**ROVER**

**What is ThingView?**

ThingView is an application that allows users to visualize IoT (Internet of Things) data stored in **ThingSpeak**, a cloud-based IoT analytics platform. It is widely used for monitoring sensor data remotely.

**Working Procedure**

**Data Collection** – IoT devices like sensors collect real-time data (e.g., temperature, humidity, pressure).

**Data Transmission** – The collected data is sent to **ThingSpeak**, which stores and processes the data.

**Data Retrieval** – ThingView retrieves this stored data from ThingSpeak via API requests.

**Data Visualization** – ThingView presents the data in graphs, charts, and tables for easy monitoring.

**Features of ThingView**

✔ **Real-time Data Monitoring** – View real-time data from IoT devices.  
✔ **Historical Data Analysis** – Access past data for trend analysis.  
✔ **Graphical Representation** – Display data in graphs for better insights.  
✔ **Multiple Channel Support** – Supports multiple ThingSpeak channels.  
✔ **User-Friendly Interface** – Simple and easy to use.

**Uses of ThingView**

✅ **IoT Monitoring** – Useful for smart homes, agriculture, healthcare, and industrial applications.  
✅ **Remote Data Visualization** – Engineers and developers can analyze data without physical access.  
✅ **Research & Development** – Helps researchers analyze sensor data.  
✅ **Smart Farming** – Monitor environmental conditions like soil moisture and temperature.  
✅ **Home Automation** – Track energy consumption and control smart devices.

**How ThingView is Built & Programming Languages Used**

ThingView is a mobile application that interacts with **ThingSpeak** to visualize IoT data. The development involves both frontend and backend technologies.

**Languages & Technologies Used in ThingView**

**Frontend (Mobile App Development)**

**Java/Kotlin (Android)** – If it's a native Android app.

**Swift (iOS)** – If it's a native iOS app.

**Flutter (Dart) / React Native (JavaScript/TypeScript)** – If it's a cross-platform app.

**Backend & API Communication**

**ThingSpeak API (REST API & MQTT Protocol)** – Retrieves sensor data.

**Python/Node.js** – If additional server-side processing is needed.

**Database & Cloud**

**ThingSpeak Cloud** – Stores and processes IoT data.

**Firebase / AWS IoT** – For extended cloud storage & real-time updates.

**Graph & Data Visualization**

**Chart.js / D3.js** – If a web-based version exists.

**MPAndroidChart / iOS Charts** – For mobile-based graphs.

**How It Works Internally?**

**IoT Device Sends Data →** Sensor data is sent to ThingSpeak via Wi-Fi or MQTT.

**ThingSpeak Stores Data →** Data is processed and stored in cloud channels.

**ThingView Fetches Data →** The app queries ThingSpeak using API requests.

**Visualizing Data →** The app converts raw data into graphs, charts, and tables.

**Can You Build a Similar App?**

Yes! You can build a similar IoT data visualization app using:

**Flutter (Dart)** for cross-platform mobile development.

**Node.js/Python** for backend API processing.

**ThingSpeak API** for IoT data retrieval.

**Chart.js / MPAndroidChart** for data visualization.

**Rover Connection Procedure for Satellites with Cloud Services: Process & Uses**

Autonomous rovers, such as those used in space exploration (e.g., NASA’s Perseverance) and terrestrial applications (e.g., remote monitoring and agricultural rovers), rely on satellite communication and cloud services for real-time data transmission and remote control.

**1. Connection Procedure for Rovers with Satellites & Cloud Services**

**Step 1: Data Acquisition from Rover Sensors**

Rovers are equipped with various sensors, cameras, and instruments to collect data.

Common sensors include LiDAR, GPS, IMU (Inertial Measurement Unit), temperature sensors, and robotic arms.

**Step 2: Data Processing & Pre-Transmission**

The onboard computer of the rover processes sensor data.

Critical data is prioritized to reduce bandwidth usage.

**Step 3: Communication with Satellite Network**

The rover sends data using **radio frequency (RF) signals** via a high-gain antenna.

Satellite types used:

**LEO (Low Earth Orbit) Satellites** (e.g., Starlink, Iridium) for low-latency communication.

**GEO (Geostationary Orbit) Satellites** for stable connections over a fixed region.

**Step 4: Data Relay to Ground Stations**

Satellites receive the data and forward it to **Earth-based ground stations** via RF communication.

Ground stations decode the transmitted signals and prepare them for cloud storage.

**Step 5: Cloud Storage & Processing**

The received data is uploaded to cloud services (AWS, Google Cloud, Azure).

AI and machine learning algorithms analyze the data for insights and decision-making.

**Step 6: Remote Access & Control**

Scientists, engineers, or operators can access the rover’s data from anywhere via web dashboards or APIs.

Commands can be sent to the rover via the cloud, relayed through satellites back to the rover.

**2. Uses of Rover-Satellite-Cloud Connectivity**

**1. Space Exploration & Planetary Rovers**

✅ **Example:** NASA’s Perseverance & Curiosity on Mars

Data from Mars is sent via the **Deep Space Network (DSN)** to Earth.

Scientists analyze images and environmental data using cloud platforms.

**2. Military & Defense**

✅ **Example:** Surveillance rovers in war zones

Enables real-time battlefield intelligence.

AI-powered object recognition in cloud services enhances threat detection.

**3. Agriculture & Environmental Monitoring**

✅ **Example:** Smart farming with autonomous tractors

Cloud AI processes satellite images and rover data for precision farming.

Helps monitor soil moisture, crop health, and climate conditions.

**4. Autonomous Vehicles & Delivery Drones**

✅ **Example:** Starship delivery rovers, Tesla’s AI-powered vehicles

Uses satellite connectivity for navigation in remote areas.

Cloud-based AI assists in route optimization and traffic prediction.

**5. Disaster Response & Remote Exploration**

✅ **Example:** Rovers deployed in disaster-hit regions

Can be used for search-and-rescue missions in earthquake or flood-affected areas.

Satellite-based communication ensures real-time coordination with rescue teams.

**3. Technologies & Protocols Used**

✔ **Satellite Communication Protocols:** TCP/IP over satellite, Delay-Tolerant Networking (DTN)  
✔ **Cloud Platforms:** AWS IoT, Google Cloud IoT, Azure IoT Hub  
✔ **Edge AI & Processing:** NVIDIA Jetson, Intel Movidius for onboard AI analysis  
✔ **Navigation & Positioning:** GPS, RTK (Real-Time Kinematics) for precise positioning

**All-in-One Robot App: Working Procedure, Features, Programming Language, and Integration with Robots and Rovers**

The **All-in-One Robot App** is a versatile application designed to control and interact with multipurpose robots, enabling various operational modes. Developed by Dipin Adhikari, a student from Nepal, this app showcases a keen interest in robotics and artificial intelligence.

[instructables.com](https://www.instructables.com/All-in-one-robot-1/?utm_source=chatgpt.com)

**Features and Working Procedure:**

**Bluetooth Controlled Mode:**

**Functionality:** Allows users to control the robot's movements (forward, backward, left, right) using a mobile device via Bluetooth.

**Working:** Commands are sent from the app to the robot's microcontroller (e.g., Arduino Uno) through the HC-05 Bluetooth module. The robot also utilizes sensors like ultrasonic and infrared to detect obstacles and adjust movements accordingly.

[instructables.com](https://www.instructables.com/All-in-one-robot-1/?utm_source=chatgpt.com)

**Voice Controlled Mode:**

**Functionality:** Enables voice commands to direct the robot's actions.

**Working:** Utilizes Google's Voice Recognition API to interpret spoken commands, which are then transmitted to the robot via Bluetooth for execution.

[instructables.com](https://www.instructables.com/All-in-one-robot-1/?utm_source=chatgpt.com)

**Obstacle Avoiding Mode:**

**Functionality:** The robot autonomously navigates its environment, avoiding obstacles.

**Working:** Employs ultrasonic and infrared sensors to detect obstacles. Upon detection, the robot assesses alternative paths and maneuvers accordingly to prevent collisions.

[instructables.com](https://www.instructables.com/All-in-one-robot-1/?utm_source=chatgpt.com)

**Line Following Mode:**

**Functionality:** Enables the robot to follow a predefined path or line.

**Working:** Utilizes infrared sensors to detect and follow lines, allowing the robot to navigate along specific routes.

[instructables.com](https://www.instructables.com/All-in-one-robot-1/?utm_source=chatgpt.com)

**Programming Language Used:**

The All-in-One Robot App is developed using **Java**, the primary language for Android application development. The app's user-friendly interface facilitates seamless interaction with the robot across various modes.

[instructables.com](https://www.instructables.com/All-in-one-robot-1/?utm_source=chatgpt.com)

**Integration with Robots and Rovers:**

The app communicates with the robot's microcontroller (e.g., Arduino Uno) via the HC-05 Bluetooth module. Depending on the selected mode, the app sends specific commands that the microcontroller interprets to control the robot's motors and sensors. For instance, in Obstacle Avoiding Mode, the robot autonomously navigates using sensor data, while in Bluetooth Controlled Mode, it responds directly to user inputs from the app.

[instructables.com](https://www.instructables.com/All-in-one-robot-1/?utm_source=chatgpt.com)

**Additional Resources:**

**GitHub Repository:** For source code and further technical details, visit the [All-in-One Robot GitHub repository](https://github.com/Dipin-Adhikari/All-in-one-robot).

**Instructables Guide:** A comprehensive guide on building and programming the All-in-One Robot is available on [Instructables](https://www.instructables.com/All-in-one-robot-1/).

**Zeb Home Camera: Operating Purpose and Working Procedure**

The **Zeb Home Camera**, specifically models like the **Zeb-Smart Cam 100** and **Zeb-Smart Cam 101**, are designed to enhance home and office security through real-time monitoring and remote accessibility.

**Operating Purpose:**

**Remote Surveillance:** Monitor your premises in real-time from anywhere using the Zeb-Home App, ensuring the safety of your property and loved ones.

**Two-Way Communication:** Engage in conversations through the camera, making it suitable for checking in on family members or communicating with visitors.

**Motion Detection:** Receive instant alerts on your smartphone when movement is detected, allowing for prompt responses to potential security breaches.

**Working Procedure:**

**Setup:**

**Power Connection:** Connect the camera to a power source using the provided DC adapter (5V/1A).

**App Installation:** Download and install the **Zeb-Home App** from the Play Store (Android) or App Store (iOS).

**Camera Pairing:** Open the app, scan the QR code on the camera, and follow on-screen instructions to connect the camera to your Wi-Fi network.

**Features and Operation:**

**Live Streaming:** Access live video feeds through the app, providing real-time surveillance.

**Two-Way Audio:** Utilize the built-in microphone and speaker for seamless communication via the app.

**Motion Detection:** The camera detects movement up to 5 meters and sends notifications to your device.

**Night Vision:** Equipped with infrared LEDs, the camera captures clear images in low-light conditions, ensuring 24/7 monitoring.

**Storage Options:** Supports Micro SD cards up to 128GB for local storage and offers cloud storage for backups.

**Remote Access:**

With an active internet connection, remotely access and control the camera via the Zeb-Home App, allowing you to monitor your space from anywhere.

**Additional Features:**

**Compact Design:** The sleek and foldable design allows for easy placement on tabletops or wall mounting, blending seamlessly into various interiors.

**Wide Compatibility:** Compatible with both Android and iOS devices, ensuring a broad user base can utilize the camera's functionalities.

CODE

#include <LiquidCrystal.h>

#include <Servo.h>

const int rs = 8, en = 9, d4 = 10, d5 = 11, d6 = 12, d7 = 13;

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

#include "DHT.h"

#define DHTPIN 3

#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

int buz=2;

int m1=6;

int m2=7 ;

int m3=4;

int m4=5;

int m5=A0;

int m6=A1;

int m7=A2;

int m8=A3;

int mos=A4;

int ms=A5;

int x;

int cnt=0;

void setup() {

pinMode(m1,OUTPUT);

pinMode(m2,OUTPUT);

pinMode(m3,OUTPUT);

pinMode(m4,OUTPUT);

pinMode(m5,OUTPUT);

pinMode(m6,OUTPUT);

pinMode(m7,OUTPUT);

pinMode(m8,OUTPUT);

pinMode(mos,INPUT);

pinMode(ms,INPUT);

pinMode(buz,OUTPUT);

Serial.begin(9600);

digitalWrite(m1,0);

digitalWrite(m2,0);

digitalWrite(m3,0);

digitalWrite(m4,0);

digitalWrite(m5,0);

digitalWrite(m6,0);

digitalWrite(m7,0);

digitalWrite(m8,0);

  lcd.begin(16, 2);

  lcd.print("smart rover");

  delay(1500);

  lcd.clear();

  dht.begin();

}

void loop() {

     int t = dht.readTemperature();

    int h = dht.readHumidity();

    int mval=1023-analogRead(mos);

    int mtval=1023-analogRead(ms);

    if(mtval>500)

   {

    digitalWrite(buz,1);

   }

   else

   {

    digitalWrite(buz,0);

   }

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("T: "+String(t)+" H:"+String(h));

      lcd.setCursor(0, 1);

    lcd.print("MOS: "+String(mval)+" MT:"+String(mtval));

    delay(500);

    cnt++;

    if (cnt > 10) {

      Serial.print("2838060,P6Z07T35NVBAGTEV,0,0,project1,12345678,"

        + String(t) + "," + String(h) + ","

        + String(mval) + "," + String(mtval) + "\n");

      cnt = 0;

    }

 x=Serial.read();

 if(x==1)

{

 digitalWrite(m1,1);

 digitalWrite(m2,0);

}

  if(x==2)

{

 digitalWrite(m1,0);

 digitalWrite(m2,1);

}

  if(x==3)

{

 digitalWrite(m3,1);

 digitalWrite(m4,0);

}

 if(x==4)

{

 digitalWrite(m3,0);

 digitalWrite(m4,1);

}

 if(x==5)

{

 digitalWrite(m5,0);

 digitalWrite(m6,0);

 digitalWrite(m7,0);

 digitalWrite(m8,0);

  digitalWrite(m1,0);

 digitalWrite(m2,0);

 digitalWrite(m3,0);

 digitalWrite(m4,0);

} if(x==6)

{

 digitalWrite(m5,1);

 digitalWrite(m6,0);

 digitalWrite(m7,1);

 digitalWrite(m8,0);

}

 if(x==7)

{

 digitalWrite(m5,0);

 digitalWrite(m6,1);

 digitalWrite(m7,0);

 digitalWrite(m8,1);

} if(x==8)

{

 digitalWrite(m5,0);

 digitalWrite(m6,1);

 digitalWrite(m7,1);

 digitalWrite(m8,0);

} if(x==9)

{

 digitalWrite(m5,1);

 digitalWrite(m6,0);

 digitalWrite(m7,0);

 digitalWrite(m8,1);

}

}

void setup() {

  // put your setup code here, to run once:

}

void loop() {

  // put your main code here, to run repeatedly:

}