A Comprehensive Implementation Guide for Custom LED Animation Systems

Part I: The 64x64 HUB75 Matrix Display System

This first part of the guide is dedicated to establishing the foundational hardware and software for the primary display system. It details the assembly of a 64x64 HUB75 LED matrix panel controlled by a Raspberry Pi 3 B+ via an Adafruit RGB Matrix HAT. The focus is on meticulous assembly and power management, which are the cornerstones of a stable, high-performance display, followed by the creation of an optimized software environment.

Chapter 1: Assembling and Powering the Core Hardware

The physical construction of the display system is the first and most critical phase. Success in this project is overwhelmingly determined by the integrity of the power delivery network and the quality of the electrical connections. Intermittent flickering, color inaccuracies, and system crashes are far more likely to originate from electrical issues than from software bugs. This chapter provides a detailed walkthrough of the physical assembly, treating power architecture as the primary engineering concern.

1.1 Component and Tool Checklist

A successful build begins with the correct components and tools. The following table outlines the necessary items, with specific recommendations and justifications to ensure system reliability and performance from the outset.

Category	Item	Recommended Specification / Model	Justification
Controller	Single-Board Computer	Raspberry Pi 3 Model B+	Provides a balance of processing power, 40-pin GPIO header compatibility, and robust community support for this application.[1]
Interface	Matrix Driver HAT	Adafruit RGB Matrix HAT + RTC (Product 2345)	Simplifies wiring complexity by providing onboard level shifters (3.3V to 5V), power protection circuitry, and a direct HUB75 connector. The RTC is a useful bonus for time-based displays.[3]
Display	LED Matrix Panel	64x64 RGB LED Matrix with HUB75 connection	The target display for this project. Ensure it is a HUB75 type, as this HAT is not compatible with addressable LEDs like NeoPixels or

			DotStars.[3]
Power	Main Power Supply	5V DC, 10A	A 64x64 panel can theoretically draw over 7.5A at full brightness.[5] A 10A supply provides essential headroom, preventing voltage drops that cause flicker and system instability. A 4A supply is insufficient for this panel size.[7]
	Raspberry Pi Power	Official Raspberry Pi 5.1V 2.5A MicroUSB Supply	Powering the Pi separately from the matrix is critical to prevent the panel's high current draw from causing voltage sags that would crash the Pi.[1]
Storage	microSD Card	32 GB, Application Class 2 (A2)	An A2-rated card offers better performance for the small, random read/write operations typical of an operating system, leading to a more responsive system.[11]
Tools	Soldering Iron & Solder	Temperature-controlled iron with a fine tip	Required for assembling the HAT headers and performing the mandatory 64x64 panel modification. Good soldering technique is crucial for reliable high-frequency signal connections.
	Wire Strippers/Cutters	Standard electronics toolkit	For preparing the power wires for the HAT's terminal block.[12]
	Multimeter	Basic digital multimeter	Highly recommended for verifying power supply voltage and checking for continuity after soldering.

1.2 Assembling the Adafruit RGB Matrix HAT

The Adafruit RGB Matrix HAT (product 2345) requires some light soldering to attach the connectors. This process is straightforward but must be done carefully to ensure all connections are solid.[1]

1. **Solder the 2x20 GPIO Socket Header:** To keep the header stable, it is helpful to first place it onto the GPIO pins of a powered-down Raspberry Pi. Then, place the HAT PCB on top, aligning the short pins of the header with the corresponding pads on the HAT. Solder one pin at each corner first, then check that the HAT is sitting level and parallel to the Pi. Once aligned, proceed to solder the remaining pins.[1] Each solder joint should be shiny and conical, indicating a good connection.
2. **Solder the 2x8 IDC Header:** This is the connector for the matrix data cable. It is keyed with a small notch to prevent the cable from being inserted backward. This notch should be oriented to match the silkscreen outline on the HAT PCB. Solder all 16 pins securely.
3. **Solder the 2-pin Terminal Block:** This block is for the main 5V power input for the matrix. It should be placed so that the wire entry points face outward from the board. Solder the two large pins to the board.[13]

1.3 The Critical 64x64 Panel Modification: Bridging the 'E' Address Line

This hardware modification on the Adafruit HAT is mandatory for use with a 64x64 panel and is a common point of failure if

missed. Standard 32x32 panels use a 1:16 scan multiplexing scheme, which requires four address lines (A, B, C, D) to select one of the 16 rows to illuminate at any given moment. However, 64x64 panels typically use a 1:32 scan, which requires a fifth address line, designated as 'E'.[15]

The Adafruit RGB Matrix HAT (Rev C and later) includes unpopulated solder pads to accommodate this requirement. To enable 64x64 support, a solder bridge must be created.

1. Locate the solder jumpers on the back of the HAT, typically labeled with 'E' in the center and '8' and '16' on either side.[1]
2. Using a soldering iron, apply a small blob of solder to connect the center 'E' pad to the pad labeled '8'. This specific connection is required for panels sold by Adafruit and many other common manufacturers.[1] This bridge routes the appropriate Raspberry Pi GPIO pin to the 'E' address line on the HUB75 connector, enabling the library to correctly address all 32 multiplexed row pairs.

Failure to perform this step will result in the display showing a garbled or duplicated image, as the control software will be unable to address the bottom half of the panel correctly.

1.4 Power Architecture: Ensuring System Stability

A stable power delivery network is the single most important factor for a flicker-free, reliable matrix display. The high number of LEDs on a 64x64 panel (4096 LEDs) can draw a surprisingly large amount of current, especially when displaying bright, white content.

- **Power Calculation:** The maximum current draw can be estimated. A common rule of thumb is to multiply the panel width by 0.12 amps.[2] For a 64-pixel wide panel, this yields $64 \times 0.12A = 7.68A$. Another method calculates per-pixel draw: each of the 4096 pixels can draw up to 60 mA (20 mA each for red, green, and blue), but due to multiplexing, only a fraction of the rows are on at any given time. Even with a 1/32 duty cycle, the instantaneous current draw can be very high. These calculations demonstrate that a 5V 4A power supply is insufficient, and a 5V 10A supply is a much safer and more reliable choice.[5]
- **Wiring Best Practices:** It is imperative to power the Raspberry Pi and the LED matrix separately. The Pi should be powered via its own dedicated MicroUSB power supply. The 5V 10A supply should be connected directly to the 2-pin terminal block on the Adafruit HAT.[1] While the HAT includes circuitry that can back-power the Pi, relying on this is not recommended. The panel's fluctuating, high current demands can introduce noise and voltage drops onto the power rail, which can easily lead to instability or crashes of the Raspberry Pi if they share the same primary power source.[10]

1.5 Final Assembly and Wiring Connections

With the HAT assembled and the power architecture planned, the final connections can be made.

1. **Mount the HAT:** With the Raspberry Pi powered off, carefully align the 2x20 socket on the HAT with the 40 GPIO pins on the Pi and press down firmly and evenly until it is fully seated.[9]
2. **Connect Matrix Power:** Connect the power cable from the 5V 10A supply to the 2-pin terminal block on the HAT. Ensure the positive wire goes to the '+' terminal and the negative (ground) wire goes to the '-' terminal.[9]
3. **Connect Matrix Data:** Take the 16-pin (2x8) IDC ribbon cable. Connect one end to the 2x8 header on the HAT. Connect the other end to the HUB75 connector labeled "INPUT" or indicated by arrows pointing away from it on the back of the LED panel.[9] Connecting to the "OUTPUT" port will not damage the panel, but it will not work.
4. **Power On Sequence:** First, power on the Raspberry Pi using its MicroUSB power supply and allow it to boot. Once the Pi is running, connect the 5V 10A power supply for the matrix to mains power. A green LED on the HAT should illuminate, indicating it has power.[9] The panel will remain dark until the control software is run.

Chapter 2: Establishing the Software Foundation

With the hardware assembled, the next stage is to prepare the Raspberry Pi's operating system and install the core software library that will control the matrix. This process is an exercise in eliminating potential resource conflicts. The goal is to create a lean, optimized environment that dedicates the Pi's processing capabilities to the demanding, real-time task of driving the display.

2.1 Preparing the Raspberry Pi OS

The choice of operating system distribution is a critical first step. For a project that requires precise hardware timing, a minimal system is always preferable to a full desktop environment.

1. **Download and Use Raspberry Pi Imager:** The official Raspberry Pi Imager tool is the recommended method for flashing the OS image to the microSD card.[18]
2. **Select Raspberry Pi OS Lite (64-bit):** It is crucial to select the "Lite" version of the OS. The full desktop version runs a graphical user interface and numerous background services that consume CPU cycles and can interfere with the hardware timers needed by the matrix library. This interference is a direct cause of visual artifacts like flicker.[5] The Lite version provides a minimal command-line environment, maximizing available resources for the display task.
3. **Pre-configure for Headless Operation:** Before writing the image, use the Imager's advanced options (accessed by clicking the gear icon or pressing Ctrl+Shift+X) to pre-configure the system. Set a hostname, create a user account with a strong password, enable SSH, and configure the WiFi credentials.[18] This allows the Raspberry Pi to be accessed remotely over the network via SSH from another computer, eliminating the need for a dedicated monitor and keyboard after the initial setup.

2.2 Installing the rpi-rgb-led-matrix Library

The heart of this project is Henner Zeller's rpi-rgb-led-matrix library, a highly optimized C++ library with Python bindings for controlling HUB75 panels.[21] The installation involves cloning the source code from its repository and compiling it on the Pi.

1. **Connect and Update:** Insert the flashed microSD card into the Pi, power it on, and connect to it via SSH. First, ensure the system's package lists and installed software are up to date:
   Bash
   ```
   sudo apt update
   sudo apt upgrade -y
   ```
   [22]
2. **Install Dependencies:** Several packages are required to build the library and its utilities. Install them using apt:
   Bash
   ```
   sudo apt install -y git python3-dev python3-pil libgraphicsmagick++-dev libwebp-dev
   ```
   This command installs git for cloning the repository, Python 3 development headers and the Pillow imaging library for the Python bindings, and GraphicsMagick/WebP development libraries for the image and GIF viewing utilities.[21]
3. **Clone the Repository:** Use git to download the latest version of the library source code into the user's home directory:
   Bash
   ```
   cd ~
   git clone https://github.com/hzeller/rpi-rgb-led-matrix.git
   ```

21

4. **Compile the Library:** The library is split into multiple parts that must be compiled from the source code.
    - **Compile C++ Core and Utilities:** Navigate into the utils directory within the cloned repository and run make. This builds essential command-line tools like led-image-viewer.
      Bash
      ```
      cd ~/rpi-rgb-led-matrix/utils
      make
      ```
      25
    - **Compile Python Bindings:** Navigate to the Python bindings directory and run make to build the rgbmatrix.so module that allows Python scripts to interface with the C++ library.
      Bash
      ```
      cd../bindings/python
      make
      ```
      21

2.3 Critical System Optimizations for Performance

The rpi-rgb-led-matrix library achieves its high performance by directly manipulating the Raspberry Pi's hardware timers and GPIO pins. To prevent conflicts, certain default system features that use these same resources must be disabled.

- **Disable Onboard Audio:** The Raspberry Pi's onboard audio hardware uses the same underlying PWM and DMA controllers that the matrix library requires for its most stable timing modes. Leaving audio enabled is a common cause of severe flickering. To disable it, edit the system configuration file:
  Bash
  ```
  sudo nano /boot/config.txt
  ```
  Add the following line to the end of the file:
  ```
  dtparam=audio=off
  ```
  Save the file (Ctrl+X, then Y, then Enter). This change will take effect after the next reboot.[5]
- **Disable 1-Wire Interface:** The 1-wire interface, if enabled (it is typically off by default), can also reserve GPIO pins. Ensure it is disabled using the raspi-config utility or by checking for a dtoverlay=w1-gpio line in /boot/config.txt and commenting it out.[16]

2.4 Initial Hardware and Software Verification

Before proceeding to custom application development, it is essential to run a baseline test to confirm that the hardware assembly, power delivery, solder modifications, and software installation have all been successful. The compiled demo programs are perfect for this verification.

The following table provides the exact command and an explanation of the flags required to test the specific 64x64 panel and Adafruit HAT setup.

Flag	Value	Purpose

sudo./demo -D 1	runtext.ppm	Runs demo #1 (scrolling image) using the provided sample image file. Requires sudo for hardware access.
--led-rows	64	Informs the library that the panel has 64 rows.[15]
--led-cols	64	Informs the library that the panel has 64 columns.[15]
--led-gpio-mapping	adafruit-hat	**Crucial:** Selects the specific GPIO pinout used by the Adafruit HAT, remapping the library's default outputs to match the HAT's hardware design.[15]
--led-slowdown-gpio	2	Slows down the GPIO writing speed. The Raspberry Pi 3 can be too fast for some panels, causing data corruption (artifacts). A value of 2 is a good starting point for this model.[15]
--led-no-hardware-pulse	(none)	Disables a specific timing feature that can sometimes be unstable with the HAT. It is a good option to include for initial testing.[21]

To run the test, navigate to the examples-api-use directory and execute the full command:

Bash

```
cd ~/rpi-rgb-led-matrix/examples-api-use
sudo./demo -D 1 runtext.ppm --led-rows=64 --led-cols=64 --led-gpio-mapping=adafruit-hat --led-slowdown-gpio=2
--led-no-hardware-pulse
```

If the system is correctly configured, a sample image with text will scroll smoothly across the 64x64 panel. If the display is garbled, flickers excessively, or shows nothing, revisit the steps in Chapter 1, paying close attention to the power supply, the 'E' address line solder bridge, and all data/power cable connections.

Part II: Building the Networked Control Interface

With a confirmed working hardware and driver foundation, the project now transitions to creating a sophisticated and user-friendly control system. This part details the development of a Python-based animation engine and a network-accessible web application built with the Flask framework. This will allow for remote management of animations, file uploads, and parameter adjustments from any device on the local network.

Chapter 3: Bringing the Matrix to Life with Animations

This chapter moves beyond simple tests to develop a flexible software engine capable of playing various types of animations. This engine will form the core of the display logic, which will later be controlled by the web interface.

3.1 Understanding the Display Methods

The rpi-rgb-led-matrix library offers several ways to display content, each suited to different use cases.

- **Method 1: Command-Line Utilities:** The simplest method is to use the pre-compiled utilities. The led-image-viewer tool, built in the previous chapter, is capable of displaying a wide range of static image formats (JPEG, PNG, etc.) as well as animated GIFs directly from the terminal. This is excellent for quick tests or simple, scripted slideshows but lacks the dynamic control needed for a web application.[25]
  Bash
  ```
  # Example: Display an animated GIF, looping indefinitely
  sudo./led-image-viewer my_animation.gif --led-rows=64 --led-cols=64 --led-gpio-mapping=adafruit-hat -f
  ```

- **Method 2: Python Bindings:** For programmatic control, the library's Python bindings are the ideal solution. They expose the core functionality of the C++ library to Python scripts. The primary classes used are RGBMatrix and RGBMatrixOptions, which allow for the creation of a matrix object and the configuration of all the same parameters (rows, columns, GPIO mapping, etc.) that were previously passed as command-line flags.[27] This method is the foundation of the custom control system.
- **Method 3: Video Playback:** Real-time decoding and playback of video files (like MP4) is computationally intensive and can overwhelm the Raspberry Pi 3's CPU, leading to poor performance and severe display flicker if attempted directly alongside matrix driving.[29] A more effective technique is framebuffer mirroring. This involves using a capable video player like vlc to play the video to the Pi's internal video buffer (the framebuffer), as if it were displaying on an HDMI monitor. A separate Python script then continuously reads pixel data from this framebuffer, resizes it, and draws it to the LED matrix.[30] While this guide focuses on image and GIF animation, this method provides a viable path for future expansion into video playback.

3.2 Developing a Python Animation Engine

To create a robust system, the animation playback logic must be separated from the web server logic. A common mistake is to place the animation loop directly inside a web request handler, which would block the entire web server, making it unresponsive. The correct approach is to create a standalone Python script, animation_player.py, that runs as an independent, persistent background process. The web application will then start, stop, and communicate with this process.

This script will be designed to accept command-line arguments to specify which animation to play and with what settings.

- **Core Structure and Double Buffering:** The animation script will utilize a technique called double buffering to ensure smooth, tear-free animations. Instead of drawing directly to the visible display, graphics are rendered to an off-screen buffer (a "canvas"). Once a frame is complete, this buffer is atomically swapped with the visible one during the display's vertical sync period. This is accomplished with the matrix.CreateFrameCanvas() and matrix.SwapOnVSync() methods.[32]
- **Animated GIF Playback Logic:** The most common use case will be playing animated GIFs. This can be achieved using the Python Imaging Library (Pillow). The logic involves opening the GIF file, iterating through each of its frames in a loop, pasting each frame onto the off-screen canvas, and then swapping the canvas to the display.
  A simplified example of this logic within animation_player.py:
  Python
  ```
  from rgbmatrix import RGBMatrix, RGBMatrixOptions
  from PIL import Image
  import time
  import sys

  #... (Argument parsing for image_path, brightness, etc.)...

  # Configuration
  ```

```
options = RGBMatrixOptions()
options.rows = 64
options.cols = 64
options.gpio_mapping = "adafruit-hat"
options.led_slowdown_gpio = 2
#... other options...
matrix = RGBMatrix(options=options)
matrix.brightness = brightness # Set initial brightness

image = Image.open(image_path)
canvas = matrix.CreateFrameCanvas()

# Main animation loop
while True:
    for frame in range(0, image.n_frames):
        image.seek(frame)
        # Resize frame to fit matrix and paste onto canvas
        frame_resized = image.convert("RGB").copy()
        frame_resized.thumbnail((matrix.width, matrix.height), Image.Resampling.LANCZOS)
        canvas.SetImage(frame_resized, 0, 0)
        canvas = matrix.SwapOnVSync(canvas)
        time.sleep(image.info.get('duration', 100) / 1000.0) # Respect GIF's frame duration
```

34

This script forms a self-contained animation player. The web application's role will be to launch this script with the appropriate arguments (e.g., python3 animation_player.py --image /path/to/file.gif --brightness 75) and to manage its lifecycle.

Chapter 4: Architecting the Web Application with Flask

With a functional animation engine, the next step is to build the web-based graphical user interface (GUI) to control it. This requires a backend framework to handle web requests and interact with the system.

4.1 Framework Showdown: Flask vs. Node.js

For a project of this nature, both Python-based Flask and JavaScript-based Node.js (with a framework like Express) are viable options. However, for this specific use case on a Raspberry Pi, Flask presents several distinct advantages:

- **Simplicity and Lightweight Nature:** Flask is a "micro-framework," meaning it provides the essentials for web development without imposing a large, complex structure or consuming significant system resources. This is ideal for a resource-constrained device like the Raspberry Pi 3.[36]
- **Python Ecosystem Integration:** The rpi-rgb-led-matrix library has robust Python bindings. Using Flask keeps the entire software stack within the Python ecosystem, simplifying development, dependency management, and the interaction between the web server and the matrix control code.
- **Lower Learning Curve:** For developers already familiar with Python, Flask is exceptionally easy to learn and use for building simple-to-moderately complex web applications and APIs.[37]

While Node.js excels in high-concurrency, real-time applications, its asynchronous programming model can add complexity. For a control panel that handles relatively infrequent user interactions, Flask's synchronous, straightforward model is more than sufficient and easier to implement correctly.[39]

4.2 Project Structure for Maintainability

A well-organized project structure is essential for managing the different components of the web application. The following directory layout separates concerns and makes the project easier to understand and maintain:

```
/led_matrix_controller/
├── app.py                # Main Flask application file
├── animation_player.py   # The standalone animation engine
├── templates/
│   └── index.html        # The HTML file for the user interface
├── static/
│   ├── css/
│   │   └── style.css     # CSS stylesheets
│   └── js/
│       └── main.js       # JavaScript for interactivity
└── uploads/              # Directory for user-uploaded animation files
```

- app.py: Contains all the Flask routes and backend logic for handling HTTP requests.
- animation_player.py: The decoupled animation process developed in the previous chapter.
- templates/: A standard Flask directory for storing Jinja2 HTML templates.[40]
- static/: A standard Flask directory for serving static assets like CSS, JavaScript, and images that the browser will need.[40]
- uploads/: A dedicated folder to store animation files uploaded by the user. This directory must be secured and managed by the Flask application.[43]

Chapter 5: Developing the Flask Backend

The Flask backend is the brain of the operation. It receives commands from the web interface, manages animation files, and controls the animation_player.py process.

5.1 Setting Up the Basic Flask Server

The core of the backend is the app.py file. It begins with a minimal setup to create and run the Flask application.

Python

```
# app.py
from flask import Flask, render_template, request, jsonify, redirect, url_for
import os
import subprocess
```

```
from werkzeug.utils import secure_filename

# Configuration
UPLOAD_FOLDER = 'uploads'
ALLOWED_EXTENSIONS = {'gif', 'png', 'jpg', 'jpeg'}

app = Flask(__name__)
app.config = UPLOAD_FOLDER

# Global variable to hold the animation process
animation_process = None

# Main route to serve the HTML page
@app.route('/')
def index():
    return render_template('index.html')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

44

The host='0.0.0.0' argument is essential; it tells Flask to listen on all network interfaces, making the web server accessible from other devices on the local network, not just from the Raspberry Pi itself.44

5.2 Implementing API Endpoints for Control

The web interface will communicate with the backend through a set of Application Programming Interface (API) endpoints. These are simply Flask routes that are designed to be called by JavaScript and typically return data in JSON format.

- **Starting and Stopping Animations:** The core control logic involves managing the animation_player.py subprocess.

```
Python
@app.route('/api/play', methods=)
def play_animation():
    global animation_process
    # Stop any currently running animation
    if animation_process:
        animation_process.terminate()
        animation_process.wait()

    data = request.json
    filename = data.get('filename')
    brightness = data.get('brightness', 75)

    if not filename:
        return jsonify({'error': 'No filename provided'}), 400

    filepath = os.path.join(app.config, filename)
    if not os.path.exists(filepath):
        return jsonify({'error': 'File not found'}), 404

    command = [
        'python3', 'animation_player.py',
        '--image', filepath,
        '--brightness', str(brightness)
    ]
```

```
    animation_process = subprocess.Popen(command)
    return jsonify({'status': 'playing', 'file': filename})

@app.route('/api/stop', methods=)
def stop_animation():
    global animation_process
    if animation_process:
        animation_process.terminate()
        animation_process.wait()
        animation_process = None
    return jsonify({'status': 'stopped'})
```

- **Managing Files:** An endpoint is needed to allow the frontend to query the list of available animations.

```
Python
@app.route('/api/get_files', methods=)
def get_files():
    files =) if os.path.isfile(os.path.join(app.config, f))]
    return jsonify(files)
```

5.3 Handling Animation File Uploads

A secure file upload mechanism is a critical feature. This involves creating a route that accepts a file via a POST request, validates it, and saves it to the uploads directory.

```
Python

def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1).lower() in ALLOWED_EXTENSIONS

@app.route('/upload', methods=)
def upload_file():
    if 'file' not in request.files:
        return redirect(request.url)
    file = request.files['file']
    if file.filename == '':
        return redirect(request.url)
    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        file.save(os.path.join(app.config, filename))
        return redirect(url_for('index'))
    return redirect(request.url)
```

43

This implementation follows best practices:

1. It checks that a file was actually included in the request.
2. The allowed_file function validates that the file has an approved extension, preventing the upload of potentially executable or harmful files.[43]
3. The secure_filename() function from Werkzeug (a core Flask dependency) is used to sanitize the filename, removing any

directory traversal characters (like ../) or other characters that could be used to manipulate the filesystem.[43]

Chapter 6: Designing an Intuitive Frontend GUI

The frontend is the user-facing component of the system, running entirely within the web browser. It consists of HTML for structure, CSS for styling, and JavaScript for interactivity and communication with the Flask backend.

6.1 Structuring the Interface with HTML

The templates/index.html file defines the layout of the control panel. It will contain several key sections.

- **File Upload Form:** A standard HTML form is used for file uploads. The enctype="multipart/form-data" attribute is mandatory for file transfers.[43]

```
HTML
<h2>Upload Animation</h2>
<form method="post" action="/upload" enctype="multipart/form-data">
  <input type="file" name="file">
  <input type="submit" value="Upload">
</form>
```

- **Playback Control Panel:** This section contains the interactive elements for controlling the display.

```
HTML
<h2>Playback Control</h2>
<div>
  <label for="file-select">Animation:</label>
  <select name="animations" id="file-select">
    </select>
</div>
<div>
  <label for="brightness-slider">Brightness:</label>
  <input type="range" id="brightness-slider" min="1" max="100" value="75">
  <span id="brightness-value">75</span>%
</div>
<div>
  <button id="play-button">Play</button>
  <button id="stop-button">Stop</button>
</div>
```

51

6.2 Styling with CSS

A basic stylesheet in static/css/style.css will be used to provide a clean and organized layout. The focus is on usability rather than elaborate design. This would include simple rules for spacing, font sizes, and button appearance to make the interface intuitive.

6.3 Adding Interactivity with JavaScript

The static/js/main.js file is where the GUI comes to life. It handles all user interactions and communicates with the Flask backend API asynchronously using the fetch API, ensuring the webpage never needs to reload.

- **Populating the File List on Load:** When the page first loads, a GET request is made to /api/get_files. The returned JSON array of filenames is then used to dynamically create <option> elements for the dropdown menu.

```
JavaScript
// In main.js
document.addEventListener('DOMContentLoaded', function() {
    const fileSelect = document.getElementById('file-select');
    fetch('/api/get_files')
        .then(response => response.json())
        .then(files => {
            files.forEach(file => {
                let option = new Option(file, file);
                fileSelect.add(option);
            });
        });
    //... other event listeners...
});
```

54

- **Handling Control Actions:** Event listeners are attached to the buttons and the brightness slider. When an event occurs (like a button click), a function is triggered to send the appropriate command to the backend.

```
JavaScript
const playButton = document.getElementById('play-button');
playButton.addEventListener('click', function() {
    const selectedFile = document.getElementById('file-select').value;
    const brightness = document.getElementById('brightness-slider').value;

    fetch('/api/play', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({
            filename: selectedFile,
            brightness: parseInt(brightness)
        }),
    });
});

const stopButton = document.getElementById('stop-button');
stopButton.addEventListener('click', function() {
    fetch('/api/stop', { method: 'POST' });
});
```

54

This client-server architecture creates a clean separation of concerns. The JavaScript in the browser is responsible only for capturing user intent and sending it to the server. The Flask backend contains all the logic for interacting with the hardware and filesystem. This makes the system robust, secure, and easy to debug or extend in the future.

Part III: Adaptation and Synchronization for a WS2811 Lightbox

This final part addresses the secondary goal of the project: creating an alternate display using a WS2811 LED string arranged as a lightbox and synchronizing its animations with the primary HUB75 matrix. This involves selecting a more suitable microcontroller, addressing specific hardware challenges, and implementing a robust communication protocol to link the two systems.

Chapter 7: Engineering the Alternate Display

This chapter details the hardware selection and construction for the WS2811-based lightbox, highlighting the key differences from the HUB75 system.

7.1 Controller Showdown: Why the ESP32 is the Right Tool

While the Raspberry Pi is a full-fledged single-board computer, the lightbox requires a microcontroller—a simpler device dedicated to a single task. The ESP32 is the ideal choice for this role, far surpassing a basic Arduino Uno for several reasons:

- **Processing Power and Memory:** The ESP32 features a much faster dual-core processor and significantly more RAM than an Arduino Uno. This is critical for driving a large number of LEDs (even ~100) with complex animations without performance degradation.[57]
- **Integrated Connectivity:** The ESP32 has built-in Wi-Fi and Bluetooth, which are essential for receiving commands from the Raspberry Pi over the network. Adding Wi-Fi to an Arduino requires an additional "shield," increasing complexity and cost.[57]
- **Real-Time Operating System (FreeRTOS):** The ESP32 runs on FreeRTOS, which allows for true multitasking. This means the microcontroller can handle network communication on one core or task while simultaneously running a time-sensitive LED animation loop on another, preventing the two functions from interfering with each other—a significant advantage for creating smooth, responsive displays.[57]

7.2 Driving WS2811 LEDs: The 3.3V to 5V Logic Level Challenge

A critical electrical consideration when using an ESP32 with WS2811-type LEDs is the difference in logic voltage levels. The ESP32's GPIO pins operate at 3.3V, while WS2811 LEDs expect a 5V data signal. While a direct connection may sometimes work for short distances, it is unreliable and operates outside the specified tolerance of the LEDs, often resulting in flickering, incorrect colors, or a complete failure to display anything.[59]

The professional solution is to use a logic level shifter. A simple and effective choice is a 74HCT series chip, such as the 74HCT245. This chip is powered by 5V and correctly interprets the 3.3V signal from the ESP32 as a "high" input, outputting a clean, robust 5V signal for the LED data line. This ensures reliable data transmission and is a mandatory component for a stable build.[59]

7.3 Assembling the Lightbox: Serpentine Grid Layout

To create a grid-like display from a single LED string, a serpentine or "S" pattern is used. The strip is laid out in rows, with each

row running in the opposite direction of the one before it.

- **Layout:** For a grid of approximately 100 LEDs, a 10x10 arrangement is practical. The strip would start at corner (0,0), run 10 pixels to (9,0), fold back and run from (9,1) to (0,1), fold again and run from (0,2) to (9,2), and so on.[61]
- **Wiring:** The ESP32's data output pin connects (via the level shifter) to the "Data In" pad of the very first LED in the string. Power (5V) and Ground from a separate, appropriately sized power supply are connected to the power injection wires on the strip. It is crucial that the ground of the LED power supply is also connected to a ground pin on the ESP32 to provide a common ground reference for the data signal.[63]

Chapter 8: Programming the WS2811 Controller

The firmware for the ESP32 will be written in C++ using the Arduino IDE or a more advanced environment like PlatformIO. This firmware will be responsible for connecting to the network, receiving commands, and rendering animations on the LED grid.

8.1 Library Showdown: FastLED vs. Adafruit NeoPixel

Two primary libraries dominate the addressable LED space. For this project, FastLED is the recommended choice.

- **Adafruit NeoPixel:** This library is very beginner-friendly and easy to use for simple effects. However, its brightness control is "destructive" (it alters the color data in memory), and it lacks many advanced features.[65]
- **FastLED:** This is a more powerful and performance-oriented library. It offers non-destructive brightness control, advanced color correction and dithering for smoother gradients, a vast library of mathematical and noise functions for generating complex procedural animations, and support for a wider range of LED chipsets. For a project focused on "custom animations," FastLED provides a far more capable and professional toolkit.[65]

8.2 The XY Mapping Function: Translating Coordinates to a Serpentine Strip

Because the LEDs are physically arranged in a 2D grid but are wired as a single 1D strip, a software function is required to map 2D (x,y) coordinates to the correct 1D pixel index. This XY function is the key to drawing 2D shapes and images on the serpentine layout.

The logic must account for the alternating direction of the rows. For even-numbered rows, the index increases from left to right. For odd-numbered rows, it increases from right to left.

C++

```
// Example XY mapping function for a 10x10 serpentine grid
#define MATRIX_WIDTH 10
#define MATRIX_HEIGHT 10

uint16_t XY(uint8_t x, uint8_t y) {
  uint16_t i;

  if (y % 2 == 0) {
    // Even rows (0, 2, 4,...): left to right
```

```
  i = (y * MATRIX_WIDTH) + x;
 } else {
  // Odd rows (1, 3, 5,...): right to left
  i = (y * MATRIX_WIDTH) + (MATRIX_WIDTH - 1 - x);
 }

 return i;
}
```

62

This function allows the rest of the animation code to think in terms of a simple 2D grid (e.g., leds = CRGB::Blue;), abstracting away the complexity of the physical wiring.

8.3 Developing the ESP32 Animation Firmware

The base firmware sketch for the ESP32 will set up the necessary components and enter a loop to listen for commands and update the display.

The structure will include:

1. **Includes:** Necessary libraries for WiFi, the chosen communication protocol (MQTT, discussed next), and FastLED.
2. **Configuration:** WiFi credentials, network addresses, LED strip parameters (pin number, LED count, etc.).
3. **setup() function:** Initializes serial communication for debugging, connects to WiFi, initializes the FastLED strip, and connects to the communication server.
4. **loop() function:** Continuously checks for new network messages and calls the appropriate animation-rendering functions based on the commands received.

Chapter 9: Unifying the System with MQTT

To achieve seamless synchronization, a robust communication protocol is needed. Directly connecting the two devices via IP sockets is brittle. A far superior approach for this type of IoT application is to use MQTT (Message Queuing Telemetry Transport).

9.1 Introducing MQTT: A Protocol for IoT

MQTT is a lightweight publish-subscribe messaging protocol. It works by having a central "broker" (server) that manages the distribution of messages. Clients can "publish" messages to named channels called "topics," and other clients can "subscribe" to those topics to receive the messages.

This model is ideal for this project because it decouples the Raspberry Pi from the ESP32:

- The Raspberry Pi (publisher) only needs to know the address of the broker. It sends a command to a topic like displays/lightbox/play without knowing or caring who is listening.[68]
- The ESP32 (subscriber) only needs to know the broker's address and the topic it's interested in. It receives commands without needing to know who sent them.
  This makes the system incredibly scalable. One could add five more lightboxes, and they would all synchronize instantly by

subscribing to the same topic, with no changes required on the Raspberry Pi or the web application.

9.2 Setting Up an MQTT Broker on the Raspberry Pi

The Raspberry Pi is the perfect device to host the MQTT broker. Mosquitto is a popular, lightweight, and easy-to-install open-source broker.

Bash

```
# Install the Mosquitto broker and command-line clients
sudo apt install mosquitto mosquitto-clients
```

69

By default, Mosquitto starts automatically and listens for connections on port 1883, allowing anonymous connections from clients on the local network, which is sufficient for this project.69

9.3 Modifying the Flask Application to Publish MQTT Messages

The Flask backend will be updated to act as an MQTT publisher. This is done using the paho-mqtt library for Python.

1. **Install the library:** pip install paho-mqtt
2. **Modify API Endpoints:** The existing API endpoints in app.py will be modified to publish an MQTT message in addition to controlling the local display.

Python

```
# In app.py, add imports and client setup
import paho.mqtt.client as mqtt

MQTT_BROKER = 'localhost'
MQTT_PORT = 1883
mqtt_client = mqtt.Client("WebAppController")
mqtt_client.connect(MQTT_BROKER, MQTT_PORT)

# Modify the /api/play route
@app.route('/api/play', methods=)
def play_animation():
    #... (existing code to control local display via subprocess)...

    # Now, publish the same command to MQTT
    topic = "displays/lightbox/control"
    payload = request.data # The JSON payload from the frontend
    mqtt_client.publish(topic, payload)

    return jsonify({'status': 'playing', 'file': filename})
```

70

9.4 Creating an MQTT Client on the ESP32

Finally, the ESP32 firmware is updated to be an MQTT client that subscribes to the control topic. The PubSubClient library is a standard choice for this in the Arduino environment.

1. **Install the library:** Install PubSubClient via the Arduino IDE's Library Manager.
2. **Implement MQTT Logic:** The sketch will be updated to connect to the Mosquitto broker running on the Raspberry Pi's IP address. A callback function will be defined to handle incoming messages.

C++

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <FastLED.h>
//... other includes and definitions...

const char* mqtt_server = "RASPBERRY_PI_IP_ADDRESS";
WiFiClient espClient;
PubSubClient client(espClient);

void callback(char* topic, byte* payload, unsigned int length) {
  // This function is called when a message arrives.
  // The payload will contain the JSON string, e.g.,
  // {"filename": "animation.gif", "brightness": 80}

  // Parse the JSON payload here (using ArduinoJson library, for example)
  // and extract the animation name and parameters.

  // Based on the parsed data, call the appropriate
  // animation function to run on the WS2811 grid.
}

void setup() {
  //... (WiFi connection logic)...
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void reconnect() {
  while (!client.connected()) {
    if (client.connect("ESP32_Lightbox_1")) {
      // Subscribe to the control topic
      client.subscribe("displays/lightbox/control");
    } else {
      delay(5000); // Wait 5 seconds before retrying
    }
  }
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop(); // This is crucial, it processes incoming messages

  //... (run the current animation)...
}
```

With this final piece in place, the system is complete. A single user action in the web GUI is now translated into a command that is executed locally on the Raspberry Pi for the HUB75 matrix and simultaneously broadcast via MQTT to the ESP32, resulting in two physically and architecturally distinct display systems playing custom animations in perfect synchrony. This event-driven architecture is not only robust but also provides a clear path for future expansion to control an entire ecosystem of synchronized displays.

Works cited

1. Adafruit RGB Matrix + Real Time Clock HAT for Raspberry Pi, accessed August 11, 2025, https://cdn-learn.adafruit.com/downloads/pdf/adafruit-rgb-matrix-plus-real-time-clock-hat-for-raspberry-pi.pdf
2. Adafruit RGB Matrix Bonnet for Raspberry Pi : ID 3211, accessed August 11, 2025, https://www.adafruit.com/product/3211
3. Adafruit RGB Matrix HAT + RTC for Raspberry Pi - Mini Kit - Vilros.com, accessed August 11, 2025, https://vilros.com/products/adafruit-rgb-matrix-hat-rtc-for-raspberry-pi-mini-kit
4. Adafruit RGB Matrix HAT + RTC for Raspberry Pi - Mini Kit, accessed August 11, 2025, https://www.adafruit.com/product/2345
5. 64x64 LED Matrix w/ adafruit bonnet - black bars, accessed August 11, 2025, https://forums.adafruit.com/viewtopic.php?t=156379
6. Pinouts | Adafruit Triple LED Matrix Bonnet for Raspberry Pi with HUB75, accessed August 11, 2025, https://learn.adafruit.com/adafruit-triple-led-matrix-bonnet-for-raspberry-pi-with-hub75/pinouts
7. 64x64 RGB LED Matrix - 3mm Pitch [192mm x 192mm] : ID 4732 ..., accessed August 11, 2025, https://www.adafruit.com/product/4732
8. 64x64 RGB LED Matrix - 2.5mm Pitch [1/32 Scan] : ID 3649 - Adafruit, accessed August 11, 2025, https://www.adafruit.com/product/3649
9. Driving Matrices | Adafruit RGB Matrix + Real Time Clock HAT for Raspberry Pi, accessed August 11, 2025, https://learn.adafruit.com/adafruit-rgb-matrix-plus-real-time-clock-hat-for-raspberry-pi/driving-matrices
10. Adafruit RGB Matrix Bonnet for Raspberry Pi, accessed August 11, 2025, https://learn.adafruit.com/adafruit-rgb-matrix-bonnet-for-raspberry-pi?view=all
11. Raspberry Pi - Home Assistant, accessed August 11, 2025, https://www.home-assistant.io/installation/raspberrypi/
12. How to Use an LED Matrix Panel with Your Raspberry Pi - Howchoo, accessed August 11, 2025, https://howchoo.com/pi/raspberry-pi-led-matrix-panel
13. Adafruit RGB Matrix + Real Time Clock HAT for Raspberry Pi, accessed August 11, 2025, https://www.sigmaelectronica.net/manuals/ADAFR-2345.pdf
14. RGB LED Matrix Cube with 25,000 LEDs - Adafruit, accessed August 11, 2025, https://cdn-learn.adafruit.com/downloads/pdf/rgb-led-matrix-cube-for-pi.pdf
15. hzeller/rpi-rgb-led-matrix: Controlling up to three chains of 64x64, 32x32, 16x32 or similar RGB LED displays using Raspberry Pi GPIO - GitHub, accessed August 11, 2025, https://github.com/hzeller/rpi-rgb-led-matrix
16. Adafruit RGB Matrix Bonnet for Raspberry Pi, accessed August 11, 2025, https://cdn-learn.adafruit.com/downloads/pdf/adafruit-rgb-matrix-bonnet-for-raspberry-pi.pdf
17. Connections | RGB LED Matrix Basics - Adafruit Learning System, accessed August 11, 2025, https://learn.adafruit.com/32x16-32x32-rgb-led-matrix/new-wiring
18. Raspberry Pi Setup Guide | Prepare Devices - Viam Documentation, accessed August 11, 2025, https://docs.viam.com/operate/reference/prepare/rpi-setup/
19. Set up your SD card | Physical Computing, accessed August 11, 2025, https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up/4
20. Installing the operating system | The Raspberry Pi Guide, accessed August 11, 2025, https://raspberrypi-guide.github.io/getting-started/install-operating-system
21. Raspberry Pi RGB LED Matrix Webapp | Adafruit, accessed August 11, 2025, https://cdn-learn.adafruit.com/downloads/pdf/raspberry-pi-rgb-led-matrix-webapp.pdf
22. Raspberry Pi LED Matrix Display - Adafruit, accessed August 11, 2025, https://cdn-learn.adafruit.com/downloads/pdf/raspberry-pi-led-matrix-display.pdf
23. RGB Matrix Panels With Raspberry Pi 5 - Adafruit Learning System, accessed August 11, 2025, https://learn.adafruit.com/rgb-matrix-panels-with-raspberry-pi-5/raspberry-pi-5-setup
24. Rpi rgb led matrix - Raspberry Valley, accessed August 11, 2025, https://raspberry-valley.azurewebsites.net/rpi-rgb-led-matrix/
25. rpi-rgb-led-matrix/utils/README.md at master - GitHub, accessed August 11, 2025, https://github.com/hzeller/rpi-rgb-led-matrix/blob/master/utils/README.md
26. RasPi 002: LED Matrix Display - Weissman Foundry - Babson College, accessed August 11, 2025, https://www.foundry.babson.edu/raspi-002
27. rpi-rgb-led-matrix/bindings/python/samples/image-draw.py at master - GitHub, accessed August 11, 2025, https://github.com/hzeller/rpi-rgb-led-matrix/blob/master/bindings/python/samples/image-draw.py
28. rpi-rgb-led-matrix/bindings/python/samples/image-viewer.py at master - GitHub, accessed August 11, 2025, https://github.com/hzeller/rpi-rgb-led-matrix/blob/master/bindings/python/samples/image-viewer.py
29. How to display video via 64*64 Led matrix? · Issue #322 · hzeller/rpi-rgb-led-matrix - GitHub, accessed August 11, 2025, https://github.com/hzeller/rpi-rgb-led-matrix/issues/322

30. MP4 Video | RGB Matrix Panels With Raspberry Pi 5 - Adafruit Learning System, accessed August 11, 2025, https://learn.adafruit.com/rgb-matrix-panels-with-raspberry-pi-5/mp4-video
31. Raspberry Pi LED Matrix Display - Adafruit Learning System, accessed August 11, 2025, https://learn.adafruit.com/raspberry-pi-led-matrix-display/overview
32. rpi-rgb-led-matrix/bindings/python/samples/runtext.py at master - GitHub, accessed August 11, 2025, https://github.com/hzeller/rpi-rgb-led-matrix/blob/master/bindings/python/samples/runtext.py
33. Understanding Canvas - rpi-rgb-led-matrix, accessed August 11, 2025, https://rpi-rgb-led-matrix.discourse.group/t/understanding-canvas/419
34. Animated GIF | RGB Matrix Panels With Raspberry Pi 5 - Adafruit Learning System, accessed August 11, 2025, https://learn.adafruit.com/rgb-matrix-panels-with-raspberry-pi-5/animated-gif
35. Displaying animated gifs #11 - hzeller/rpi-rgb-led-matrix - GitHub, accessed August 11, 2025, https://github.com/hzeller/rpi-rgb-led-matrix/issues/11
36. Node.js vs. Flask: Which is Better in 2025? - Flatirons Development, accessed August 11, 2025, https://flatirons.com/blog/nodejs-vs-flask-which-is-better-2024/
37. Which Is Faster Flask Or NodeJS - Unbiased Coder, accessed August 11, 2025, https://unbiased-coder.com/which-is-faster-flask-or-nodejs/
38. Use Python flask or NodeJS for controlling stepper motors via Web?, accessed August 11, 2025, https://raspberrypi.stackexchange.com/questions/95449/use-python-flask-or-nodejs-for-controlling-stepper-motors-via-web
39. Python Flask vs Node.js Express. When building web applications... | by Jason Roell | Medium, accessed August 11, 2025, https://medium.com/@roelljr/python-flask-vs-node-js-express-4662b6f97b28
40. Raspberry Pi Web Server using Flask to Control GPIOs - Random Nerd Tutorials, accessed August 11, 2025, https://randomnerdtutorials.com/raspberry-pi-web-server-using-flask-to-control-gpios/
41. Running a Python Flask Web App on a Raspberry Pi - Pi My Life Up, accessed August 11, 2025, https://pimylifeup.com/raspberry-pi-flask-web-app/
42. Quickstart — Flask Documentation (3.1.x), accessed August 11, 2025, https://flask.palletsprojects.com/en/stable/quickstart/
43. Uploading Files — Flask Documentation (3.1.x), accessed August 11, 2025, https://flask.palletsprojects.com/en/stable/patterns/fileuploads/
44. Using Flask to Send Data to a Raspberry Pi - SparkFun Learn, accessed August 11, 2025, https://learn.sparkfun.com/tutorials/using-flask-to-send-data-to-a-raspberry-pi/all
45. Raspberry Pi - Create a Flask Server - The Robotics Back-End, accessed August 11, 2025, https://roboticsbackend.com/raspberry-pi-create-a-flask-server/
46. Create the app | Build a Python Web Server with Flask - Code Club Projects, accessed August 11, 2025, https://projects.raspberrypi.org/en/projects/python-web-server-with-flask/1
47. How to Upload File in Python-Flask - GeeksforGeeks, accessed August 11, 2025, https://www.geeksforgeeks.org/python/how-to-upload-file-in-python-flask/
48. Flask File Uploading - Tutorialspoint, accessed August 11, 2025, https://www.tutorialspoint.com/flask/flask_file_uploading.htm
49. Upload a File with Python Flask, accessed August 11, 2025, https://pythonbasics.org/flask-upload-file/
50. Flask-Uploads — Flask-Uploads 0.1.1 documentation, accessed August 11, 2025, https://flask-uploads.readthedocs.io/
51. How to Change Brightness using HTML Range Slider ? - GeeksforGeeks, accessed August 11, 2025, https://www.geeksforgeeks.org/html/how-to-change-brightness-using-html-range-slider/
52. range slider brightness - darkcode - CodePen, accessed August 11, 2025, https://codepen.io/ruuuff/pen/qBOryWB
53. Tailwind CSS Dropdown - Flowbite, accessed August 11, 2025, https://flowbite.com/docs/components/dropdowns/
54. JavaScript, fetch, and JSON — Flask Documentation (3.1.x), accessed August 11, 2025, https://flask.palletsprojects.com/en/stable/patterns/javascript/
55. How do you call a function in flask through html elements? - Reddit, accessed August 11, 2025, https://www.reddit.com/r/flask/comments/14yn81p/how_do_you_call_a_function_in_flask_through_html/
56. How to do a post to flask from a button - Stack Overflow, accessed August 11, 2025, https://stackoverflow.com/questions/67052349/how-to-do-a-post-to-flask-from-a-button
57. ESP32 vs Arduino - Reddit, accessed August 11, 2025, https://www.reddit.com/r/esp32/comments/1bhrf2d/esp32_vs_arduino/
58. Which board would be the best for these LEDs? - General Guidance - Arduino Forum, accessed August 11, 2025, https://forum.arduino.cc/t/which-board-would-be-the-best-for-these-leds/1349577
59. ESP32 and WS2812B - LEDs and Multiplexing - Arduino Forum, accessed August 11, 2025, https://forum.arduino.cc/t/esp32-and-ws2812b/569469
60. bertmelis/esp32WS2811: Arduino library for ESP32 to drive WS2811 LEDs using the RMT peripheral - GitHub, accessed August 11, 2025, https://github.com/bertmelis/esp32WS2811
61. Serpentine wired LED panel - static gradient of colour : r/WLED - Reddit, accessed August 11, 2025, https://www.reddit.com/r/WLED/comments/lri6lg/serpentine_wired_led_panel_static_gradient_of/
62. Fastled XY matrix using the serpentine style wiring - LEDs and Multiplexing - Arduino Forum, accessed August 11, 2025, https://forum.arduino.cc/t/fastled-xy-matrix-using-the-serpentine-style-wiring/422068
63. What tips do you have for wiring 20m of WS2811 12V RGB Strips ? : r/WLED - Reddit, accessed August 11, 2025, https://www.reddit.com/r/WLED/comments/1j0b0rv/what_tips_do_you_have_for_wiring_20m_of_ws2811/
64. WS2811 Strips design help - Teensy Forum, accessed August 11, 2025, https://forum.pjrc.com/index.php?threads/ws2811-strips-design-help.24041/

65. Adafruit vs FastLED Library for Addressable LED Strip - Suntech Lite, accessed August 11, 2025, https://suntechlite.com/adafruit-vs-fastled-library-for-addressable-led-strip/
66. Adafruit_NeoPixel vs. FastLED - LEDs and Multiplexing - Arduino Forum, accessed August 11, 2025, https://forum.arduino.cc/t/adafruit_neopixel-vs-fastled/343660
67. dmadison/FastLED_NeoPixel: An Arduino library for using Adafruit NeoPixel library animations with FastLED - GitHub, accessed August 11, 2025, https://github.com/dmadison/FastLED_NeoPixel
68. ESP32 MQTT Publish Subscribe with Arduino IDE - Random Nerd Tutorials, accessed August 11, 2025, https://randomnerdtutorials.com/esp32-mqtt-publish-subscribe-arduino-ide/
69. How to Connect an ESP32 WiFi Microcontroller to a Raspberry Pi Using IoT MQTT, accessed August 11, 2025, https://predictabledesigns.com/how-to-connect-esp32-microcontroller-to-raspberry-pi-using-iot-mqtt/
70. ESP32 and MQTT - Raspberry Pi Forums, accessed August 11, 2025, https://forums.raspberrypi.com/viewtopic.php?t=265048
71. Communicating ESP32 with Arduino IDE via MQTT to Raspberry Pi - Stack Overflow, accessed August 11, 2025, https://stackoverflow.com/questions/77058670/communicating-esp32-with-arduino-ide-via-mqtt-to-raspberry-pi