Design and construction sun drying wet clothes system (DCDCS)

Hoàng Phi Long
FPT University
Ho Chi Minh City

Nguyễn Đình Phong
FPT University
Ho Chi Minh City

Trịnh Bình
FPT University
Ho Chi Minh City

***Abstract*— Vietnamese have long working hours which means they spend time at evening and night to do their chores. The chores include washing and drying clothes. However, Vietnam also has long rainy season which indicate a persistent problem of inefficient clothes drying process.**

Keywords—DCDCS–Design and construction sun drying wet clothes system, clothes drying system, rain sensor, Internet of Things, React Native, ExpressJS, NodeJS, automatic clothes-drying system

1. INTRODUCTION

An automatic clothes drying system, which uses rain sensor to detect rain and ESP8266 for communications between Android application and embedded device, was developed to allow consumers to effectively manage their chores. This document will explain the foundations and the processes of this innovative system.

Furthermore, this document will describe our working process in 4 months includes our perspective in the system, component designs and detailed core workflows. We hope the system will help resolve some aspects of the problem that the current face recognition systems are facing today.

2. PROBLEM AND SOLUTION

2.1 Problem definition

Advantage of existing system on the market

- UV disinfection
- Built-in dryer
- Strong structure which can lift up to 25kg of clothes
- Drawbacks of existing system on the market
- High production costs which lead to relatively unaffordable selling prices
- Hard to extend
- Automation fully dependent on electricity
- Cannot automatically collecting clothes.

2.2 Proposed solution

Our proposed solution is to design and construct an automatic clothes drying system called DCDCS to solve missing feature of the current “Smart Clothesline Rigs”. Our system will allow the automation of laundry-colleting in the case of rains. Our system will also be competitively priced, easier installation, more compact and mobile, and extendable compare to the existing system.

DCDCS system includes a mobile app and an embedded device with following functions:

2.2.1 Feature Functions

- **Mobile App:**
  - Control the system through wireless
  - Check weather information
  - Check system status
- **Embedded Device:**
  - Check system status
  - Control system through hard buttons

2.2.2 Advantages and Disadvantages

- Advantages:
  - Low costs which allow more affordable prices
  - Fast rain detection
  - Can control using mobile app
  - Use solar energy and store extra energy as battery for use under adverse conditions
- Disadvantages:
  - Cannot detect whether the clothes is dry or not
  - Cannot detect whether rain is over or not

3. IMPLEMENTATION

3.1 Basic concept

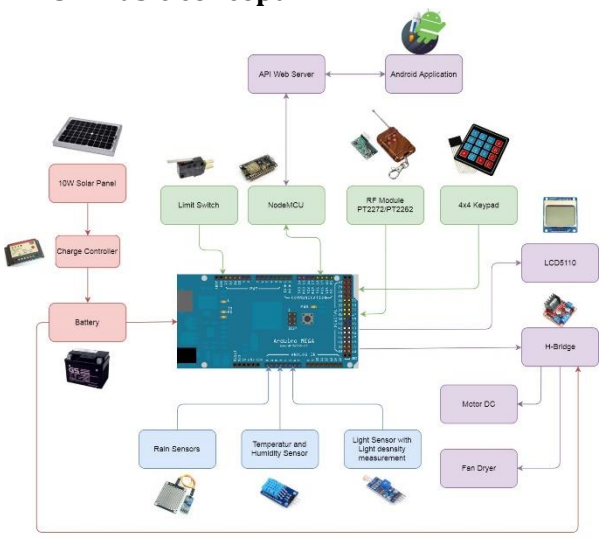

Figure 1: System block diagram

The block diagram above illustrates the underlying concept of the Automatic clothes-drying system. At first, the system will gather data from connected sensors. From collected data, the system will determine its own action to control the dryer or dc motor to collect or dry clothes. After that, the system will send all those data to the server via HTTPS request for the mobile application. Beside sensors data, the system can be controlled by RF Remote or Keypad on the system. About Mobile application, users can check system information with HTTPS request/response. To control the system with the mobile app, each time the user request an action; the mobile app will send a request to a server which is a message contains data to control the system. The server receives the message and pushes it to the message queue. Now, the system will ask each server for any messages from the system. If there is, the server will send the message via HTTPS response for the system. From the given message, the system now can control dc motor or dryer according to users' wants.

3.2 Architecture

3.2.1 System overview architecture

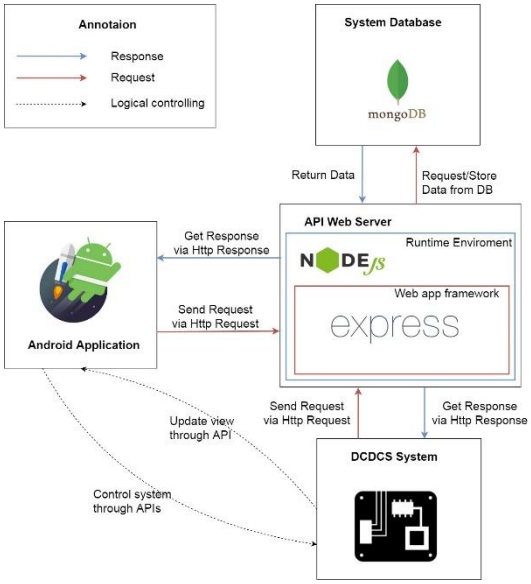

Figure 2: System overview architecture

3.2.2 API Web server architectural design

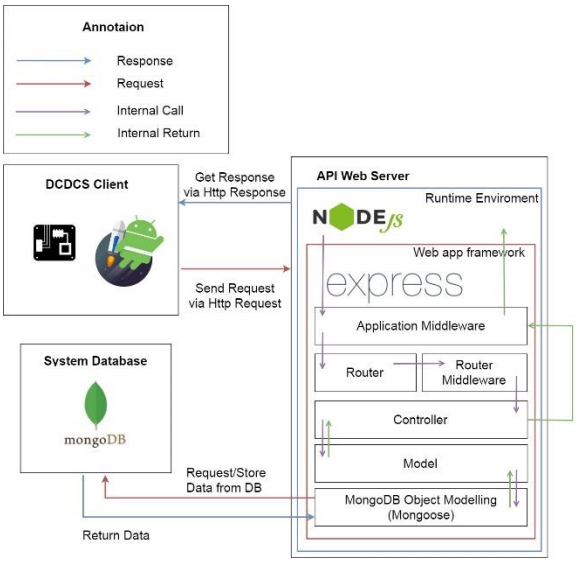

Figure 3: API Web server architectural design

In API development, the system is developed under MVC architecture style. We

choose this architecture for API because of the following advantages:

- With MVC architecture, we can separate business code with Controller and View, so we can use the business code in API web server without repeating the code.
- It can eliminate the creation of the singleton and factory classes and well-defined interface to business layers.
- By separating concerns into 3 distinct pieces, we can perform unit testing easily. Our Presentation layer can be tested free of the Model or Controller, and vice-a-versa
- It supports all aspects of application development, business aspects, persistence aspects, etc., so we can develop a complete application.

This project follows MVC architecture with the following components:

- **Controller:** the part of the application that acts like event handler to handles user interactions. Typically, the controller reads data from a request and calls the appropriate business's method then selects the view to return to the user.
- **View:** The view renders the contents of a model. It gets data from the model and specifies how that data should be presented. It updates data presentation when the model changes. A view also forwards user input to a controller. Depending on the task being performed by the user the model can be looked at from different perspectives.
- **Model:** Represents the business data and any business logic that govern access to and modification of the data. The model notifies views when it changes and lets the view query the model about its state. It also lets the controller access application functionality encapsulated by the model. Typically, when a change in the model is to be reflected from user, it should be reflected in all the model's views

3.2.3 Hardware System Architecture

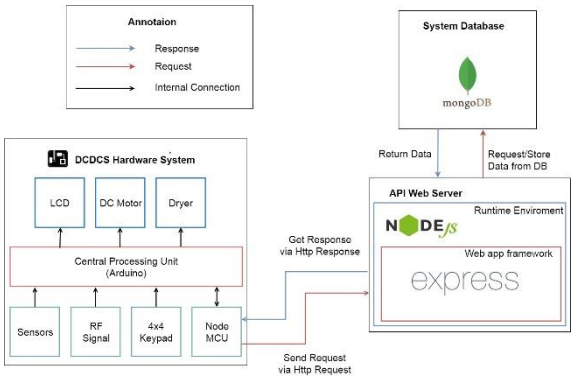

Figure 4: Hardware System Architecture

In Embedded Hardware control application, the system is developed under Internet of Things architecture style. We choose this architecture for Embedded Hardware control application because of following advantages:

- Highly scalable and available out of the box due to the nature of each selected component.
- Minimal knowledge required to start.
- It's scalable and fault tolerant by design.
- Reduces the development and deployment costs and timeframes

The system follows IoT architecture with following components:

- Sensors and Actuators: this part measures a physical quantity such as sound, temperature, moisture etc. and converts it into electrical quantity to make the system understand and act accordingly
- Connectivity (NodeMCU): The received signals are to be uploaded on the network using different communication medium such as Wi-Fi, Bluetooth or BLE, LoPAN etc.
- People and Processes: Networked inputs are then combined into bidirectional system that integrate data, people and processes for better decision making.

3.2.4 Mobile Application Architecture

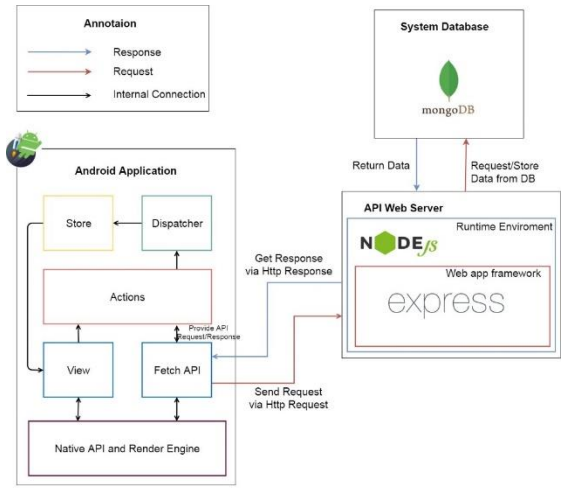

Figure 5: Mobile Application Architecture

In Android application, the system is developed under Flux architecture. We choose this architecture for Android Application because of following advantages:

- Flux is all about controlling the flow inside the app—and making it as simple to understand as possible.
- Easy to implement and understand. Hence it makes source code easier to maintain and reduce time to develop application
- Having supported library (Redux)
- Suitable for React Native codebase

Android Application follows Flux architecture with following components:

- **Actions:** Helpers that pass data to the Dispatcher. Are simple objects with a type property and some data. For example, an action could be:
  {"type": "IncreaseCount", "payload": {"delta": 1}}
- **Dispatcher:** Receives these Actions and broadcast payloads to registered callbacks. Acts as a central hub. The dispatcher processes actions (for example, user interactions) and invokes callbacks that the stores have registered with it. The dispatcher isn't the same as controllers in the MVC pattern—usually the dispatcher does not have much logic inside it and you can reuse the same dispatcher across projects
- **Stores**: Contain the application's state and logic. The best abstraction is to think of stores as managing a particular domain of the application. They aren't the same as models in MVC since models usually try to model single objects, while stores in Flux can store anything. The real work in the application is done in the Stores. The Stores registered to listen in on the actions of the Dispatcher will do accordingly and update the Views.

Views: are **controller-views**, also very common in most GUI MVC patterns. They listen for changes from the stores and re-render themselves appropriately. Views can also add new actions to the dispatcher, for example, on user interactions. The view are usually coded in React, but it's not necessary to use React with Flux.

4. ALGORITHMS AND DATA STRUCTURES

4.1 System control

4.1.1 Definition

System has many ways to control the system; i.e. RF Remote, Android application, hardware button. From these controllers, they can control many another devices like DC Motor to collect or dry clothes.

4.1.2 Define Problem

While using multiple controller at the same time. It causes a collision that leading to the system doesn't work correctly.

4.1.3 Solution

We use one thread and blocking I/O to sequentially reading each controller. Therefore, when we're handling a single controller. Another controller will be ignored.

4.1.4 Pros & Cons

- Pros:
  - No more collisions
  - Easy to control because the system now works on priority of the controller
  - Easy to extends when there are new controller
  - Memory reduced due to using only a single thread
- Cons:
  - An action takes longer time than user to complete (due to the priority)

4.1.5 Algorithm Complexity

- Time: O(n) with n is the number of controller
- Space: O(1) because we don't use any additional spaces

4.1.6 Overview Flowchart

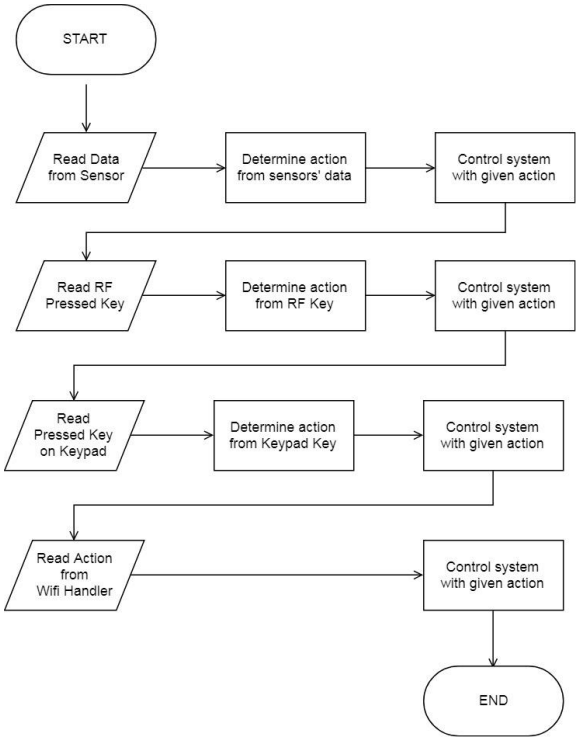

Figure 6: System Control overview flowchart

4.2 Message Queue

4.2.1 Definition

System can receive control message from multiple mobile application. Each mobile device can send multiple action including Get system data, Control DC Motor, Control dryer, etc. Also, each mobile device can control different system. This leading to a big problem about communication between mobiles and systems.

4.2.2 Define Problem

While the message is being sent to the server, the server will store it in a database. However, the order of the message is not sorted. Therefore the server doesn't know which message should be sent first and later.

4.2.3 Solution

We will build a queue for each device id, naming it as a message queue. In this queue, a message will have a priority note and timestamp along with another data. In order to send a message, the server will sort all messages in the queue by priority and then by timestamp. If 2 messages have the same priority then the one which comes earlier has higher order. If 2 messages have same priority and timestamp, they will be sent in random order. For any messages in queue longer than 30 seconds will be deleted.

4.2.4 Pros & Cons

- Pros:
  - No more collisions
  - Handling message is easier due individual device queue
  - With priority, user can force system do a certain action with very high priority
- Cons:
  - Some message unable to send due to time limitation and priority

4.2.5 Algorithm Complexity

- Time:
  - O(nlogn) for sorting algorithm.
  - O(1) for adding message to queue
  - O(1) for getting a message from queue
  - **Total:** O(nlogn)
- Space: O(n) if database use indexing for faster sorting and searching.

5. PERFORMANCES

5.1 Control System from Android App Performances

Request time performance

Network Type	System	Request Time
Wi-Fi (Ping 2, Down: 20Mb, Up: 15Mb)	Android Application	143ms
Wi-Fi (Ping 2, Down: 20Mb, Up: 15Mb)	DCDCS System	252ms
3G	Android Application	471ms

Table 1: API request performance

Response time performance

Network Type	System	Request Time
Wi-Fi	Android Application	352ms

(Ping 2, Down: 20Mb, Up: 15Mb)		
Wi-Fi (Ping 2, Down: 20Mb, Up: 15Mb)	DCDCS System	461ms
3G	Android Application	621ms

Table 2: API response performance

5.2 Control System from RF Remote/Keypad

Control Type	Distance	Response Time
RF Remote	0m	5ms
RF Remote	50m	32ms
RF Remote	100m	182ms
Keypad	0m	5ms

Table 3: RF Remote/Keypad performance

6. EVALUATION

Achievement:

- System can be controlled via mobile phone
- System can detect rain within ~50ms.
- Server can handle at least 1000 request per second.
- System is easy to construct.
- System works with 95% accuracy.

Limitations:

- Cannot determine whenever clothes is dry or not.
- Cannot determine wherever rain is stopped or not
- Cannot control via Web application

Further Suggestions

- Implement Hidden Markov Models (HMM) for rain forecasting
- Implement the ability to detect when the rain stop using HMM
- Build a website for user to check their account information and control the system along with mobile application
- Build a system that can detect whenever the clothes is dry or wet.

7. CONCLUSION

Our research of the DCDCS system shows that it has a great potential to succeed in the market. However, there are some limitations to the system such as the fact that its inability to detect rain using Mathematics process. These limitations need to be improved to make customers more willing to buy our product as well as giving the product a competitive advantage over other competitors.

ACKNOWLEDGEMENT

We would like to give a special thanks to my supervisor Mr. Nguyen Duc Loi for his professional guidance and advices during the process of researching and implementing the project. We especially appreciate his assistance of supporting devices for our research.

REFERENCES

1. Flux Architecture: https://facebook.github.io/flux/
2. How Expo Works: https://docs.expo.io/versions/latest/workflow/how-expo-works
3. React Native Mechanism Explanation: https://wetalkit.xyz/react-native-what-it-is-and-how-it-works-e2182d008f5e
4. Understanding Node.js & Express.js: https://medium.com/@LindaVivah/the-beginners-guide-understanding-node-js-express-js-fundamentals-e15493462be1
5. Understand Express: https://evanhahn.com/understanding-express/
6. The 4 stages of an IoT architecture: https://techbeacon.com/4-stages-iot-architecture