Smart Home with IOT- “Digital Photo Frame”

Master of Engineering in
Information Technology
Mobile Computing
Pretom Kumar Saha
Matrikal No: 1276545
saha@stud.fra-uas.de

Supervised By:

Armin Lehmann

1 CONTENTS

Abstract 4

2 Introduction 5

3 Important tools 7

3.1 Internet of Things (IoT) 7

3.2 Constrained Application Protocol (CoAP) 7

3.3 Motion Sensor 8

3.4 Face Detection Sensor 8

4 supporting programming language 9

5 principles of project 10

5.1.1 Human exist in the room (Motion Sensor) 11

5.1.2 Identify friends and family persons (Face Detection Sensor) 11

5.2 Modeling of Use Case 11

6 Project Architecture 13

6.1 Dependencies 13

6.2 Classes 14

6.2.1 DigitalPhotoFrameControlar 14

6.2.2 MotionDetector 15

6.2.3 FaceDetectorSensor 16

6.2.4 DigitalphotoframeApplication 17

6.2.5 Getter And Setter 18

6.3 Output files and Wire shark capture 18

7 User Manual 22

8 Conclusion 23

9 Reference 24

Figure 1: Architecture of smart-home 10
Figure 2: Flow diagram of motion detection and display photos 11
Figure 3: Flow diagram of face detection and display photos 11
Figure 4: Use case diagram behind the Smart Home 12
Figure 5: Use case diagram for the monitoring web portal 12
Figure 6: POM.XML file and Dependencies 13
Figure 7: Flow diagram of Controller Class 14
Figure 8: Controller Class 14
Figure 9: Flow diagram of Motion Detector Class 15
Figure 10: Motion Detector Class 15
Figure 11: Flow diagram of Face Detector Class 16
Figure 12: Face Detector Class 16
Figure 13: Flow diagram of Application Class 17
Figure 14: Application Class 17
Figure 15: Getter And Setter Class 18
Figure 16:Message of CoAP (JASON) 18
Figure 17: Message of HTTP (XML) 19
Figure 18: Wire Shark Capture for CoAP 19
Figure 19: Wire Shark Capture for CoAP Payload 20
Figure 20: Wire Shark Capture for HTTP 20
Figure 21: Wire Shark Capture for HTTP Payload 21
Figure 22: Wire Shark Capture for JPGE Payload 21
Figure 23: Digital Photo frame with Photos on Web browser 22
Figure 24: Digital Photo Frame without Photos on Web browser 22

ABSTRACT

This documented paper presents an approach to provide Digital Photo Frame and a simplest way of deploying Internet of Things (IoT) for smart home, together with due consideration given to user convenience in operating the system. The IoT smart home system runs on The Constrained Application Protocol (CoAP) server. This IoT project allows objects to be sensed and controlled remotely across existing network infrastructure, creating opportunities for more direct integration of the physical world into computer-based systems, and resulting in improved efficiency, accuracy and economic benefit.[1]

Keywords— Internet of Things, motion sensor, face detection sensor, Constrained Application Protocol (CoAP)

2 INTRODUCTION

People preserve experiences through many forms. Spoken, written and drawn forms of experiences have long been used to record our legacies, narratives and experiences. With the invention of the camera, our thousand words were captured in one photographic image. Through the photographic image, our traditions and history became visible through time. The story of our lives no longer relied just on words but was accompanied by photographic image of events that supported the stories. These experiences were shared with vivid accuracy, as a viewer who didn't experience the event first hand imagined themselves in those photographic images, among family and friends.

Photo making has become an integral part of how families, with the use of film-based cameras, preserved their legacy and shared narratives of events and experiences with each other. Printed photos decorated homes in photo frames on fireplace mantles, bookshelves or coffee tables. A denser collection of photos can be found in family albums, or stored in multiple shoeboxes. People would give gifts of photo images on mugs or shirts as well as collages or duplicate photos. They could be seen adding messages to the photos in albums or even on the photos themselves as they shared and recorded their thoughts. Ultimately, people display photos in their homes to share narratives and stimulate social interactions.

Advancing technology continued to support families and their endeavors to use photography as a means to support social interactions. Of the many technological contributions to photography, the introduction and rapid acceptance of digital cameras created a paradigm shift in the way people recorded and reviewed images of everyday life. From the invention of the camera obscure [1] to the modern day film-based cameras, cameras allowed people to more easily capture, reproduce, and share visual narratives. However, the time and the money needed to process the film and print the pictures strongly influenced people's behavior. Digital cameras invoked a change by allowing instant review of photos taken, easier duplication and sharing, while still supporting the overall need to capture and share visual narratives.

People's reasons for recording images have not changed. However, technology has changed to support and enhance their experiences in making photos. As film-based cameras became more robust, easier to use and cheaper, people expanded the role of photos from capturing special events to capturing everyday moments in life. People shared photos by sending them through the mail, constructing albums, and displaying them in photo frames in their homes and offices. While digital photography has changed the capture and electronic sharing of photos, the construction of albums and the display of photos in the home have not been greatly affected.

The home computer has become a digital archive, taking on storage, retrieval and display duties; however, its location in the home and its aesthetic appearance limit the type of sharing and social interactions that happen. For presentation most people still print out physical photos for display and interaction in the home. People do not often change the photos because of the effort involved. Somehow, the digital photos haven't made the full transition to support existing behaviors of people when sharing the photos. Furthermore, current digital photo frames have not yet addressed these needs in the home. The increase in the number of digital photos people are acquiring makes digital display in the home desirable.

Smart digital photo frames can provide great opportunities for increasing social interactions in the home that will improve the experience of people’s lives by supporting and increasing the emotional connections among them. This is the opportunity IoT seeks to explore and, ultimately, complete the transition of digital photos displayed at homes. As traditionally printed photos supported rich social interactions, this project hopes to support these rich social interactions and co-experience through mediating digital photos displayed electronically. The project follows a User Centered Design process and demonstrates three methods in which the digital photo frame can support social interactions: ability to rapidly display photos of people in the home to increasing social interactions, automated updating of photos in the home to make the photos more present and to stimulate reminiscing of the recent and the distant past, and ability to spontaneously share photos and memories of photos with those at a distance directly from the digital photo frame. To support existing user behavior of sharing social narratives at home, the digital photos will have to be displayed in the context of how photos are already viewed at homes using albums and photo frames. Using digital photos to display in the home allows for rich interaction possibilities such as ease of selecting and displaying, annotation and sharing. Furthermore, the digital photos can be organized in the context of social roles of the family (i.e. father, mother, son, grandmother) in addition to time, event and location. Accessing, organizing and sharing digital photos support today’s families who are already engaged in activities with traditionally printed photos. The design opportunities come from addressing those interactions with digital photos for the home.[3]

3 IMPORTANT TOOLS

3.1 INTERNET OF THINGS (IOT)

The Internet of Things (IoT) is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

The definition of the Internet of Things has evolved due to the convergence of multiple technologies, real-time analytics, machine learning, commodity sensors, and embedded systems. Traditional fields of embedded systems, wireless sensor networks, control systems, automation (including home and building automation), and others all contribute to enabling the Internet of Things. In the consumer market, IoT technology is most synonymous with products pertaining to the concept of the "smart home", covering devices and appliances (such as lighting fixtures, thermostats, home security systems and cameras, and other home appliances) that support one or more common ecosystems, and can be controlled via devices associated with that ecosystem, such as smartphones and smart speakers.[4]

3.2 CONSTRAINED APPLICATION PROTOCOL (COAP)

Constrained Application Protocol (CoAP) is a specialized Internet Application Protocol for constrained devices, as define. It enables those constrained devices called "nodes" to communicate with the wider Internet using similar protocols. CoAP is designed for use between devices on the same constrained network (e.g., low-power, lossy networks), between devices and general nodes on the Internet, and between devices on different constrained networks both joined by an internet. CoAP is also being used via other mechanisms, such as SMS on mobile communication networks.

CoAP is a service layer protocol that is intended for use in resource-constrained internet devices, such as wireless sensor network nodes. CoAP is designed to easily translate to HTTP for simplified integration with the web, while also meeting specialized requirements such as multicast support, very low overhead, and simplicity. Multicast, low overhead, and simplicity are extremely important for Internet of Things (IoT) and Machine-to-Machine (M2M) devices, which tend to be deeply embedded and have much less memory and power supply than traditional internet devices have. Therefore, efficiency is very important. CoAP can run on most devices that support UDP or a UDP analogue.[5]

3.3 Motion Sensor

A motion detector is a device that detects moving objects, particularly people. Such a device is often integrated as a component of a system that automatically performs a task or alerts a user of motion in an area. They form a vital component of security, automated lighting control, home control, energy efficiency and other useful systems.

An electronic motion detector contains an optical, microwave, or acoustic sensor, and in many cases a transmitter for illumination. However, a passive sensor senses a signature only from the moving object via emission or reflection, i.e., it can be emitted by the object, or by some ambient emitter such as the sun or a radio station of sufficient strength. Changes in the optical, microwave, or acoustic field in the device's proximity are interpreted by the electronics based.[6]

3.4 Face Detection Sensor

A facial recognition system is a technology capable of identifying or verifying a person from a digital image or a video frame from a video source. There are multiple methods in which facial recognition systems work, but in general, they work by comparing selected facial features from given image with faces within a database. It is also described as a Biometric Artificial Intelligence based application that can uniquely identify a person by analyzing patterns based on the person's facial textures and shape. [7]

4 SUPPORTING PROGRAMMING LANGUAGE

A programming tool or software development tool is a computer program that software developers use to create, debug, maintain, or otherwise support other programs and applications. The term usually refers to relatively simple programs, that can be combined together to accomplish a task, much as one might use multiple hand tools to fix a physical object. The most basic tools are a source code editor and a compiler or interpreter, which are used ubiquitously and continuously. Other tools are used more or less depending on the language, development methodology, and individual engineer, and are often used for a discrete task, like a debugger or profiler. Here, I use

- Spring Boots
- HTML
- CSS
- JavaScript
- Ajax

5 PRINCIPLES OF PROJECT

Smart home is the residential extension of building automation and involves the control and automation of all its embedded technology. It defines a residence that has appliances, lighting, heating, air conditioning, TVs, computers, entertainment systems, big home appliances such as washers/dryers and refrigerators/freezers, security and camera systems capable of communicating with each other and being controlled remotely by a time schedule, phone, mobile or internet. These systems consist of switches and sensors connected to a central hub controlled by the home resident using wall-mounted terminal or mobile unit connected to internet cloud services.

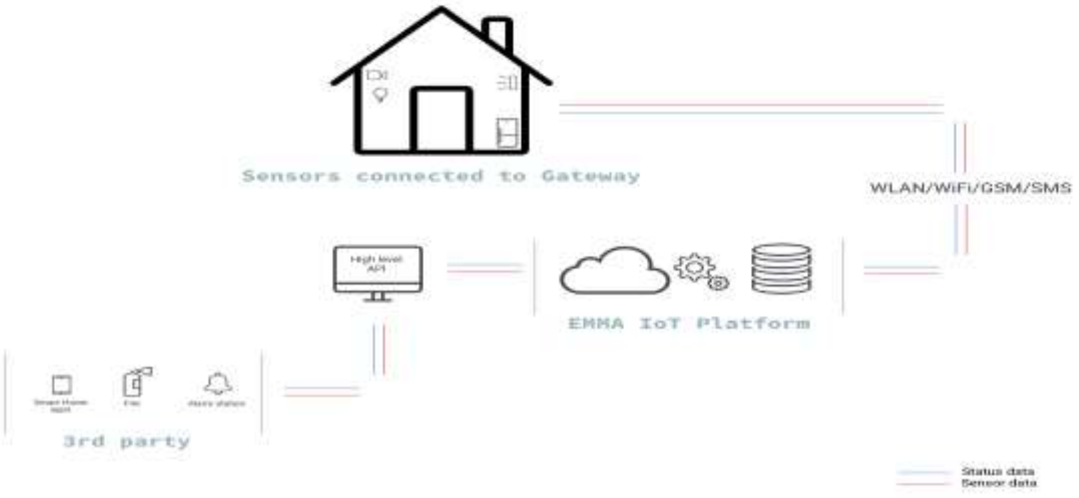


Figure 1: Architecture of smart-home

Here I have worked for automation of digital photo frame for smart home with the help of IoT device and two sensor. As a matter of fact, I tried to include two case in my project with the help of this two sensor. The cases are

- Human exist in the room (Motion Sensor)
- Identify friends and family persons (Face Detection Sensor)

5.1.1 Human exist in the room (Motion Sensor)

In the room a motion sensor will be installed and configured with the Digital photo frame. Then motion sensor will identify the presence of human in the room. If the sensor find any movement, then it detect and send the message to the digital photo frame through the IoT. As a result, the digital photo frame start to show the sliding of photos.

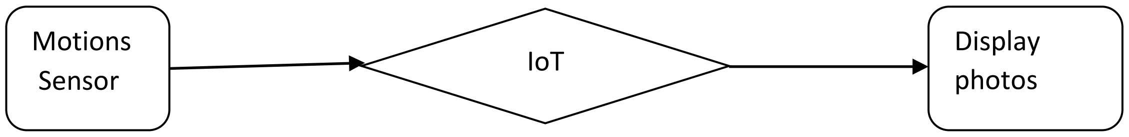


Figure 2: Flow diagram of motion detection and display photos

5.1.2 Identify friends and family persons (Face Detection Sensor)

I try to implement a new feature for Digital Photo frame by using face detection sensor. This sensor will help to show the pictures of the family and friends who will be present in the room at that time. The sensor will detect the person and if it can find the name of the person, then it will deliver the name to the IoT. As a result, the IoT will inform the Digital photo frame to show the pictures of that preson.

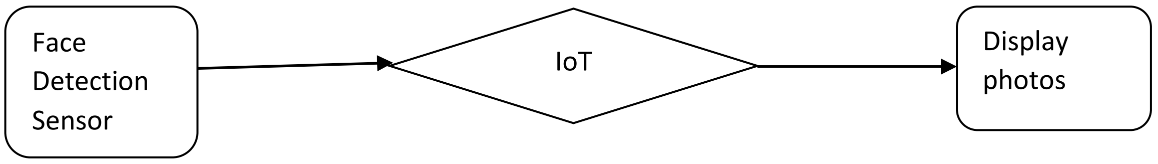


Figure 3: Flow diagram of face detection and display photos

5.2 Modeling of Use Case

The Use case system for the smart home digital photo frame is depicted in the Fig. 4. This case system is developed in Unified Modelling Language (UML). This helps in analyzing the interaction between the system and user. The interactions such as motion sensing and face detection capability, sensed data processing and message delivery are concerned with a IoT. Whereas the receive status, home digital photo frame update is the information related to the user. Fig. 3. shows the use case diagram for the interaction behind the monitoring web portal. The possible communications between the system and user.
:

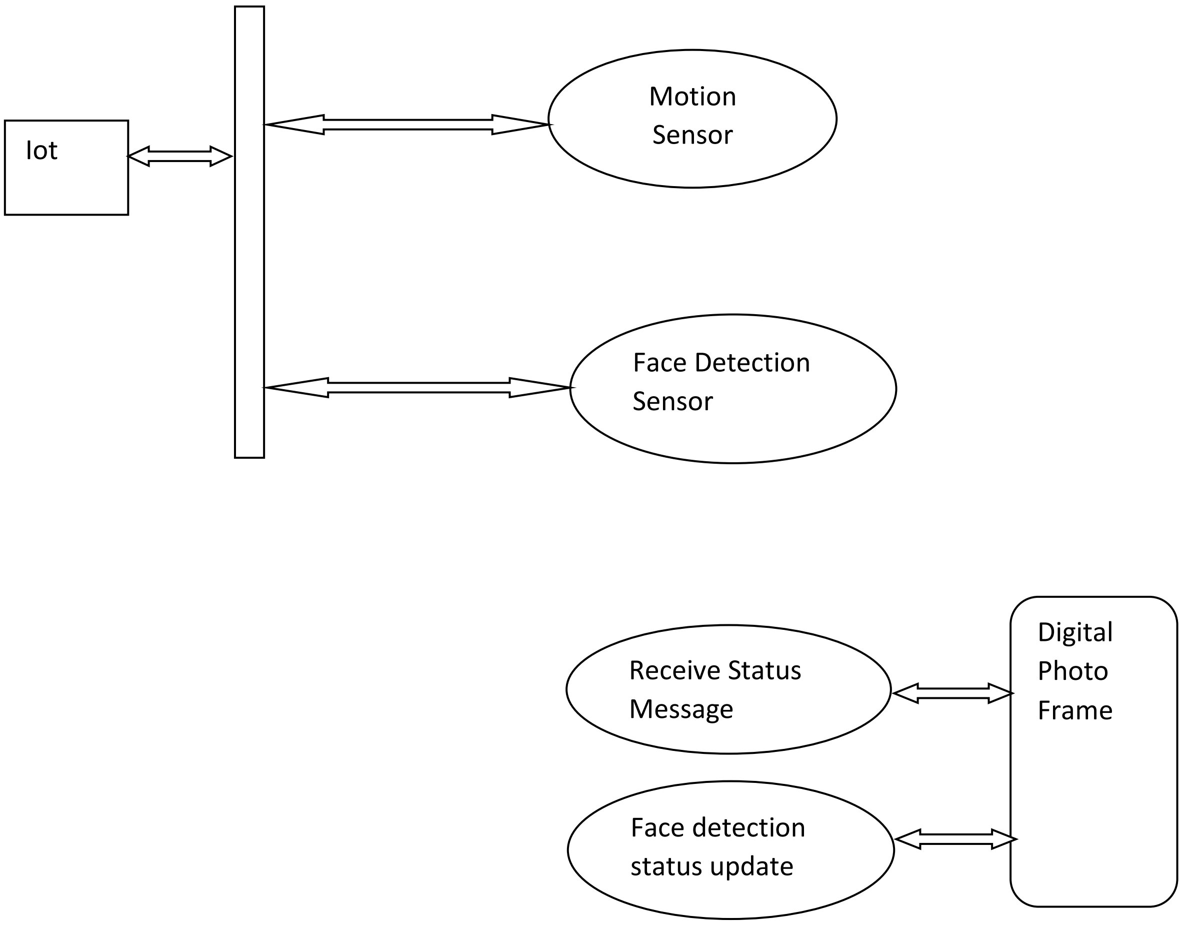


Figure 4: Use case diagram behind the Smart Home

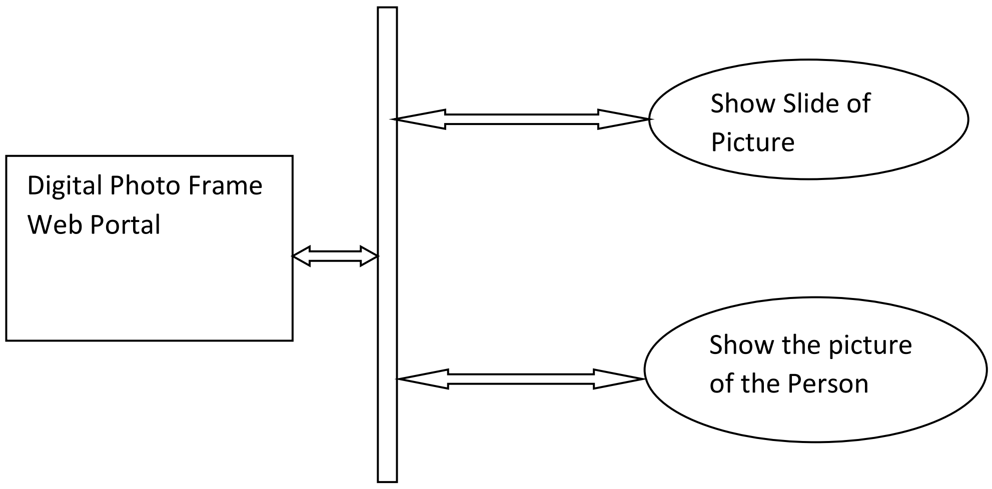

Digital Photo Frame Web Portal
Show Slide of Picture
Show the picture of the Person

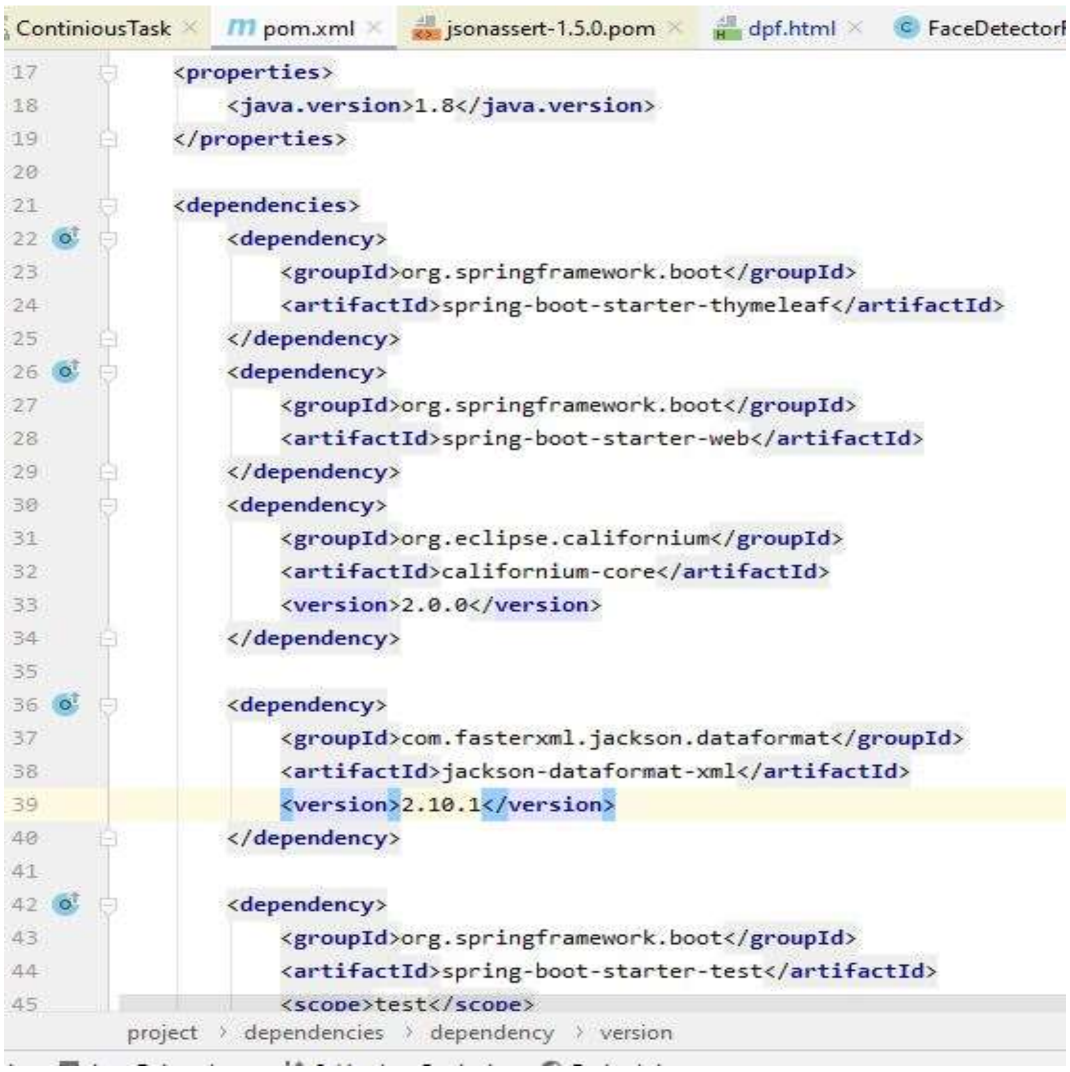
Figure 5: Use case diagram for the monitoring web portal

6 Project Architecture

In this section I describe the project architecture that I solve in order to find out the best strategies for users. Here I try to use flow diagram and snap of my code for description.

6.1 Dependencies

POM stands for "Project Object Model". It is an XML representation of a Maven project held in a file named pom.xml. When in the presence of Maven folks, speaking of a project is speaking in the philosophical sense, beyond a mere collection of files containing code. A project contains configuration files, as well as the developers involved and the roles they play, the defect tracking system, the organization and licenses, the URL of where the project lives, the project's dependencies, and all of the other little pieces that come into play to give code life. It is a one-stop-shop for all things concerning the project



```xml
    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.eclipse.californium</groupId>
            <artifactId>californium-core</artifactId>
            <version>2.0.0</version>
        </dependency>

        <dependency>
            <groupId>com.fasterxml.jackson.dataformat</groupId>
            <artifactId>jackson-dataformat-xml</artifactId>
            <version>2.10.1</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
```

Figure 6: POM.XML file and Dependencies

6.2 CLASSES

For this project I used seven java classes. The classes are describe here –

6.2.1 DigitalPhotoFrameControlar

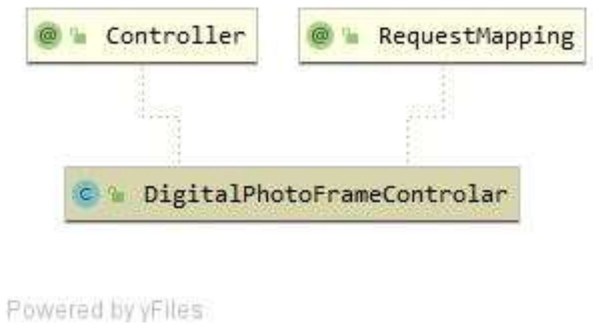
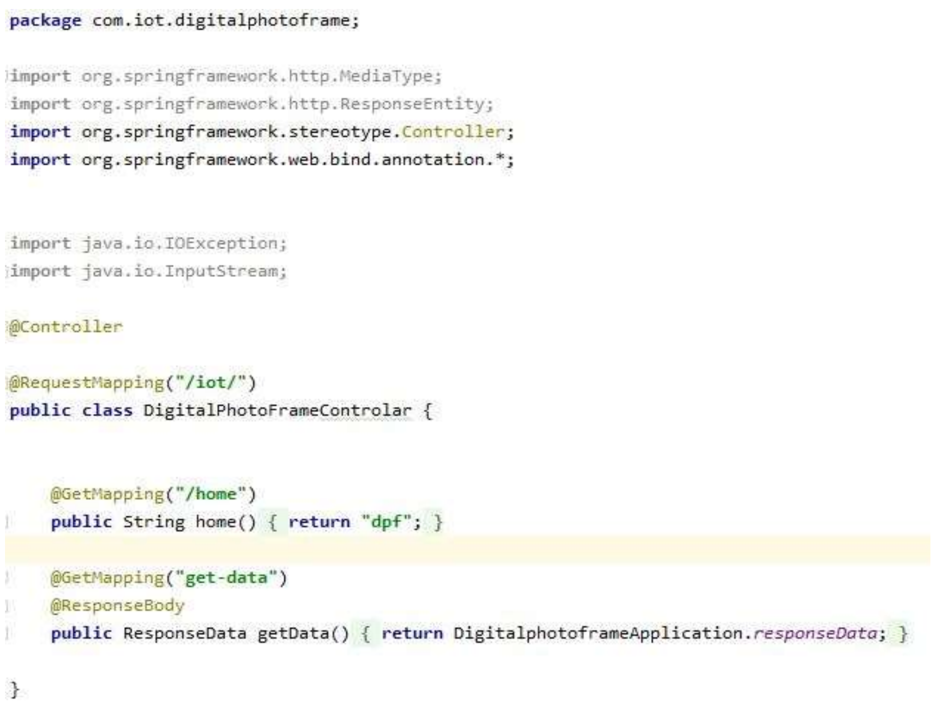


Figure 7: Flow diagram of Controller Class

It is a controller class, typically, in Spring MVC, we write a controller class to handle requests coming from the client. Then, the controller invokes a business class to process business-related tasks, and then redirects the client to a logical view name, which is resolved by Spring's dispatcher servlet in order to render results or output



```java
package com.iot.digitalphotoframe;

import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;


import java.io.IOException;
import java.io.InputStream;

@Controller

@RequestMapping("/iot/")
public class DigitalPhotoFrameControlar {


    @GetMapping("/home")
    public String home() { return "dpf"; }

    @GetMapping("get-data")
    @ResponseBody
    public ResponseData getData() { return DigitalphotoframeApplication.responseData; }

}
```

Figure 8: Controller Class

6.2.2 MotionDetector

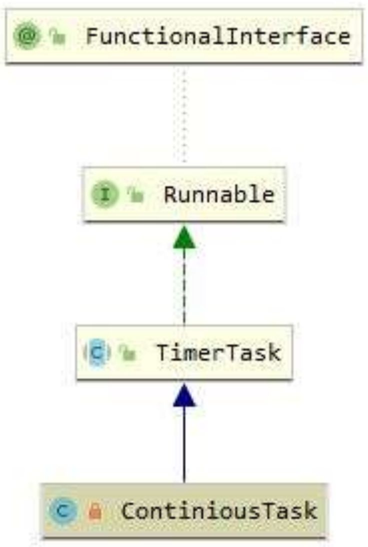
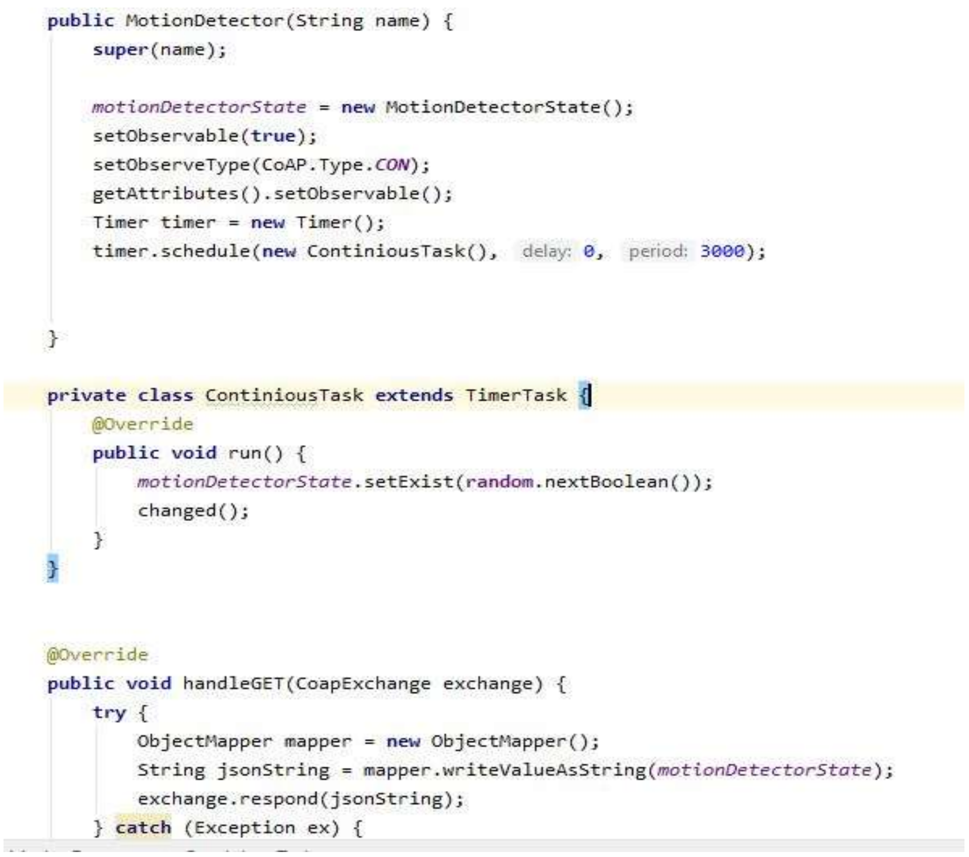


Figure 9: Flow diagram of Motion Detector Class

This class will return the data which comes from motion sensor device to the IoT. So That the digital photo frame can show the photo slide. If there is any person in the room then the sensor will return the true value, Otherwise it return a false value. Here, I used a random value as a sensor data.



```
public MotionDetector(String name) {
    super(name);

    motionDetectorState = new MotionDetectorState();
    setObservable(true);
    setObserveType(CoAP.Type.CON);
    getAttributes().setObservable();
    Timer timer = new Timer();
    timer.schedule(new ContiniousTask(),  delay: 0,  period: 3000);


}

private class ContiniousTask extends TimerTask {
    @Override
    public void run() {
        motionDetectorState.setExist(random.nextBoolean());
        changed();
    }
}


@Override
public void handleGET(CoapExchange exchange) {
    try {
        ObjectMapper mapper = new ObjectMapper();
        String jsonString = mapper.writeValueAsString(motionDetectorState);
        exchange.respond(jsonString);
    } catch (Exception ex) {
```

Figure 10: Motion Detector Class

6.2.3 FaceDetectorSensor

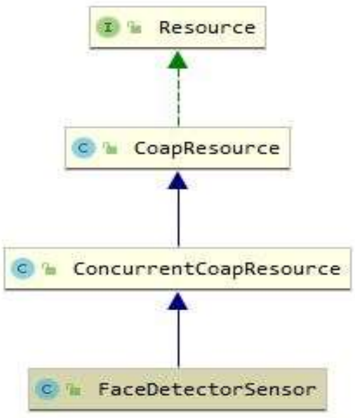
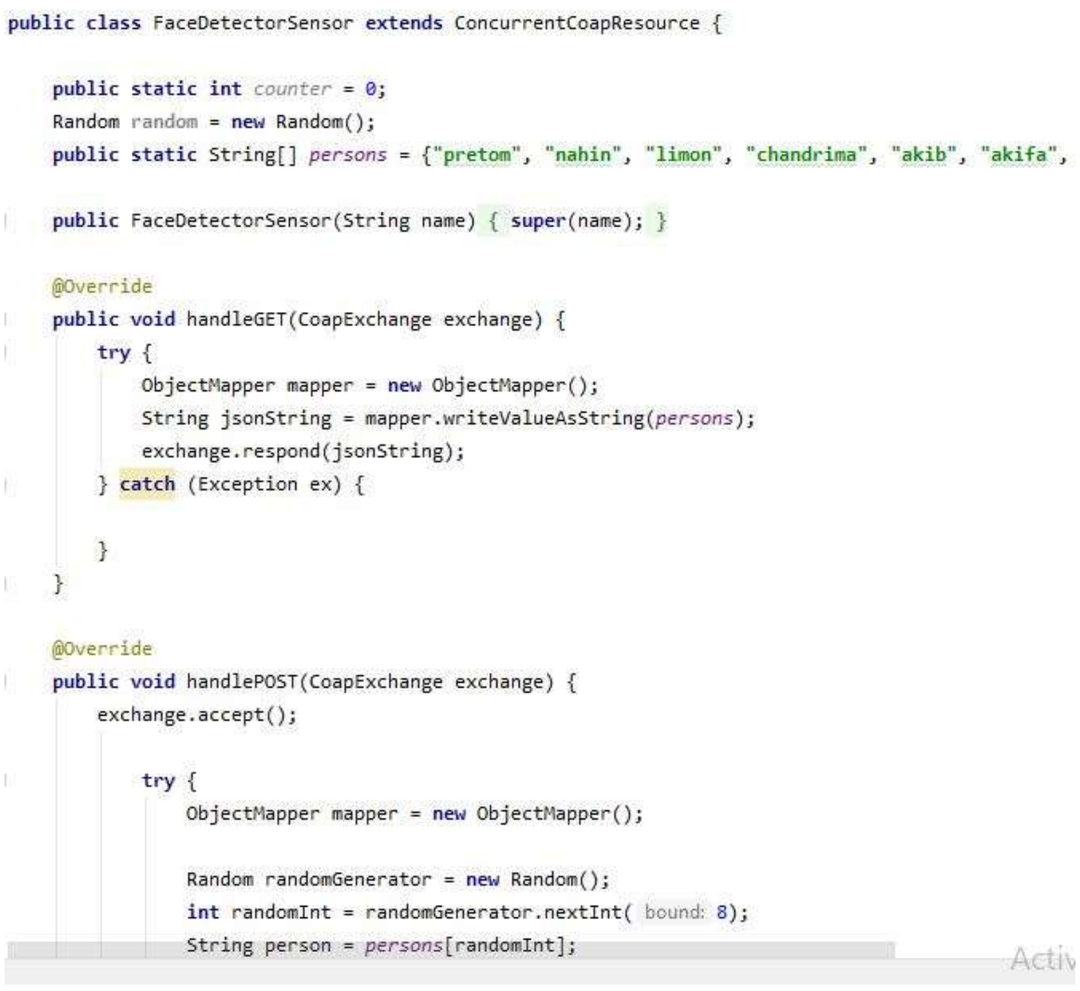


Figure 11: Flow diagram of Face Detector Class

Face detector sensor will detect the face and if the face is familier with the ditector then it will return a name of the person to the IoT. Then the digital photo frame will show his picture. Here, a string is define with some of the name of my friend and a random function will return a name if the motion picture senses person in the room.



```
public class FaceDetectorSensor extends ConcurrentCoapResource {

    public static int counter = 0;
    Random random = new Random();
    public static String[] persons = {"pretom", "nahin", "limon", "chandrima", "akib", "akifa",

    public FaceDetectorSensor(String name) { super(name); }

    @Override
    public void handleGET(CoapExchange exchange) {
        try {
            ObjectMapper mapper = new ObjectMapper();
            String jsonString = mapper.writeValueAsString(persons);
            exchange.respond(jsonString);
        } catch (Exception ex) {

        }
    }

    @Override
    public void handlePOST(CoapExchange exchange) {
        exchange.accept();

            try {
                ObjectMapper mapper = new ObjectMapper();

                Random randomGenerator = new Random();
                int randomInt = randomGenerator.nextInt( bound: 8);
                String person = persons[randomInt];
```

Figure 12: Face Detector Class

6.2.4 DigitalphotoframeApplication

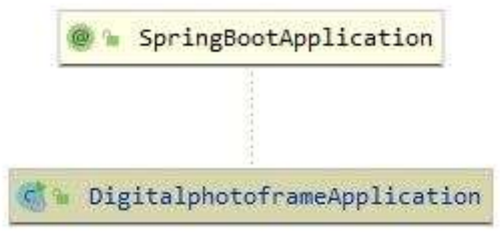
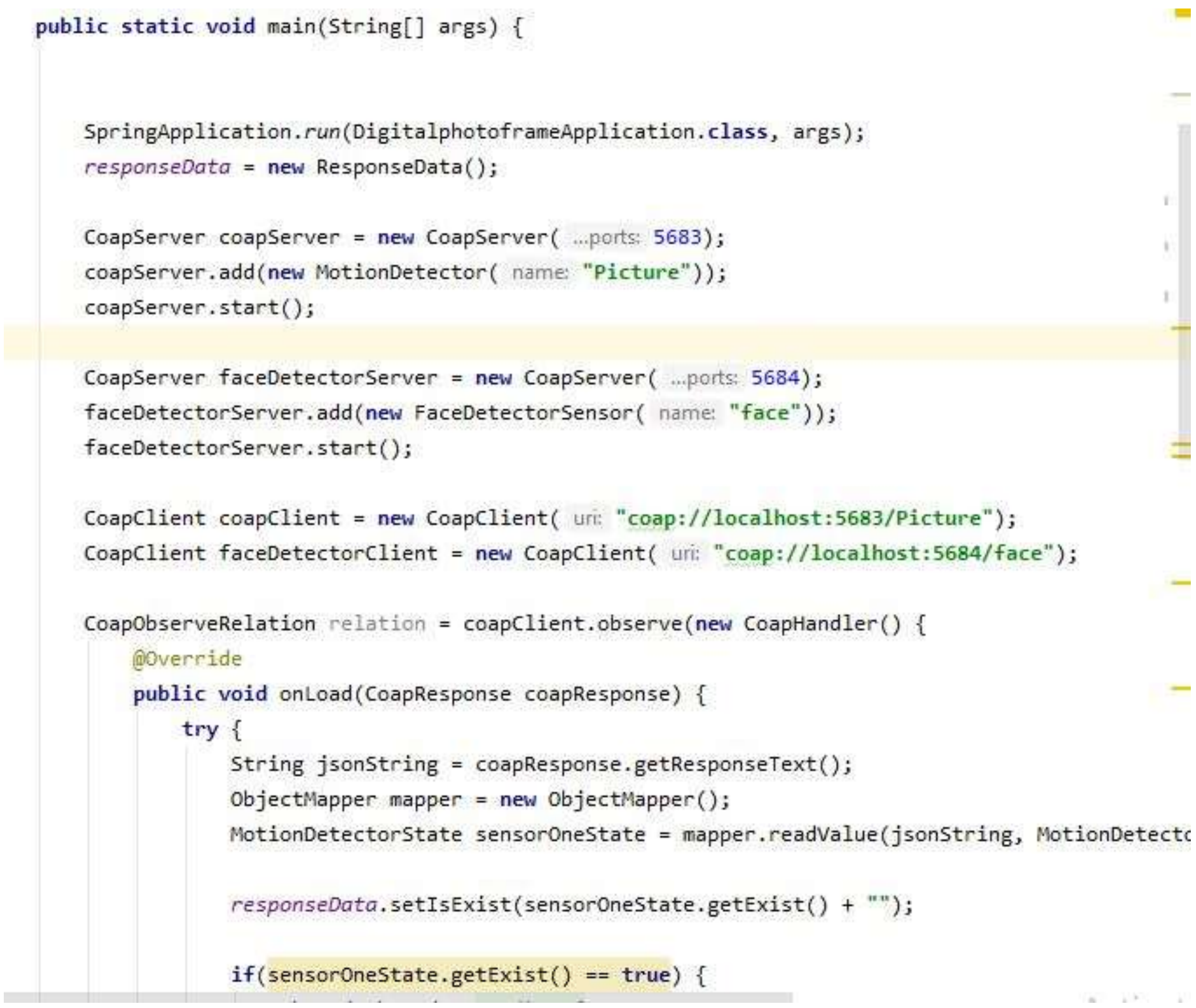


Figure 13: Flow diagram of Application Class

This class with the main method as the starting point of the spring boot Application. Putting simply, this is where the system knows this is a spring boot application. CoAP Server and Client are initialized and configured here at the port ‘5683’ for the motion sensor and ‘5684’ for face detector.

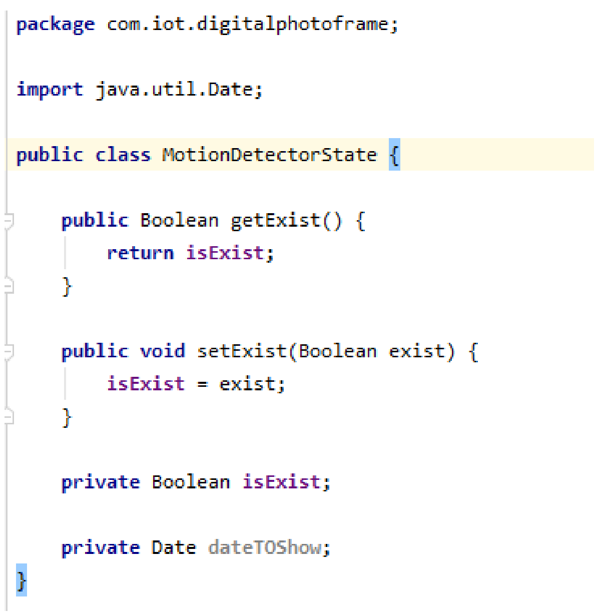


```
public static void main(String[] args) {


    SpringApplication.run(DigitalphotoframeApplication.class, args);
    responseData = new ResponseData();

    CoapServer coapServer = new CoapServer( ...ports: 5683);
    coapServer.add(new MotionDetector( name: "Picture"));
    coapServer.start();

    CoapServer faceDetectorServer = new CoapServer( ...ports: 5684);
    faceDetectorServer.add(new FaceDetectorSensor( name: "face"));
    faceDetectorServer.start();

    CoapClient coapClient = new CoapClient( uri: "coap://localhost:5683/Picture");
    CoapClient faceDetectorClient = new CoapClient( uri: "coap://localhost:5684/face");

    CoapObserveRelation relation = coapClient.observe(new CoapHandler() {
        @Override
        public void onLoad(CoapResponse coapResponse) {
            try {
                String jsonString = coapResponse.getResponseText();
                ObjectMapper mapper = new ObjectMapper();
                MotionDetectorState sensorOneState = mapper.readValue(jsonString, MotionDetectc

                responseData.setIsExist(sensorOneState.getExist() + "");

                if(sensorOneState.getExist() == true) {
```

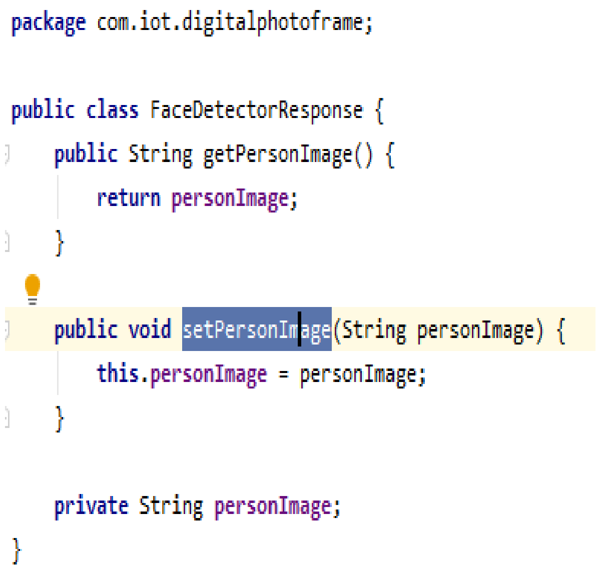
Figure 14: Application Class

6.2.5 Getter And Setter

You can annotate any field with @Getter and/or @Setter, to let lombok generate the default getter/setter automatically. A default getter simply returns the field, and is named. Here, getExist() if the field is called isExist the field's type is Boolean. A default setter is named setExist() if the field is called isExist, returns void. Same as to other getPersonImage() and setPersonImage() the field types are string and void.



```
package com.iot.digitalphotoframe;

import java.util.Date;

public class MotionDetectorState {

    public Boolean getExist() {
        return isExist;
    }

    public void setExist(Boolean exist) {
        isExist = exist;
    }

    private Boolean isExist;

    private Date dateTOShow;
}
```

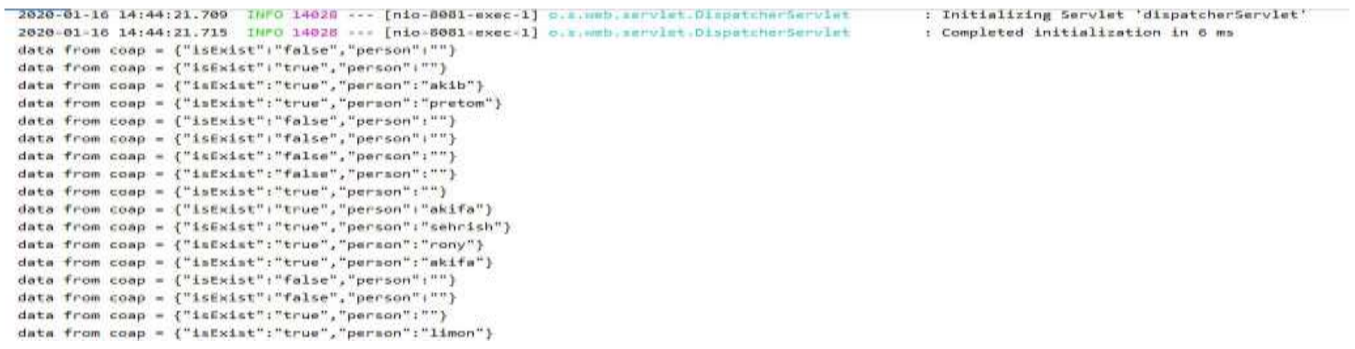


```
package com.iot.digitalphotoframe;

public class FaceDetectorResponse {
    public String getPersonImage() {
        return personImage;
    }

    public void setPersonImage(String personImage) {
        this.personImage = personImage;
    }

    private String personImage;
}
```

Figure 15: Getter And Setter Class

6.3 OUTPUT FILES AND WIRE SHARK CAPTURE

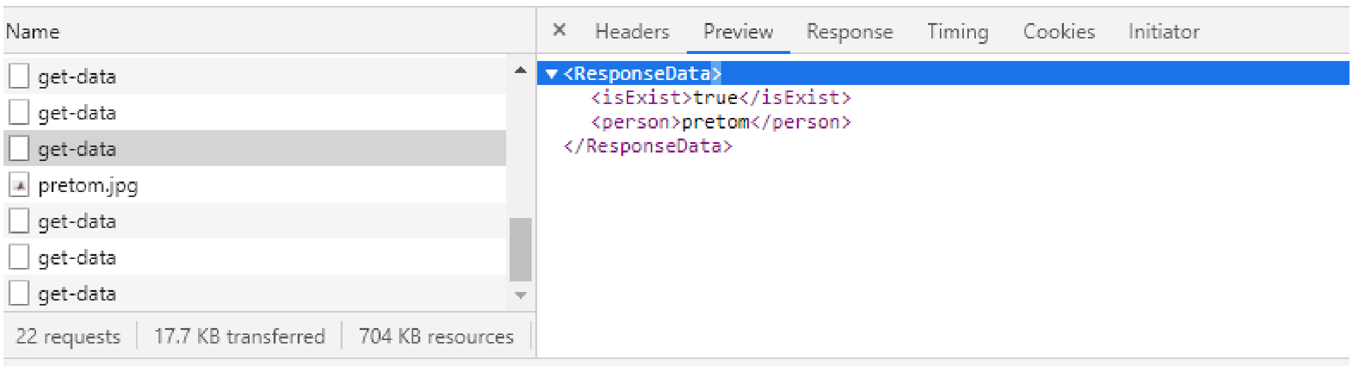
The sensor send the message to the IoT using the CoAP protocol. And decide the action of the device depend on this result. On the other hand, when a browser send a request to the server it follows the HTTP protocols. Here, Payload inside the CoAP messages need to be formatted using the encoding schemes JSON Figure 8.IsExist contain the value of motion sensor in Boolean, Beside this, Person contain the name of the person that detect on the face detector.



```
2020-01-16 14:44:21.709  INFO 14028 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2020-01-16 14:44:21.715  INFO 14028 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 6 ms
data from coap = {"isExist":"false","person":""}
data from coap = {"isExist":"true","person":""}
data from coap = {"isExist":"true","person":"akib"}
data from coap = {"isExist":"true","person":"pretom"}
data from coap = {"isExist":"false","person":""}
data from coap = {"isExist":"false","person":""}
data from coap = {"isExist":"false","person":""}
data from coap = {"isExist":"false","person":""}
data from coap = {"isExist":"true","person":""}
data from coap = {"isExist":"true","person":"akifa"}
data from coap = {"isExist":"true","person":"sehrish"}
data from coap = {"isExist":"true","person":"rony"}
data from coap = {"isExist":"true","person":"akifa"}
data from coap = {"isExist":"false","person":""}
data from coap = {"isExist":"false","person":""}
data from coap = {"isExist":"true","person":""}
data from coap = {"isExist":"true","person":"limon"}
```

Figure 16:Message of CoAP (JASON)

And Payload inside the HTTP messages need to be formatted using the encoding schemes XML Figure 9.



Name
get-data
get-data
get-data
pretom.jpg
get-data
get-data
get-data

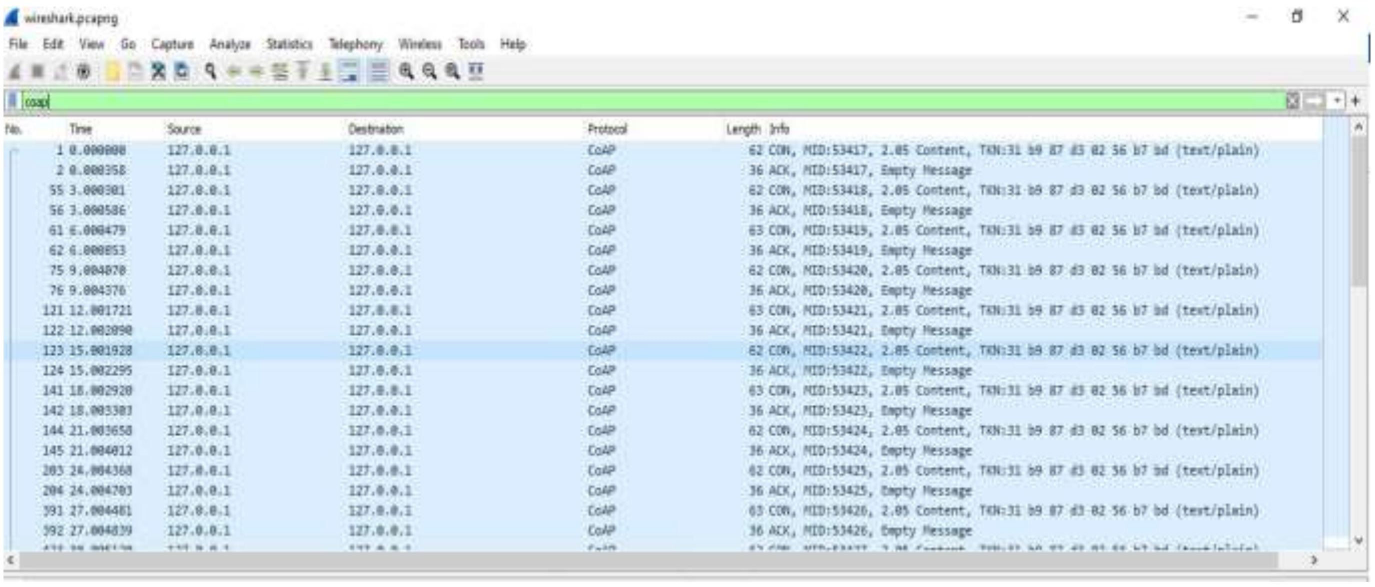
22 requests | 17.7 KB transferred | 704 KB resources

× Headers Preview Response Timing Cookies Initiator

```
▼<ResponseData>
   <isExist>true</isExist>
   <person>pretom</person>
 </ResponseData>
```

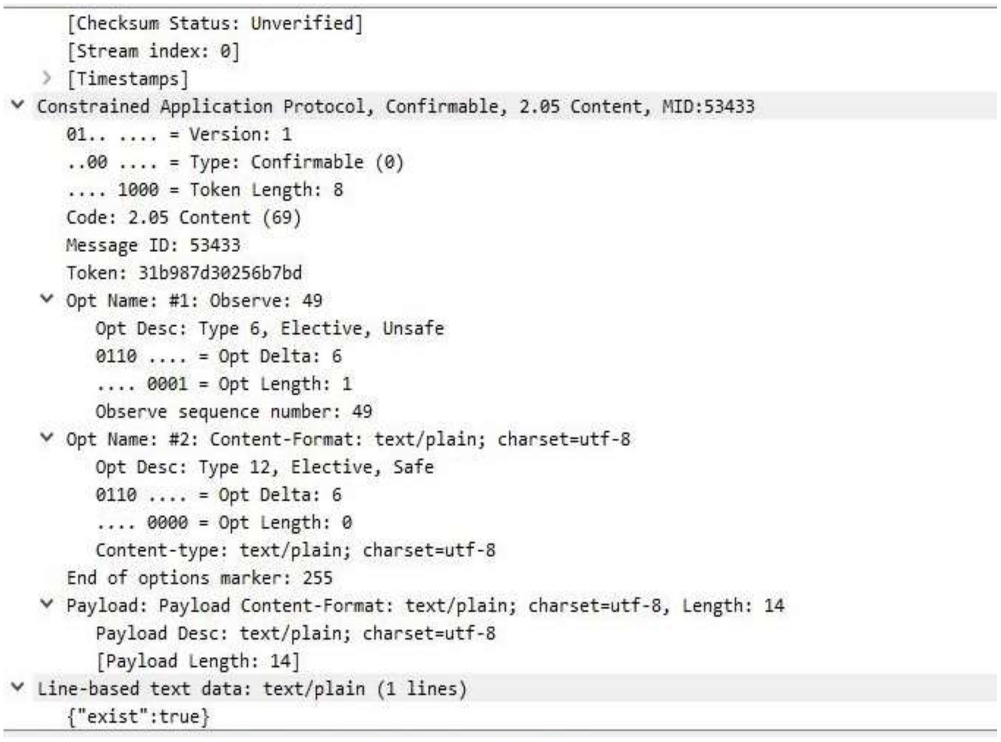
Figure 17: Message of HTTP (XML)

The traffic between the Coap and HTTP of the system is captured by the Wire shark. The result of the wire shark trace is fully satisfied and successful. Here I showing some snap of my wire shark capture.


wireshark.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
coap

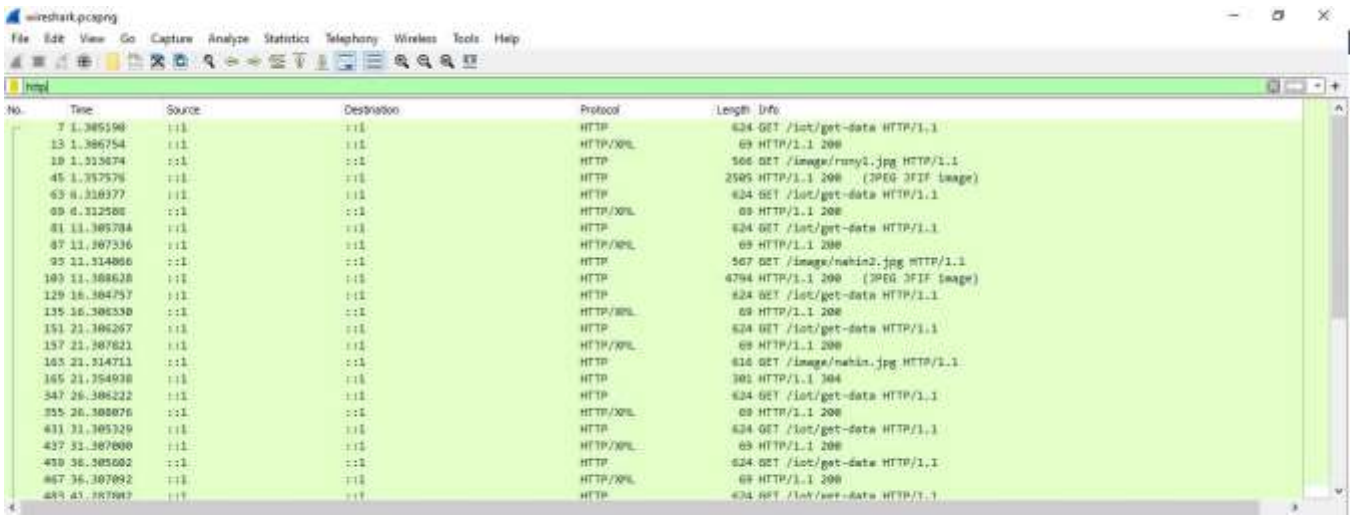
```
No.   Time       Source     Destination  Protocol  Length Info
  1 0.000000   127.0.0.1  127.0.0.1    CoAP      62 CON, MID:53417, 2.05 Content, TKN:31 b9 87 d3 02 56 b7 bd (text/plain)
  2 0.000358   127.0.0.1  127.0.0.1    CoAP      36 ACK, MID:53417, Empty Message
 55 3.000301   127.0.0.1  127.0.0.1    CoAP      62 CON, MID:53418, 2.05 Content, TKN:31 b9 87 d3 02 56 b7 bd (text/plain)
 56 3.000586   127.0.0.1  127.0.0.1    CoAP      36 ACK, MID:53418, Empty Message
 61 6.000479   127.0.0.1  127.0.0.1    CoAP      63 CON, MID:53419, 2.05 Content, TKN:31 b9 87 d3 02 56 b7 bd (text/plain)
 62 6.000853   127.0.0.1  127.0.0.1    CoAP      36 ACK, MID:53419, Empty Message
 75 9.004070   127.0.0.1  127.0.0.1    CoAP      62 CON, MID:53420, 2.05 Content, TKN:31 b9 87 d3 02 56 b7 bd (text/plain)
 76 9.004376   127.0.0.1  127.0.0.1    CoAP      36 ACK, MID:53420, Empty Message
121 12.001721  127.0.0.1  127.0.0.1    CoAP      63 CON, MID:53421, 2.05 Content, TKN:31 b9 87 d3 02 56 b7 bd (text/plain)
122 12.003090  127.0.0.1  127.0.0.1    CoAP      36 ACK, MID:53421, Empty Message
123 15.001928  127.0.0.1  127.0.0.1    CoAP      62 CON, MID:53422, 2.05 Content, TKN:31 b9 87 d3 02 56 b7 bd (text/plain)
124 15.002295  127.0.0.1  127.0.0.1    CoAP      36 ACK, MID:53422, Empty Message
141 18.002920  127.0.0.1  127.0.0.1    CoAP      63 CON, MID:53423, 2.05 Content, TKN:31 b9 87 d3 02 56 b7 bd (text/plain)
142 18.003303  127.0.0.1  127.0.0.1    CoAP      36 ACK, MID:53423, Empty Message
144 21.003650  127.0.0.1  127.0.0.1    CoAP      62 CON, MID:53424, 2.05 Content, TKN:31 b9 87 d3 02 56 b7 bd (text/plain)
145 21.004012  127.0.0.1  127.0.0.1    CoAP      36 ACK, MID:53424, Empty Message
203 24.004368  127.0.0.1  127.0.0.1    CoAP      62 CON, MID:53425, 2.05 Content, TKN:31 b9 87 d3 02 56 b7 bd (text/plain)
204 24.004703  127.0.0.1  127.0.0.1    CoAP      36 ACK, MID:53425, Empty Message
391 27.004481  127.0.0.1  127.0.0.1    CoAP      63 CON, MID:53426, 2.05 Content, TKN:31 b9 87 d3 02 56 b7 bd (text/plain)
392 27.004839  127.0.0.1  127.0.0.1    CoAP      36 ACK, MID:53426, Empty Message
```

Figure 18: Wire Shark Capture for CoAP



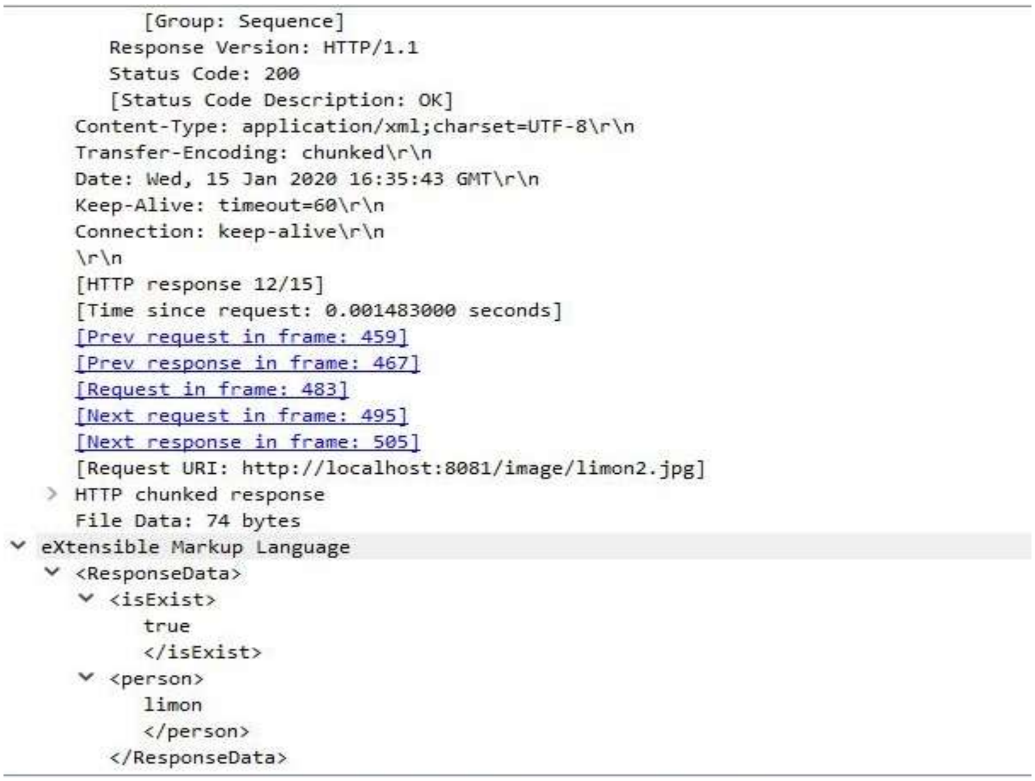
```
        [Checksum Status: Unverified]
        [Stream index: 0]
      > [Timestamps]
  ˅ Constrained Application Protocol, Confirmable, 2.05 Content, MID:53433
        01.. .... = Version: 1
        ..00 .... = Type: Confirmable (0)
        .... 1000 = Token Length: 8
        Code: 2.05 Content (69)
        Message ID: 53433
        Token: 31b987d30256b7bd
      ˅ Opt Name: #1: Observe: 49
           Opt Desc: Type 6, Elective, Unsafe
           0110 .... = Opt Delta: 6
           .... 0001 = Opt Length: 1
           Observe sequence number: 49
      ˅ Opt Name: #2: Content-Format: text/plain; charset=utf-8
           Opt Desc: Type 12, Elective, Safe
           0110 .... = Opt Delta: 6
           .... 0000 = Opt Length: 0
           Content-type: text/plain; charset=utf-8
        End of options marker: 255
      ˅ Payload: Payload Content-Format: text/plain; charset=utf-8, Length: 14
           Payload Desc: text/plain; charset=utf-8
           [Payload Length: 14]
  ˅ Line-based text data: text/plain (1 lines)
        {"exist":true}
```

Figure 19: Wire Shark Capture for CoAP Payload


wireshark.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

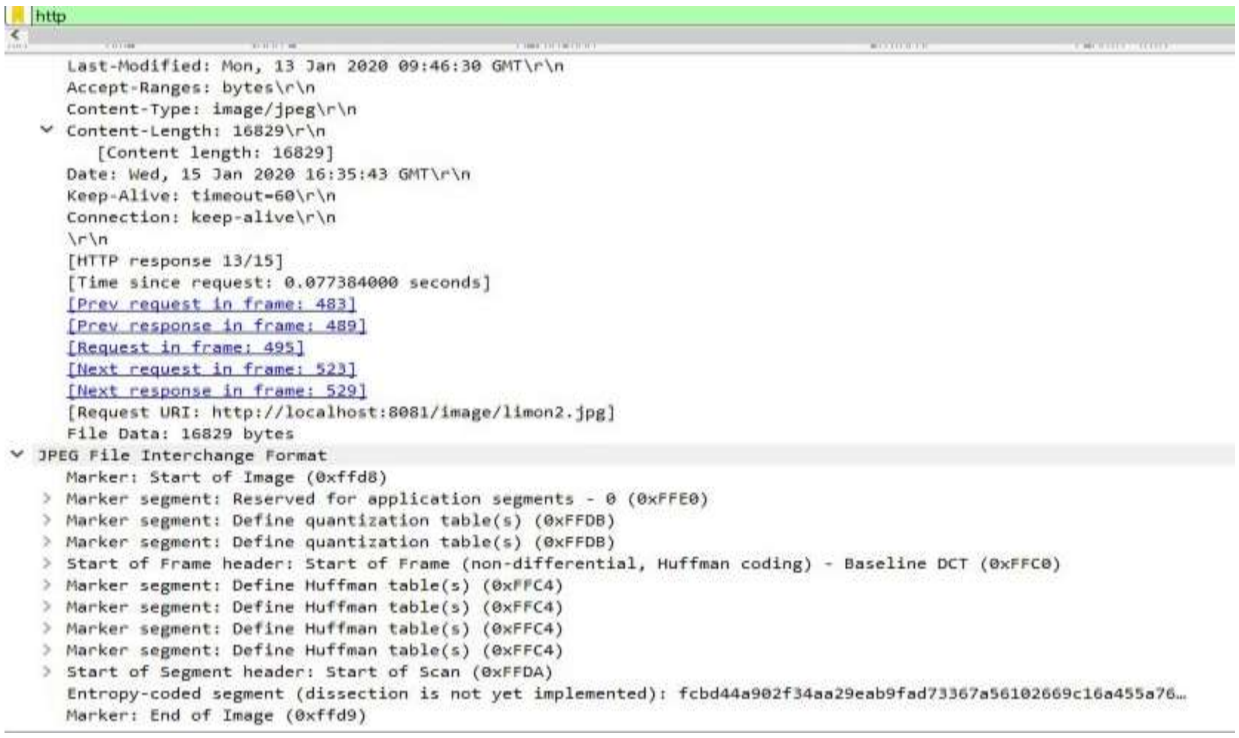
```
http
No.     Time        Source   Destination   Protocol   Length Info
    7 1.305190      ::1      ::1           HTTP          624 GET /iot/get-data HTTP/1.1
   13 1.306754      ::1      ::1           HTTP/XML       69 HTTP/1.1 200
   19 1.313674      ::1      ::1           HTTP          506 GET /image/rony1.jpg HTTP/1.1
   45 1.357576      ::1      ::1           HTTP         2505 HTTP/1.1 200  (JPEG JFIF image)
   63 6.310377      ::1      ::1           HTTP          624 GET /iot/get-data HTTP/1.1
   69 6.312588      ::1      ::1           HTTP/XML       69 HTTP/1.1 200
   81 11.305784     ::1      ::1           HTTP          624 GET /iot/get-data HTTP/1.1
   87 11.307336     ::1      ::1           HTTP/XML       69 HTTP/1.1 200
   95 11.314866     ::1      ::1           HTTP          567 GET /image/nahin2.jpg HTTP/1.1
  103 11.388628     ::1      ::1           HTTP         4794 HTTP/1.1 200  (JPEG JFIF image)
  129 16.304757     ::1      ::1           HTTP          624 GET /iot/get-data HTTP/1.1
  135 16.306330     ::1      ::1           HTTP/XML       69 HTTP/1.1 200
  151 21.306267     ::1      ::1           HTTP          624 GET /iot/get-data HTTP/1.1
  157 21.307821     ::1      ::1           HTTP/XML       69 HTTP/1.1 200
  163 21.314711     ::1      ::1           HTTP          616 GET /image/nahin.jpg HTTP/1.1
  165 21.354938     ::1      ::1           HTTP          301 HTTP/1.1 304
  347 26.306222     ::1      ::1           HTTP          624 GET /iot/get-data HTTP/1.1
  355 26.308076     ::1      ::1           HTTP/XML       69 HTTP/1.1 200
  431 31.305329     ::1      ::1           HTTP          624 GET /iot/get-data HTTP/1.1
  437 31.307080     ::1      ::1           HTTP/XML       69 HTTP/1.1 200
  459 36.305602     ::1      ::1           HTTP          624 GET /iot/get-data HTTP/1.1
  467 36.307092     ::1      ::1           HTTP/XML       69 HTTP/1.1 200
  483 41.387002     ::1      ::1           HTTP          624 GET /iot/get-data HTTP/1.1
```

Figure 20: Wire Shark Capture for HTTP



```
            [Group: Sequence]
         Response Version: HTTP/1.1
         Status Code: 200
         [Status Code Description: OK]
      Content-Type: application/xml;charset=UTF-8\r\n
      Transfer-Encoding: chunked\r\n
      Date: Wed, 15 Jan 2020 16:35:43 GMT\r\n
      Keep-Alive: timeout=60\r\n
      Connection: keep-alive\r\n
      \r\n
      [HTTP response 12/15]
      [Time since request: 0.001483000 seconds]
      [Prev request in frame: 459]
      [Prev response in frame: 467]
      [Request in frame: 483]
      [Next request in frame: 495]
      [Next response in frame: 505]
      [Request URI: http://localhost:8081/image/limon2.jpg]
    > HTTP chunked response
      File Data: 74 bytes
 ˅ eXtensible Markup Language
    ˅ <ResponseData>
       ˅ <isExist>
            true
            </isExist>
       ˅ <person>
            limon
            </person>
         </ResponseData>
```

Figure 21: Wire Shark Capture for HTTP Payload



```
     Last-Modified: Mon, 13 Jan 2020 09:46:30 GMT\r\n
     Accept-Ranges: bytes\r\n
     Content-Type: image/jpeg\r\n
   ˅ Content-Length: 16829\r\n
         [Content length: 16829]
     Date: Wed, 15 Jan 2020 16:35:43 GMT\r\n
     Keep-Alive: timeout=60\r\n
     Connection: keep-alive\r\n
     \r\n
     [HTTP response 13/15]
     [Time since request: 0.077384000 seconds]
     [Prev request in frame: 483]
     [Prev response in frame: 489]
     [Request in frame: 495]
     [Next request in frame: 523]
     [Next response in frame: 529]
     [Request URI: http://localhost:8081/image/limon2.jpg]
     File Data: 16829 bytes
˅ JPEG File Interchange Format
     Marker: Start of Image (0xffd8)
   > Marker segment: Reserved for application segments - 0 (0xFFE0)
   > Marker segment: Define quantization table(s) (0xFFDB)
   > Marker segment: Define quantization table(s) (0xFFDB)
   > Start of Frame header: Start of Frame (non-differential, Huffman coding) - Baseline DCT (0xFFC0)
   > Marker segment: Define Huffman table(s) (0xFFC4)
   > Marker segment: Define Huffman table(s) (0xFFC4)
   > Marker segment: Define Huffman table(s) (0xFFC4)
   > Marker segment: Define Huffman table(s) (0xFFC4)
   > Start of Segment header: Start of Scan (0xFFDA)
     Entropy-coded segment (dissection is not yet implemented): fcbd44a902f34aa29eab9fad73367a56102669c16a455a76…
     Marker: End of Image (0xffd9)
```

Figure 22: Wire Shark Capture for JPGE Payload

7 User Manual

It’s a very simple user friendly environment. A user only can observe the photo frame version on the web page and see how its work. It show two data on the web Page. One is the data from motion sensor. IF there is a presence of person in the room then it shows “true”, otherwise it will show “false”. And the other data shows the name of the person who is on the room. Moreover, It will show the sliding of the photos including the pictures of the person if it can find on the device

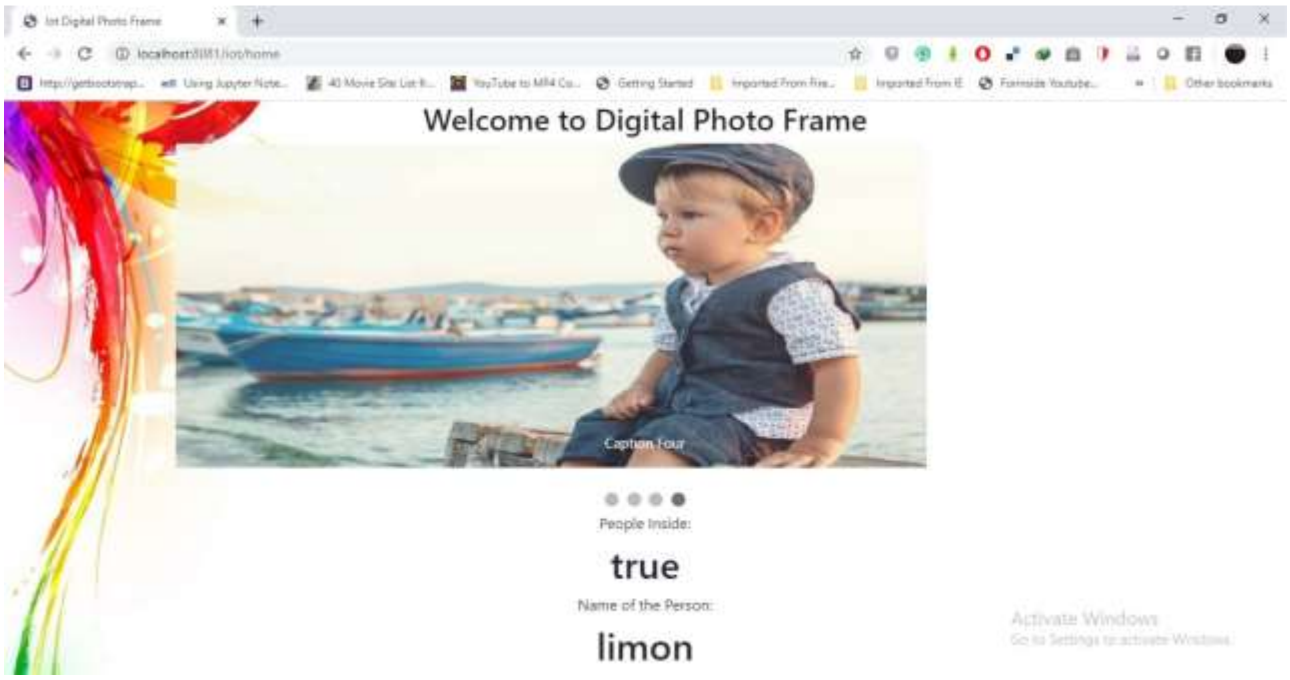


Figure 23: Digital Photo frame with Photos on Web browser

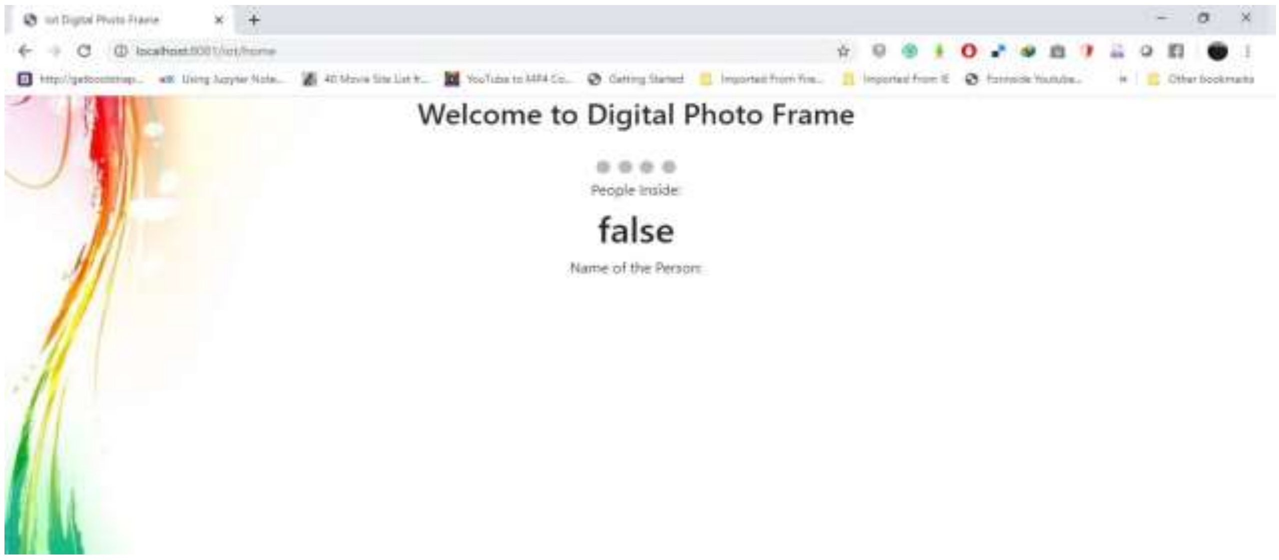


Figure 24: Digital Photo Frame without Photos on Web browser

8 CONCLUSION

We developed an IoT enabled smart home “Digital Photo Frame” in this paper. The developed prototype was named as ‘Smart Photo Frame,’ and after developing, its performance was evaluated to verify its working capability. Future work will be on the large-scale implementation of ‘Smart Photo Frame,’ considering the realistic possible cases. Also thinking of the possibilities and efficient operation of collecting Pictures and displaying service as per the study area size and complexity. As per the recent article, it was suggested that, the use of smart facilities in relevant sector of Smart Home will help in the progress. The developed smart photo frame facility can be used in the smart home for having proper management of pictures. [2]

9 REFERENCE

[1] Vishal Sharma, Hrishikesh Thakur, Pankaj Gaud, Hrishikesh Yadav, Prof Mahavir Devmane, "IoT Enabled Smart-Home" 2017 IJSRCSEIT , Volume 2 , Issue 2

[2] Sirisha Yerraboina, Nallapaneni Manoj Kumar, K. S. Parimala, N. Aruna Jyothi, "MONITORING THE SMART GARBAGE BIN FILLING STATUS: AN IOT APPLICATION TOWARDS WASTE MANAGEMENT" , Volume 9, Issue 6, June 2018, pp. 373–381, Article ID: IJCIET_09_06_043.

[3] Jeong Kim, John Zimmerman, "Smart Digital Photo Frames,"

[4] https://en.wikipedia.org/wiki/Internet_of_things.

[5] https://en.wikipedia.org/wiki/Constrained_Application_Protocol.

[6] https://en.wikipedia.org/wiki/Motion_detector

[7] https://en.wikipedia.org/wiki/Facial_recognition_system.