DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
THE UNIVERSITY OF TEXAS AT ARLINGTON

DETAILED DESIGN SPECIFICATION
CSE 4317: SENIOR DESIGN II
FALL 2021

THE LIGHTERS
LIGHTHOUSE

DIPIKA GIRI
DHRUV PATEL
PARKER SKINNER
DAVID TRIMINO

Revision History

Revision	Date	Author(s)	Description
0.1	09.20.2021	DG, DP, PS, DT	document creation
0.2	09.26.2021	DG, DP, PS, DT	Draft Complete

CONTENTS

List of Figures

List of Tables

1 Introduction

The overall product concept is a web server application that acts like a controller which has a multitude of functionality. The main functionality of the web server is to control the General Purpose Input/Output (GPIO) on a raspberry pi. The system will need to take in the user's customization preferences and display them through the LED lights connected to the raspberry pi as per the Customer Requirements laid out to our team(System Requirements Specification). The system should allow the user to customization bundled in a cloud repository for other people to have and share. In this document, we will specify subsystem applicable hardware, operating system, software dependencies, data structures and data processing. We will be using previous documents, Architectural Design Specification and System Requirements Specification as a reference to complete this document as the basic detailed requirements and the overview of the entire project is outlined in those two documents.

2 System Overview

The system will essentially consist of three layers: an application layer, middle-ware, and the hardware layer. The two layers Application and Middle-Ware are separate layers as both accomplish and handle different aspects although all taking place in the pi itself. The user layers function is to deal with the presentational and interactional functions in regard to a Dispatcher, Store, Action, and View. The user layer will be using a flux architecture. The middle-ware with input validation of the user, and to process the JSON objects and translate them into readable code for the LED lights. The validation step is essential to ensure the information being sent to the LED lights is valid. The hardware layer is responsible for displaying the LED lights to the user's preferences.

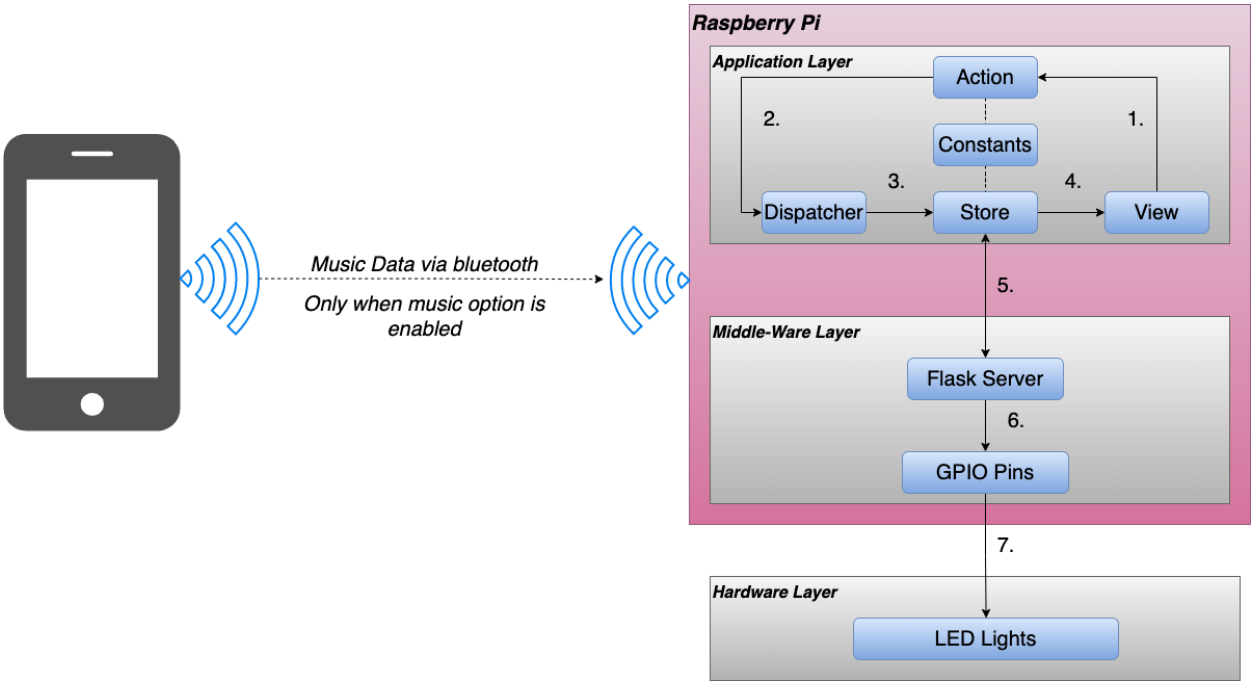

Figure 1: System Architecture

3 Application Layer Subsystems

The Application is the user interface for the user using the LightHouse Web Application and the background processes that deliver the needed information across all components and the back-end server. The layer consists of five layers: Action, Dispatcher, Store, View, and Constants. Each are responsible for updating the DOM accurately and effectively to give the user an excellent experience.

3.1 Layer Hardware

Not applicable for this layer.

3.2 Layer Operating System

It works on any operating system (Linux, Windows, MacOS).

3.3 Layer Software Dependencies

A few libraries we will be using are:

- Events Emitter Library
- React Router Library
- React Color Library
- Cypress (End-to-End Testing)

3.4 Action - Subsystem 1

According to the flux documentation: Action creators are helper methods, collected into a library, that create an action from method parameters, assign it to a type and provide it to a dispatcher. In other words, the dispatcher method allows us to trigger a dispatch to the store and include a payload of data, which we call an action. It is an action creator or helper methods that pass the data to the dispatcher.

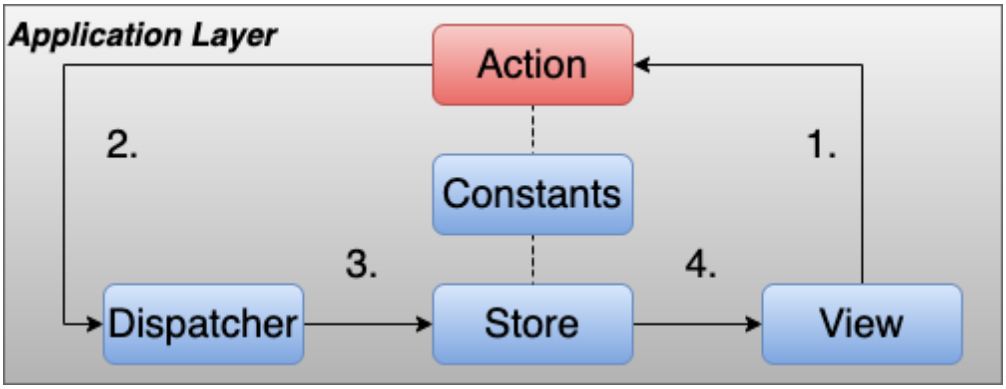

Figure 2: Action Subsystem Description Diagram

3.4.1 Subsystem Hardware

Not applicable for this layer.

3.4.2 Subsystem Operating System

It works on any operating system (Linux, Windows, MacOS).

3.4.3 Subsystem Software Dependencies

A description of any software dependencies (libraries, frameworks, design software for mechanical parts or circuits, etc) required by the subsystem.

3.4.4 Subsystem Programming Languages

React programming will be used.

3.4.5 Subsystem Data Structures

The following will be the format of an action object:

```
dispatcher.dispatch({
  type: "CREATE_LED",
  id: id,
});
```

3.4.6 Subsystem Data Processing

Not applicable at the moment.

3.5 Dispatcher - Subsystem 2

Views are really just React components that listen to change events and retrieve Application state from Stores. They then pass that data down to their child components via props.

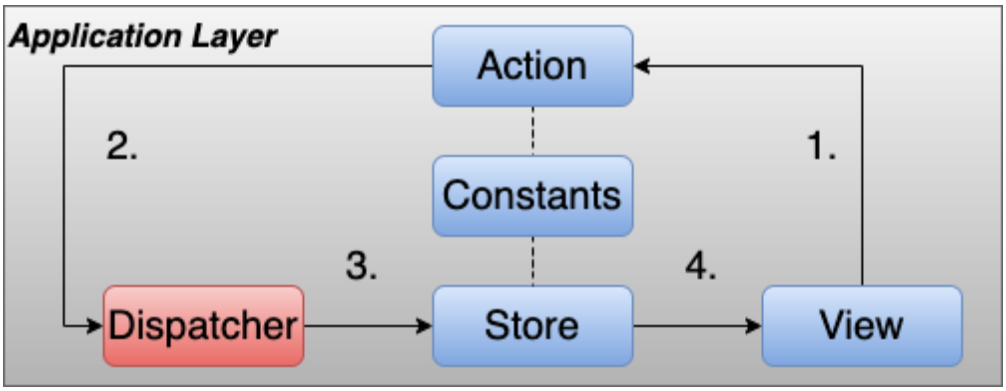

Figure 3: Dispatcher Subsystem Description Diagram

3.5.1 Subsystem Hardware

Not applicable for this layer.

3.5.2 Subsystem Operating System

It works on any operating system (Linux, Windows, MacOS).

3.5.3 Subsystem Software Dependencies

Not applicable for this subsystem.

3.5.4 Subsystem Programming Languages

React programming will be used.

3.5.5 Subsystem Data Structures

The following will be the format of an action object:

```
dispatcher.dispatch({
  type: "CREATE_LED",
  id: id,
});
```

3.5.6 Subsystem Data Processing

Not applicable at the moment.

3.6 Store - Subsystem 3

According to the flux documentation: Stores contain application state and logic. Stores manage application state for a particular domain within your application. From a high level, this basically means that per app section, stores manage the data, data retrieval methods and dispatcher callbacks.

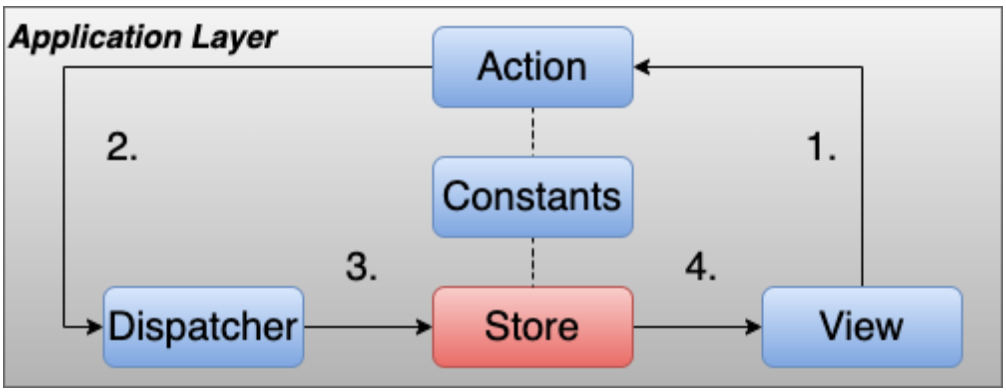

Figure 4: Store Subsystem Description Diagram

3.6.1 Subsystem Hardware

Not applicable for this layer.

3.6.2 Subsystem Operating System

It works on any operating system (Linux, Windows, MacOS).

3.6.3 Subsystem Software Dependencies

Not applicable for this subsystem.

3.6.4 Subsystem Programming Languages

React programming will be used.

3.6.5 Subsystem Data Structures

The following is an example implementation of a TODOStore Class:

```
class TodoStore extends EventEmitter {
  constructor() {
    super()
    this.todos = [
```

```
      {
        id: 113464613,
        text: "Go Shopping",
        complete: false
      },
      {
        id: 235684679,
        text: "Pay Water Bill",
        complete: false
      },
    ];
  }

  createTodo(text) {
    const id = Date.now();

    this.todos.push({
      id,
      text,
      complete: false,
    });

    this.emit("change");
  }
  }
```

3.6.6 Subsystem Data Processing

Not applicable at the moment.

3.7 View - Subsystem 4

According to the flux documentation: Stores contain application state and logic. Stores manage application state for a particular domain within your application. From a high level, this basically means that per app section, stores manage the data, data retrieval methods and dispatcher callbacks. Is a class or function when it is a class it will extend React.Component

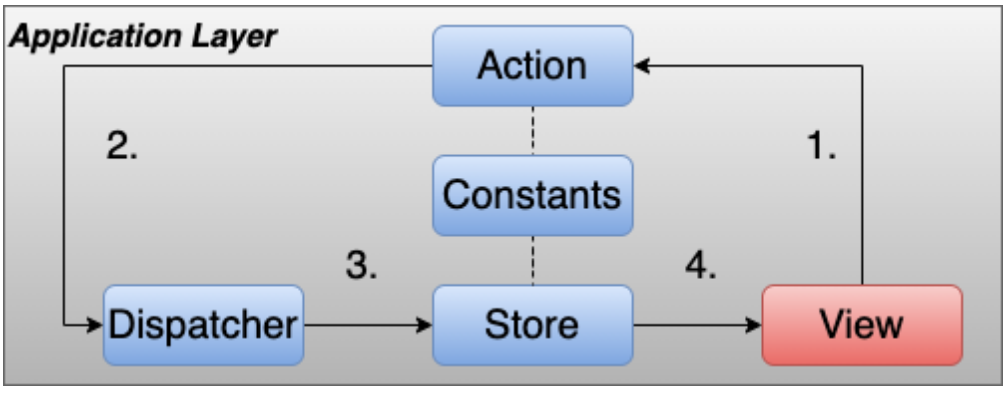

Figure 5: View Subsystem Description Diagram

3.7.1 Subsystem Hardware

Not applicable for this layer.

3.7.2 Subsystem Operating System

It works on any operating system (Linux, Windows, MacOS).

3.7.3 Subsystem Software Dependencies

Not applicable for this subsystem.

3.7.4 Subsystem Programming Languages

React programming will be used.

3.7.5 Subsystem Data Structures

```
export default class Todo extends React.Component {
  constructor(props) {
    super();
  }

  render() {
    const { complete, edit, text } = this.props;

    if (edit) {
      return (
        <li>
          <input value={text} focus="focused"/>
        </li>
      );
    }

    return (
      <li>
        <span>{text}</span>
        <span>{icon}</span>
      </li>
    );
  }
}
```

3.7.6 Subsystem Data Processing

Not applicable at the moment.

4 Middle-Ware Layer Subsystems

The Middle-Ware contains Flask server to build Rest APIs. The layer connects with GPIO pins to send I/O functional signal to the LED lights.

4.1 Layer Hardware

This layer uses GPIO pins in the Raspberry Pi.

4.2 Layer Operating System

This layer will interact with the Raspberry Pi's OS.

4.3 Layer Software Dependencies

The layer uses Flask micro web framework.

4.4 Flask Server - Subsystem 1

The flask server is a web framework that allows us to communicate between the application layer and the middle-ware layer in order to pass along commands to the GPIO Pins and back to the store.

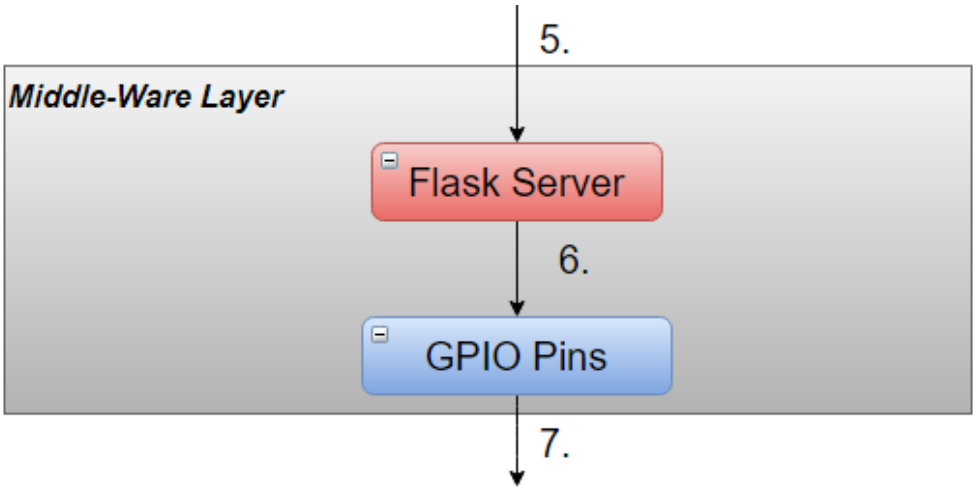

Figure 6: Flask Server Subsystem Description Diagram

4.4.1 Subsystem Hardware

The Flask Server will run on the Raspberry Pi

4.4.2 Subsystem Operating System

No OS requirements for the Flask Server.

4.4.3 Subsystem Software Dependencies

The subsystem requires Flask framework.

4.4.4 Subsystem Programming Languages

Python language will be utilized.

4.4.5 Subsystem Data Structures

A data structure called pin model will be used to contain the Pin ID, Number, color, and state

4.4.6 Subsystem Data Processing

Not applicable at the moment.

4.5 GPIO Pins - Subsystem 2

The GPIO pins will be responsible for the transmitting of bits to the LED lights so they can be displayed correctly as per the user preference. There are a total of 28 GPIO pins on the raspberry pi and can be used to manage and control stuff from the outside world such as LED lights, motors etc. by connecting to electrical circuits.

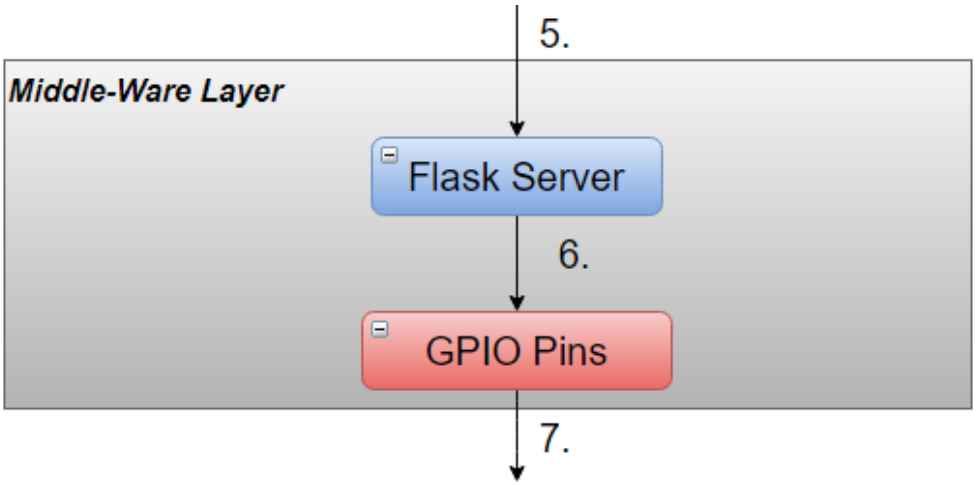

Figure 7: GPIO Pins Subsystem Description Diagram

4.5.1 Subsystem Hardware

Raspberry Pi and GPIO Pins.

4.5.2 Subsystem Operating System

Raspberry OS (32-bit).

4.5.3 Subsystem Software Dependencies

The dependencies for the GPIO pins are the library for them which is RPi.GPIO.

4.5.4 Subsystem Programming Languages

The programming language that will be used is Python.

4.5.5 Subsystem Data Structures

The GPIO pins transmit data through the Raspberry Pi's general-purpose input/output library. Calls to this library can set the state of each pin.

4.5.6 Subsystem Data Processing

Not applicable at the moment.

5 Hardware Layer Subsystems

In this hardware layer we have a single subsystem which is the the WS281B LED Lights. This layer is solely responsible for the LED lights and the lighting of them. The arrow shown number 7 comes from the Raspberry Pi Middle-Ware Layer with commands on how the LEDs should be lit up.

5.1 Layer Hardware

The hardware present in this layer simply includes the WS281B LED lights which will receive commands from the GPIO pins to light up in specific ways and patterns. The lights are RGB so different combinations can be given from the different HEX codes.

5.2 Layer Operating System

Not applicable for this system as they are the LED lights and they will be receiving information from the pi.

5.3 Layer Software Dependencies

The dependencies for this layer are the GPIO pins outputs and proper power supply to light the LEDs.

5.4 WS281B LED Lights

The WS281B subsystem simply contains the LED lights that the whole product is based upon. The lights receive power from the power supply subsystem and inputs on how to light up from the raspberry pi and ESP2866 layers. This piece of hardware will display the requested patterns and designs and bring to life what the user wants. This lights will be set up in a strip currently to display the current patterns.

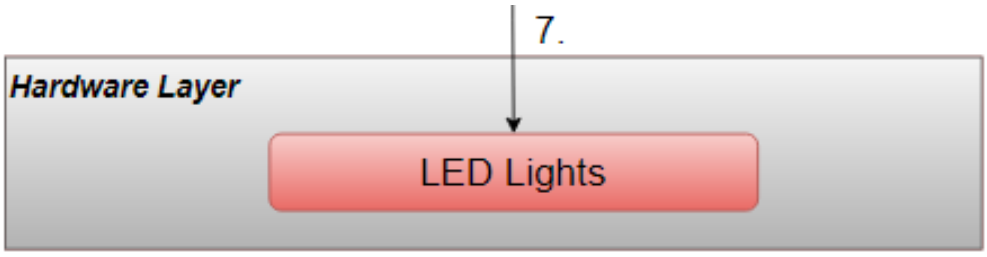

Figure 8: LED Lights Subsystem Description Diagram

5.4.1 Subsystem Hardware

The hardware necessary in this layer are just the power supply and more importantly the LED lights.

5.4.2 Subsystem Operating System

Not applicable for this layer.

5.4.3 Subsystem Software Dependencies

Only dependencies are from the GPIO pins with the patterns and the power from the power supply.

5.4.4 Subsystem Programming Languages

Not applicable for this layer.

5.4.5 Subsystem Data Structures

Bits will be transferred from the GPIO pins on the raspberry pi over to the LED lights. None other applicable for this layer.

5.4.6 Subsystem Data Processing

Not applicable in this layer.

6 Appendix A

None at this moment.

REFERENCES

LightHouse Team, "System Requirements Specification", University of Texas at Arlington

LightHouse Team, "Architectural Design Specification", University of Texas at Arlington