Ashish Anil Bhogate & Gargi Sadashiv Gaitonde
Tejaswi Linge Gowda
AME 598: Programming in IOT
05 December 2023

SecureEye: Security Surveillance

The evolution of Internet of Things (IoT) technology is redefining the traditional paradigms of security measures. It has emerged as a transformative tool, enabling a proactive approach to safeguarding environments, assets, and information. This project presents the development of an IoT security system leveraging a Raspberry Pi equipped with a motion detection module and a user-friendly web interface is implemented allowing users to conveniently monitor and review captured events. The integration of hardware and software components enables real-time monitoring and facilitates remote surveillance, enhancing security measures through efficient motion-triggered image capture and remote accessibility.

INTRODUCTION

In a world characterized by bustling schedules and diverse living environments, the need for personalized security solutions has become increasingly pertinent. As students navigating our busy routines, the challenge of safeguarding deliveries, mitigating package theft, and ensuring the integrity of online order deliveries has resonated deeply with our experiences. Instances where packages are reported as delivered despite being unaccounted for in our non-gated community prompted the realization of an unmet need for a reliable surveillance system.

The fusion of Internet of Things (IoT) technology with surveillance systems presents an innovative approach to address this concern. The majority of commercially available smart camera control systems operate as opaque entities within our homes, lacking transparency regarding their internal mechanisms and processes. Despite the presence of reputable vendors, there exists a desire to delve deeper into the creation of a personalized smart camera control system. Moreover, pre-packaged smart camera solutions often raise concerns about data privacy and security. Thus, we aim for crafting a fully-fledged, customizable product and perhaps creating a versatile, market-ready IoT security solution. This exploration is driven by the intent to comprehend the foundational aspects of IoT while cultivating a deeper understanding of network protocols and data transmission in IoT ecosystems.

METHODOLOGY

The methodology section outlines the approach and tools used to accomplish the system's key functions. The study implements two distinct methodologies to achieve this objective. Initially, the system is developed and tested on a laptop system utilizing its camera module. Subsequently, a standalone project is executed using Raspberry Pi along with its camera, enabling deployment outside an apartment setting.

The primary functionality of the system involves capturing images triggered by motion detection. A specialized motion detection module is employed for this purpose. Once motion is detected, the system captures an image and securely stores it on a remote server. Access to these stored images, alongside their respective timestamps, is facilitated through a dedicated web page connected to the server. This permits users to review captured events at their convenience.

Furthermore, to ascertain the efficiency and performance of both implementations, a comparative analysis was conducted. The comparative analysis focused on assessing the time taken by each system to detect motion, capture images, and store them securely on the remote server.

(I) *Motion Detection:*

The detection.py code is designed for motion detection with an added functionality to upload captured images to an AWS EC2 server. The script begins by setting up parameters such as the server URL, EC2 details (host, username, private key path, and remote folder), and initializes variables to keep track of motion detection status and timestamps. The motion detection process is achieved through the OpenCV library, using the createBackgroundSubtractorMOG2 method to establish a background model and countNonZero to determine the area of motion in the frame. When motion is detected (defined by a threshold value), the script captures the current frame, saves it as an image file with a timestamped filename, and proceeds to upload the image to the specified AWS EC2 server.

The image upload process to the EC2 server is accomplished through the paramiko library, which facilitates SSH connections and SFTP file transfers. The script establishes an SSH connection to the EC2 server using the provided credentials, creates a directory on the server if it does not exist, and uploads the captured image to the designated folder. The script then measures the time taken for the image upload process, providing insights into the efficiency of data transfer. Furthermore, the script measures the response time by sending an HTTP POST request to the specified server URL using the requests library. It includes the captured image file as part of the POST request's payload. The server-side code on the AWS EC2 server, as represented by the provided server.js script, handles this incoming request by saving the image to a folder and responding with relevant image details.

In technical terms, the image transfer utilizes SFTP (SSH File Transfer Protocol) for secure file upload to the EC2 server. The script employs exception handling to manage potential errors during the image upload process, ensuring robustness in handling network or server-related issues. The combined motion detection and image upload functionalities make the detection.py script a versatile component for real-time monitoring and remote storage of captured images, demonstrating the integration of computer vision, networking, and cloud technologies in a cohesive system.

(II) ***Server:***

The server.js code serves as the backend component of a motion detection and image upload system. Leveraging the Express framework in Node.js, this script establishes a web server to facilitate communication with a front-end or other client applications. The server serves static files from the 'public' folder, although this aspect appears unused in the provided code. It also exposes an endpoint at '/images' where clients can request details about captured images. Upon receiving a request, the server asynchronously reads the contents of the 'captured_images' directory, retrieves file names and their corresponding upload times, and responds with a JSON object containing this information. This functionality seamlessly integrates with a motion detection system, as the client can dynamically retrieve information about the images captured during motion events. The server's ability to serve static images enhances the system's scalability and enables efficient retrieval of image details, contributing to a comprehensive and responsive user experience. Additionally, the server.js script showcases effective file system operations and HTTP handling within the Express framework, aligning with best practices for building robust and scalable web applications.

(III) ***Hardware Implementation:***

1. Running Code on Local Device (Laptop) with Webcam:

For the local device implementation, the project leverages a laptop equipped with a webcam. The OpenCV library interfaces with the laptop's webcam to perform real-time motion detection. The motion detection algorithm processes video frames captured by the webcam, utilizing background subtraction techniques to identify areas of interest. Upon detecting motion exceeding a predefined threshold, the system captures the current frame, timestamps the image, and initiates a secure upload to an AWS EC2 server. This implementation is ideal for scenarios where a portable and easily accessible device, such as a laptop, serves as the monitoring and processing hub. The webcam provides a cost-effective and widely available camera solution, making it suitable for various indoor surveillance and monitoring applications.

2. Running Code on Raspberry Pi with Pi Camera Module:

In the Raspberry Pi implementation, the project takes advantage of the Raspberry Pi single-board computer and its dedicated Pi Camera Module. The Pi Camera Module offers a compact and efficient camera solution designed specifically for Raspberry Pi devices. The motion detection script, running on the Raspberry Pi, interfaces with the Pi Camera Module to capture video frames. The motion detection algorithm, powered by OpenCV, processes these frames, identifies motion events, and triggers the capture and upload of time stamped images to the AWS EC2 server. This hardware setup is well-suited for scenarios where a compact, energy-efficient, and dedicated device like the Raspberry Pi is preferred. The Pi Camera Module provides high-quality imaging capabilities, making it suitable for various outdoor and IoT applications, where space and power efficiency are critical considerations.

Both implementations showcase the flexibility of the motion detection system, adapting to different hardware configurations while maintaining the core functionality of real-time motion detection and secure image upload to a remote server. The choice between a laptop and a Raspberry Pi depends on specific project requirements, such as portability, power consumption, and the need for a dedicated IoT device.

CHALLENGES ENCOUNTERED

1. Hardware Compatibility and Configuration:
    - Local Device (Laptop): Meticulous adjustment of device drivers and OpenCV configurations for seamless interfacing with different webcam models.
    - Raspberry Pi: Overcoming compatibility issues, optimizing the interaction with the Pi Camera Module, and fine-tuning camera module settings.
2. Achieving optimal sensitivity through extensive experimentation to find the right threshold.
3. Establishing secure communication with AWS EC2 server, including key management for SSH connections and SFTP implementation.
4. Addressing potential network challenges, such as latency and packet loss, through server and network settings configuration.
5. Implementing robust exception handling mechanisms in Python for motion detection and image upload scripts.
6. Graceful handling of issues like network interruptions, server unavailability, and unexpected errors to prevent system failures and data loss.

7. Optimization of file handling in the server-side script (server.js) for efficient storage and retrieval of uploaded images.
8. Synchronization and proper organization of images on the server to prevent conflicts and facilitate streamlined access.
9. Calibration procedures, including fine-tuning motion detection parameters, adjusting camera settings, and ensuring consistent performance across different devices and environments.

CLASSROOM IMPACT

My journey in the Internet of Things (IoT) class, led by Professor Tejaswi Gowda, has been immensely enlightening, particularly considering my non-CS background. Key learnings include:

- Understanding the process of setting up an AWS EC2 instance provided valuable insights into cloud computing. Learning to create and manage a server environment on AWS expanded my knowledge of infrastructure as a service (IaaS) and its applications in IoT.
- The ability to create a personal server and configure it for specific tasks was a pivotal skill gained. The hands-on experience in server setup enhanced my understanding of server architecture and its role in IoT applications.
- Learning how to send real-time data to a specific IP address was a crucial aspect of the course. This skill is fundamental for IoT applications where timely and accurate data transmission is imperative.

RESEARCH EVOLUTION

Throughout the project journey, involved rigorously testing the functionality of the developed APIs to ensure seamless data transfer between the local device and the AWS EC2 server. The hands-on experience with API creation and testing in Postman not only contributed to the successful implementation of the motion detection system but also broadened my understanding of API integration in IoT projects. This newfound knowledge serves as a cornerstone for future research and development endeavors in the realm of IoT.

Overall, the classroom impact and research evolution in the IoT class have equipped me with practical skills and insights that transcend theoretical knowledge. The ability to set up servers, transmit real-time data, and implement APIs opens avenues for continued exploration and innovation in the dynamic field of IoT.

IMPLEMENTATION DISCREPANCIES

Intended Plan:

The initial project plan aimed to conduct motion detection on two distinct systems: a laptop with a webcam and a Raspberry Pi with the Pi Camera Module. The primary objective was to compare the efficiency of the motion detection algorithm on these platforms and assess the performance of image uploads to an AWS EC2 server. As per the professor's suggestion, the plan included recording upload times for each motion-detected image which measures the duration it takes to transfer a captured image from the local device (laptop or Raspberry Pi) to the AWS EC2 server.

Actual Execution:

The project successfully executed the intended plan, addressing a range of software and hardware challenges encountered during implementation. On the laptop with a webcam, the motion detection script interfaced with OpenCV, capturing video frames and demonstrating robust performance. On the Raspberry Pi with the Pi Camera Module, the script seamlessly integrated with the camera module, adapting to varying environmental conditions. Both systems successfully recorded upload times, providing crucial insights into the efficiency of image uploads. During the actual execution, upload times were systematically measured for each captured image during the transfer to the AWS EC2 server.

Moreover, the actual execution involved additional metrics beyond the initial plan, including response time and timestamp recording. The response time represented the duration between initiating the image upload process and receiving confirmation from the server using HTTP POST request. Timestamps were systematically added to each image during the upload process, creating a chronological record of image transfers. This comprehensive approach allowed for a thorough comparative analysis of the systems, providing insights into upload efficiency and temporal characteristics of the motion detection and image transfer processes. Detailed documentation was maintained, encompassing configurations, code adjustments, and the outcomes of the comparative analysis, contributing to a transparent and well-documented account of the project's execution.

CONCLUSION

Project successfully implemented a motion detection system on both a laptop with a webcam and a Raspberry Pi with the Pi Camera Module. The system demonstrated adaptability to diverse hardware configurations, effectively addressing challenges and showcasing robust performance. By recording upload times, timestamps, and response times during image transfers to an AWS EC2 server, the project provided valuable insights into

system efficiency. The motion detection system's resilience to environmental variations and its seamless integration with cloud-based storage underscore its potential for real-world applications in monitoring and surveillance. The lessons learned in overcoming challenges and optimizing performance contribute to a solid foundation for future projects at the intersection of computer vision, IoT, and cloud computing.

FUTURE SCOPE:

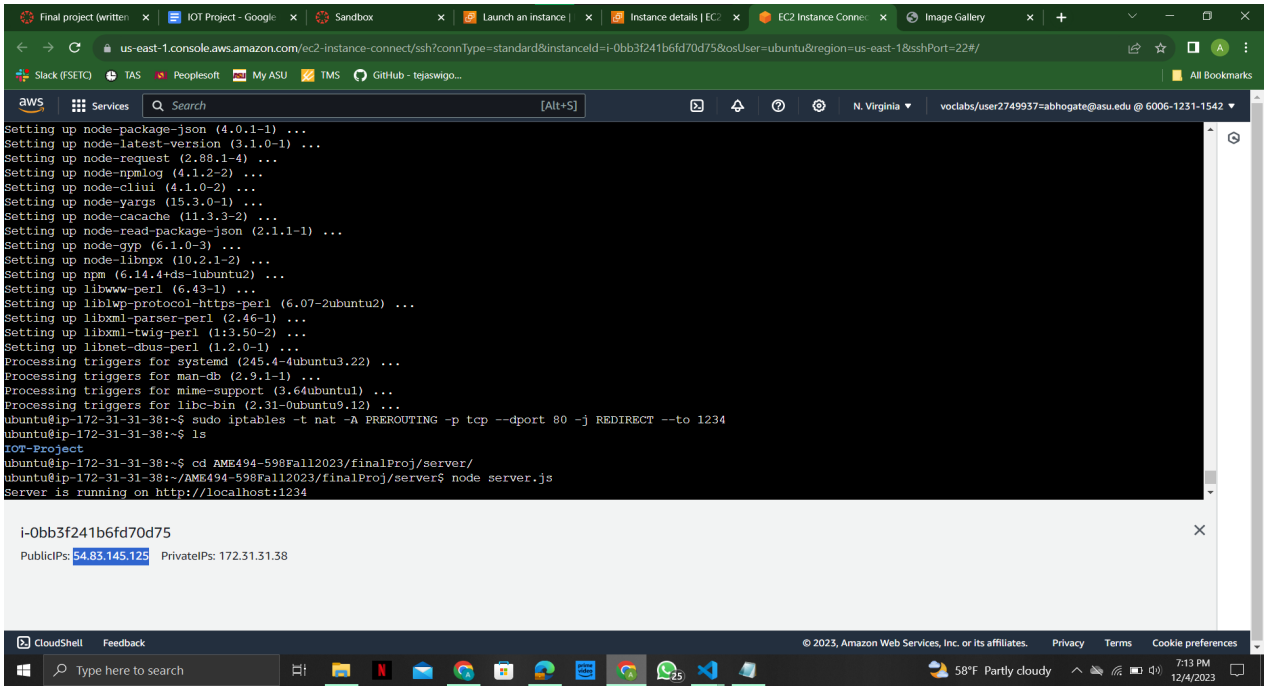
Looking ahead, the motion detection system presents several avenues for improvement and expansion. One critical area of focus involves implementing robust security measures to prevent unauthorized access. The introduction of authentication mechanisms, such as user credentials or token-based access, would fortify the system's defenses and ensure that only authorized users can access and manage the stored images on the AWS EC2 server. Additionally, future iterations could benefit from upgrading to more advanced camera modules, enhancing visual clarity through improved resolution and optical features. This upgrade would enable the system to capture higher-quality images, facilitating more detailed analysis and visualization of motion events. Moreover, the system's adaptability to specific applications can be further refined by introducing dynamic threshold adjustments. Configurable threshold limits would allow users to tailor the system's sensitivity, generating more or fewer images based on the nature of motion activities. These collective enhancements promise to elevate the system's security, visual capabilities, and applicability across a diverse range of scenarios, reinforcing its potential for real-world applications.

APPENDICES

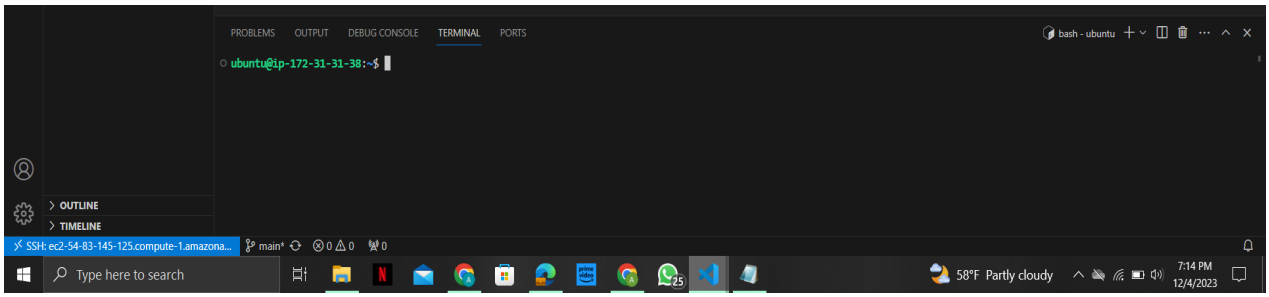
1. **Motion Detected images**

Image 1: 20231204190853.jpg Image 2: 20231204190857.jpg Image 3:20231204190853.jpg

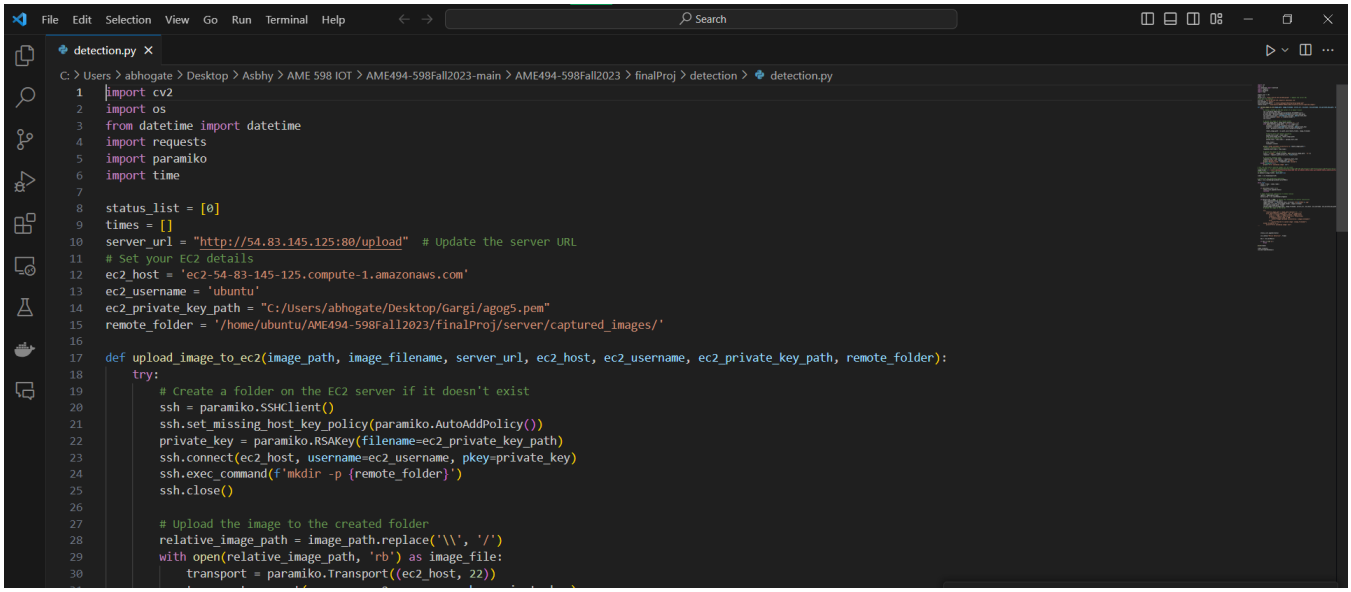
2. Starting AWS EC2 instance



3. Connecting VS Code with EC2 instance using SSH



4. Running Motion detection code, uploading images to the server and displaying server URL.

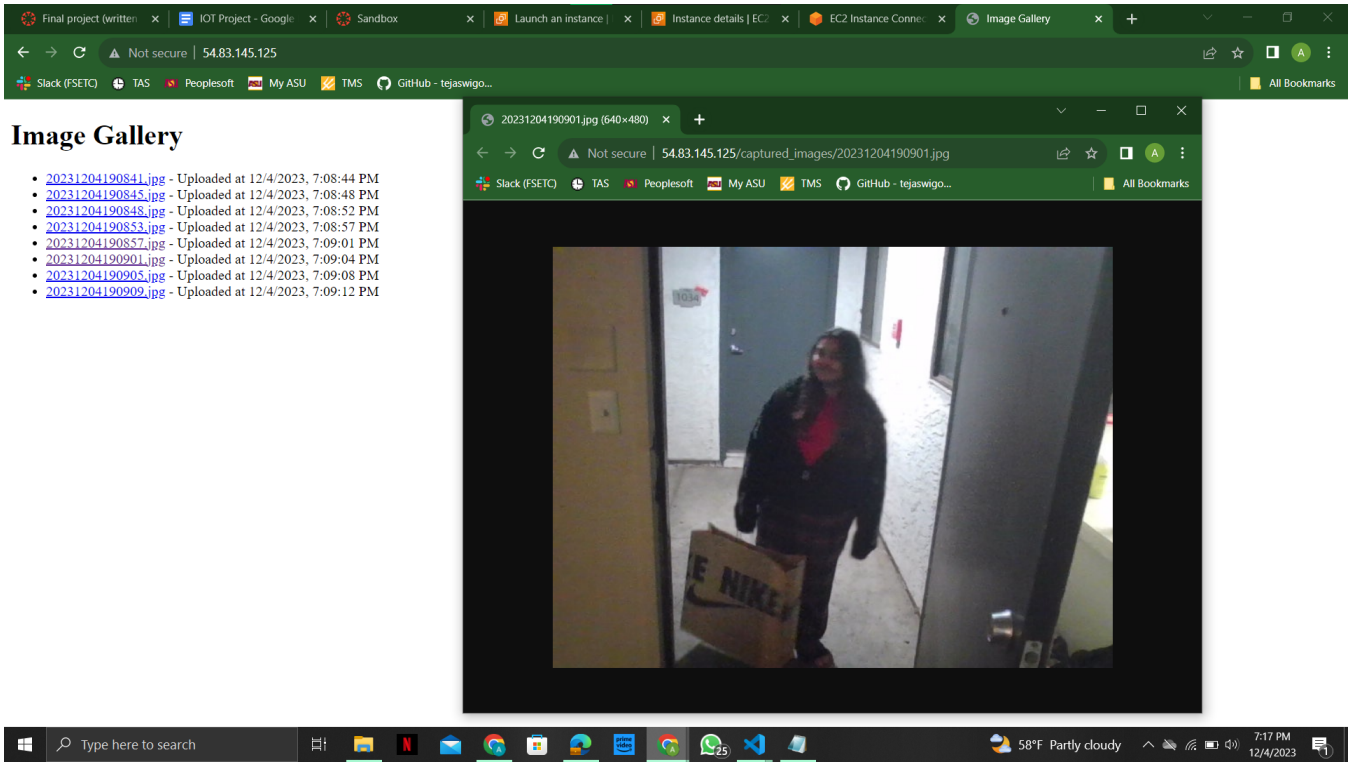


```python
import cv2
import os
from datetime import datetime
import requests
import paramiko
import time

status_list = [0]
times = []
server_url = "http://54.83.145.125:80/upload"  # Update the server URL
# Set your EC2 details
ec2_host = 'ec2-54-83-145-125.compute-1.amazonaws.com'
ec2_username = 'ubuntu'
ec2_private_key_path = "C:/Users/abhogate/Desktop/Gargi/agog5.pem"
remote_folder = '/home/ubuntu/AME494-598Fall2023/finalProj/server/captured_images/'

def upload_image_to_ec2(image_path, image_filename, server_url, ec2_host, ec2_username, ec2_private_key_path, remote_folder):
    try:
        # Create a folder on the EC2 server if it doesn't exist
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        private_key = paramiko.RSAKey(filename=ec2_private_key_path)
        ssh.connect(ec2_host, username=ec2_username, pkey=private_key)
        ssh.exec_command(f'mkdir -p {remote_folder}')
        ssh.close()

        # Upload the image to the created folder
        relative_image_path = image_path.replace('\\', '/')
        with open(relative_image_path, 'rb') as image_file:
            transport = paramiko.Transport((ec2_host, 22))
```

5. Recording upload time and response time

```
PS C:\Users\abhogate\Desktop\Asbhy\AME 598 IOT\AME494-598Fall2023-main\AME494-598Fall2023\finalProj> & C:/Users/abhogate/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/abhogate/Desktop/Asbhy/AME 598 IOT/AME494-598Fall2023-main/AME494-598Fall2023/finalProj/detection/detection.py"
Image uploaded successfully to /home/ubuntu/AME494-598Fall2023/finalProj/server/captured_images/20231204190841.jpg
Upload time: 0.5416724681854248 seconds
Response time: 0.35111474990844727 seconds
Image uploaded successfully to /home/ubuntu/AME494-598Fall2023/finalProj/server/captured_images/20231204190845.jpg
Upload time: 0.41295886039733887 seconds
Response time: 0.34766101837158203 seconds
Image uploaded successfully to /home/ubuntu/AME494-598Fall2023/finalProj/server/captured_images/20231204190848.jpg
Upload time: 0.4229426383972168 seconds
Response time: 0.46049928665161133 seconds
Image uploaded successfully to /home/ubuntu/AME494-598Fall2023/finalProj/server/captured_images/20231204190853.jpg
Upload time: 0.4564998149871826 seconds
Response time: 0.5901570320129395 seconds
Image uploaded successfully to /home/ubuntu/AME494-598Fall2023/finalProj/server/captured_images/20231204190857.jpg
Upload time: 0.43903231620788574 seconds
```

6. Displaying captured images on server URL



References

[1] Gowda, Tejaswi. “Tejaswigowda/AME498-598FALL2023.”, github.com/tejaswigowda/ame498-598Fall2023.

[2] Cipriani, Luca. “Building an IOT Security Camera with Raspberry Pi and Render.” *Medium*, Medium, 20 July 2022, astrolinux.medium.com/building-an-iot-security-camera-with-raspberry-pi-and-render-13c5056ed0e8.