testing for the web application confirmed that the data inserted during unit testing was displayed successfully on the dashboard.

D. End-to-End

1) Basic Acceptance Testing

The end-to-end testing focused on three key evaluation points: Serial Monitor, Database, and Dashboard. All three evaluation points passed successfully.

2) Reliability Testing

This test is performed over a relatively long period of time (~10 hours). In this test, the Arduino device and web server will be kept running for the duration of the test. After the test's conclusion, the data collection interval will be analyzed for discrepancies. From the result of this analysis, we will be able to infer the reliability of the system. To isolate the device's reliability and to eliminate external factors, the test is done on local network (localhost) instead of on cloud provider servers.

Performing the test on local network ensures that any anomalies or irregularities caused by third party factors (such as the cloud service provider) are eliminated.

a) Local Testing

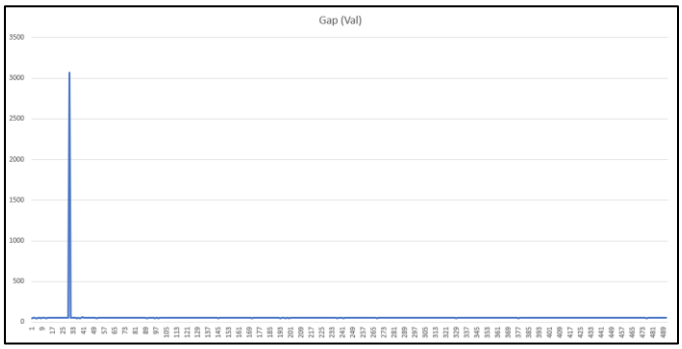

Fig. 14. A Visualization of the Time Gap Anomaly (Local Test)

	Timecode	Gap	Gap (Val)
27	26/10/2023 22:42	00:56	54.44444
28	26/10/2023 22:43	00:56	54.44444
29	26/10/2023 22:44	00:57	55.41667
30	26/10/2023 22:45	00:56	54.44444
31	26/10/2023 22:46	52:33	3065.417
32	26/10/2023 23:38	00:57	55.41667
33	26/10/2023 23:39	00:57	55.41667
34	26/10/2023 23:40	00:56	54.44444
35	26/10/2023 23:41	00:57	55.41667
36	26/10/2023 23:42	00:56	54.44444

Fig. 15. The Gap Value (Local Test)

The reliability testing was performed from 26/10/2023 10:19:46 PM until 27/10/2023 6:45:21 AM, which amounted to 8 hours, 25 minutes, and 35 seconds. In this timeframe, there was 492 successful insertion cycles with only a single anomaly between the 30th and 31st insertions. Under normal operating conditions, the gap between insertions is around ~56 seconds, but in the anomaly, the delay between insertions was 52 minutes and 33 seconds. Within that gap there should have been ~60 insertion cycles.

The overall reliability rating can be calculated by dividing what is by what should have, which means that the reliability is ~ 492 / (492+60), which amounts to ~89.13%.

b) Cloud Testing

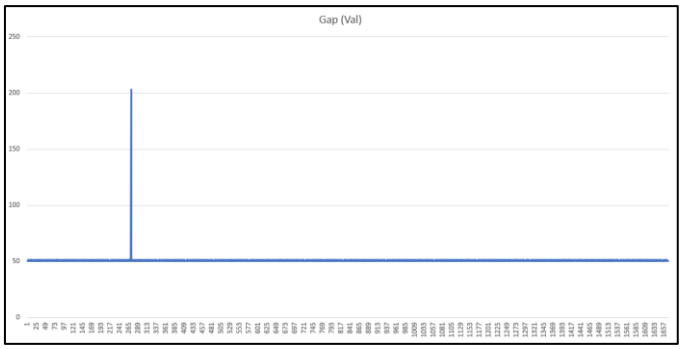

Fig. 16. A Visualization of the Time Gap Anomaly (Cloud Test)

270	28/11/2023 17:45	00:52	50.55556
271	28/11/2023 17:46	00:52	50.55556
272	28/11/2023 17:47	00:52	50.55556
273	28/11/2023 17:47	03:29	203.1944
274	28/11/2023 17:51	00:52	50.55556
275	28/11/2023 17:52	00:53	51.52778
276	28/11/2023 17:53	00:52	50.55556
277	28/11/2023 17:54	00:52	50.55556

Fig. 17. The Gap Value (Cloud Test)

The cloud reliability testing was performed from 28/11/2023 13:52:04 PM UTC until 29/11/2023 14:07:18 PM UTC, which amounted to 24 hours, 15 minutes, and 14 seconds. In this timeframe, there was 1670 successful insertion cycles with only a single anomaly between the 272nd and 273rd insertions. Under normal operating conditions, the gap between insertions is around ~52-53 seconds, but in the anomaly, the delay between insertions was 3 minutes and 29 seconds. Within that gap there should have been ~4 insertion cycles.

The overall reliability rating can be calculated by dividing what is by what should have, which means that the reliability is ~ 1670 / (1670+4), which amounts to ~99.76%.

3) Multi-Device Testing

In this test, I will evaluate the API's capability to accept connections (Data insertion) from multiple devices. This test is very simple. It will evaluate whether anything in the system behaves abnormally when data is being inserted from two devices simultaneously for 10 connection (data insertion) cycles.

For the purposes of this test, I have made a second Arduino contraption but without any of the sensor modules. The second contraption is designed to emulate a second sensor device and is therefore equipped only with a Wi-Fi (ESP-01) module. All the data that the emulator device inserts will be '69.42'.

11	2023-09-12 19:05:46	SE001	27.00	27.10	63.00	986.06	-6.02	13.95
12	2023-09-12 19:06:10	SE002	69.42	69.42	69.42	69.42	69.42	69.42
13	2023-09-12 19:06:43	SE001	27.00	27.10	63.00	971.12	-6.41	8.00
14	2023-09-12 19:07:06	SE002	69.42	69.42	69.42	69.42	69.42	69.42
15	2023-09-12 19:07:41	SE001	27.13	27.10	62.00	971.12	-6.52	7.99
16	2023-09-12 19:08:02	SE002	69.42	69.42	69.42	69.42	69.42	69.42
17	2023-09-12 19:08:39	SE001	27.19	27.10	62.00	951.20	-6.33	11.93
18	2023-09-12 19:09:37	SE001	27.25	27.10	62.00	971.12	-6.42	11.92
19	2023-09-12 19:09:41	SE002	69.42	69.42	69.42	69.42	69.42	69.42
20	2023-09-12 19:10:34	SE001	27.25	27.10	62.00	991.04	-5.84	63.19
21	2023-09-12 19:10:38	SE002	69.42	69.42	69.42	69.42	69.42	69.42

Fig. 18. Database Results for MDT

As can be seen in the figure above, the data insertion process has proceeded normally. Both devices inserted data into the database without issue.

E. Research Questions & Hypotheses Evaluation

The research questions and hypotheses were evaluated, covering aspects related to system performance, components, building requirements, differences from manual methods, and commercialization readiness.

VII. Discussion

A. Not Ready for Commercialization

The implemented solution is deemed not ready for commercialization due to usability challenges and the need for additional development.

B. *Range Extension*

The suggested range extension using a mesh network was considered physically impossible due to power constraints and Arduino Uno limitations.

C. *Wi-Fi Reliability*

Wi-Fi reliability issues were noted, and automated detection or recovery systems were not feasible due to communication challenges.

D. *Data Collection Frequency*

The suggested data collection frequency was once every 30 minutes to an hour. Managing data storage and older data were mentioned as considerations.

E. *Documentation Scarcity*

Challenges in documentation for components like the ESP8266 Wi-Fi module and SoftwareSerial class were highlighted. The lack of comprehensive resources impacted the research process.

F. *UI Design Simplicity*

The UI design's overt simplicity was discussed, acknowledging potential concerns regarding product image.

VIII. CONCLUSION & RECOMMENDATION

A. *Conclusion*

1) *Current State: Manual Data Collection*

- Just Hydroponics relies on manual data collection, leading to challenges in responding to environmental changes.
- The use of manual tools, lack of standardization, and vulnerability of whiteboard data storage pose risks.
- Time-consuming processes result in inefficient resource allocation, delayed responses, and inaccuracies.
- Labor-intensive procedures hinder scalability and profitability.

2) *Proposal: Automated System*

- Proposed solution: Arduino sensors and a cloud-based web application.
- Aim: Address vulnerabilities, inefficiencies, labor intensity, and improve response to environmental changes.
- Successfully produced a working prototype as a proof of concept.
- Acknowledges commercial unviability in the current state.
- Demonstrates the viability of an Arduino-based hydroponic sensor.
- Limited resources and skills hinder full commercialization within the case study's scope.
- Suggests that a well-resourced company can further develop and introduce the product to the market.

B. *Recommendation*

These recommendations aim to improve the product's readiness for commercialization. While the proof-of-concept is promising, addressing issues related to casing, power supply, economic viability, and database resilience will contribute to a more robust and market-ready solution.

1) *Casing Design*

- The current prototype is not ready for environmental use, especially in a hydroponic greenhouse.
- Recommends designing a casing to protect against environmental factors and electronic disruptions.

2) *Independent Power Supply for Wi-Fi Module*

- The current power supply from the Arduino board's 3.3V pin may be insufficient for the Wi-Fi module's full capabilities.
- I suggest implementing an independent power supply to enhance reliability.

3) *Economic Viability Study*

- An economic viability study can lower the device’s unit cost and enhance attractiveness to consumers.

4) *Parallel Databases*

- Parallel databases can reduce reliance on a stable internet connection.

ACKNOWLEDGMENT

This case study is dedicated to a few people, without whom I would have to endure unspeakable difficulties to get to where I am today. Words alone cannot express how grateful I am to have their unyielding support and company.

1. To my core family, especially my mother, to whom I owe everything.
2. To my extended family, which have been so kind and supportive as to allow me to use their business as the object of this case study and put up with my annoyingly large number of questions.
3. To my girlfriend, for her unwavering support, for her words of encouragement, and for her extraordinary ability to calm me down when times are difficult.
4. To my closest friends, for their invaluable friendship, their company, and the laughs we shared, which has opened many doors to me, be it physical, mental, or emotional.
5. To my thesis supervisor, Samuel Mahatmaputra T, S.Kom., M.Info.Tech., for providing me with invaluable guidance and inspiration during my thesis period and throughout the years since I started studying in Binus International, for being the lecturer who taught me the very lessons that are quintessential to my ability to perform this case study, for being the anchor of stability in my efforts of navigating Binus’s bureaucratic nightmare for Master Track, all of which were crucial in enabling me to finish this case study.

References

Please note that this is **not** the complete list of references used in this case study. Most of the references from the main report are not included because the relevant citations did not make it into this technical report. This is due to length limitations, as the technical report can only contain 10 pages as opposed to the 165 pages of the main report.

[1] L. Janes. (2019, July 30). *Top 15 Reasons Why You Should Grow Vegetables In A Hydroponic Garden*. VH Hydroponics. [Online]. Available: https://www.vhhydroponics.com/hydroponic-garden/

[2] Arduino. (2021, September 15). *About Arduino*. Arduino. [Online]. Available: https://www.arduino.cc/en/about

[3] freeCodeCamp. (2021, June 8). *Arduino Course for Beginners – Open-Source Electronics Platform*. freeCodeCamp.org. [Video]. Available: https://www.youtube.com/watch?v=zJ-LqeX_fLU

[4] S. Shinde. (2023, January 25). *What are the Key Pros and Cons of the Arduino Programming Language?* Emeritus. [Online]. https://emeritus.org/blog/coding-arduino-programming-language/

[5] A. Barela. (2023). *ESP8266 Temperature / Humidity Webserver*. Adafruit. [Online]. Available: https://learn.adafruit.com/esp8266-temperature-slash-humidity-webserver/overview

[6] L. Ada. (2015). *Adafruit HUZZAH ESP8266 Breakout*. Adafruit. [Online]. Available: https://learn.adafruit.com/adafruit-huzzah-esp8266-breakout/overview

[7] Bhimsen. (2022, September 9). *ESP8266 AT Commands List and Working Explained*. Electronics Fun. [Online]. Available: https://electronics-fun.com/esp8266-at-commands/

[8] Maxim. (2019). *DS18B20 – Programmable Resolution 1-Wire Digital Thermometer*. Maxim Integrated. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ds18b20.pdf

[9] R. Santos. (2016, August 24). *Guide for DS18B20 Temperature Sensor with Arduino*. Random Nerd Tutorials. [Online]. Available: https://randomnerdtutorials.com/guide-for-ds18b20-temperature-sensor-with-arduino/

[10] Mouser Electronics. (n.d.). *DHT11 Humidity & Temperature Sensor*. Mouser Electronics. [Online]. Available: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

[11] S. Campbell. (2015, October 1). *How to Set Up the DHT11 Humidity Sensor on an Arduino*. Circuit Basics. [Online]. Available: https://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-an-arduino/

[12] Winsen. (2016). *Intelligent Infrared CO_2 Module (Model: MH-Z19B)*. Zhengzhou Winsen Electronics Technology Co., Ltd. [Online]. Available: https://www.winsen-sensor.com/d/files/infrared-gas-sensor/mh-z19b-co2-ver1_0.pdf

[13] CimpleO. (2020, April 23). *Arduino pH-meter using PH-4502C*. CimpleO. [Online]. Available: https://cimpleo.com/blog/simple-arduino-ph-meter/

[14] PHP. (2023). *History of PHP and Related Projects*. PHP. [Online]. Available: https://www.php.net/manual/en/history.php

[15] R. D. Hernandez. (2021, April 19). *The Model View Controller Pattern – MVC Architecture and Frameworks Explained*. freeCodeCamp.org. [Online]. Available: https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/

[16] W3Schools. (n.d.). *HTML Introduction*. W3Schools. [Online]. Available: https://www.w3schools.com/html/html_intro.asp

[17] W3Schools. (n.d.). *CSS Introduction*. W3Schools. [Online]. Available: https://www.w3schools.com/css/css_intro.asphttps://www.w3schools.com/css/css_intro.asp

[18] Mozilla. (2023, September 16). *What is JavaScript?* MDN Web Docs. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

[19] XAMPP Team. (2023). *Windows Frequently Asked Questions*. Apache Friends. [Online]. Available: https://www.apachefriends.org/faq_windows.html