A PROJECT BY *github.com/nishantssoni*

INTRODUCTION

The IoT-based Automated Medicine Dispensing System is designed to revolutionize the way medication is administered, ensuring accuracy and efficiency in patient care. This system integrates modern technology to facilitate precise delivery of medication, significantly reducing the potential for human error. At its core, the system involves a mobile application through which doctors or caregivers input the prescribed dosage of medicine in millilitres. This data is transmitted to the cloud via Firebase, where it is accessed by the Node MCU microcontroller.

Upon receiving the data, the Node MCU microcontroller activates a diaphragm pump, dispensing the exact amount of medicine required. Each dispensation event is meticulously logged, recording the date, time, and quantity of medication administered into a Google Sheet in Excel format. This historical data can be accessed through the mobile application, allowing for continuous monitoring and review.

By ensuring timely and accurate administration of medicine, this system enhances patient care, particularly in environments where continuous monitoring is crucial. The IoT-based Automated Medicine Dispensing System not only streamlines the medication process but also provides a reliable method to track and review dispensation records, thereby supporting healthcare providers in delivering optimal care.

VISION

The vision is to revolutionize healthcare delivery through the integration of advanced IoT technology, leading to a future where medication management is precise and efficient. The goal is to ensure that every patient receives their medication with absolute accuracy, eliminating human error and inconsistencies. By leveraging IoT and automation, the vision extends to improving patient outcomes and reducing the administrative burden on healthcare providers. This vision supports a healthcare environment where technology amplifies human expertise, setting new standards for medication administration and enhancing overall care quality.

MISSION

The mission is to develop an advanced automated medicine dispensing system that utilizes IoT and cloud technologies to transform medication management. The focus is on minimizing human errors by automating the dosage process, ensuring patients receive the correct medication at the right time. The system will provide real-time monitoring and log each dispensation event, including dosage, date, and time, into a Google Sheet. This comprehensive record-keeping supports healthcare providers in tracking patient adherence and making informed decisions. By simplifying the medication management process and reducing manual intervention, the mission is to enhance overall patient care and operational efficiency in healthcare settings.

OBJECTIVE

The primary objective of the IoT-based Automated Medicine Dispensing System is to improve the accuracy and efficiency of medication administration. By integrating a mobile application, cloud technology, and a microcontroller, the system aims to eliminate human errors in dosage calculation and delivery. The mobile app inputs the prescribed medication amount, which is sent to the cloud and processed by the Node MCU microcontroller. This triggers a diaphragm pump to dispense the precise dosage. Each dispensation event, including date, time, and amount, is logged into a Google Sheet, allowing for easy tracking and monitoring. The objective is to streamline the medication management process, reduce provider workload, and improve patient care through accurate and timely medication delivery.

SWOT ANALYSIS

Strengths

1. **Accuracy in Medication Dispensing:** The IoT-based Automated Medicine Dispensing System offers a high level of precision in medication administration. By automating the dosage process, it effectively eliminates the risk of human error associated with manual calculation and dispensing. This precision ensures that patients receive the exact amount of medication prescribed, which is crucial for effective treatment and minimizing adverse effects. Accurate dispensing not only enhances patient safety but also improves the reliability of treatment outcomes, making the system a valuable tool in healthcare settings.
2. **Real-Time Monitoring and Logging:** The system's integration with cloud technology allows for real-time monitoring and logging of medication dispensation events. Each instance of medication delivery is recorded with details such as dosage, date, and time, which are stored in a Google Sheet. This comprehensive record-keeping enables healthcare providers to track medication adherence, review historical data, and make informed decisions based on accurate records.
3. **Ease of Use:** Designed with user experience in mind, the system features a mobile application that simplifies the process of inputting prescribed dosages and managing medication dispensing. Healthcare providers and caregivers can easily enter the required dosage into the app, which then communicates with the cloud and microcontroller to automate the dispensing process.
4. **Automated System:** By automating the medication dispensing process, the system reduces the manual workload on healthcare providers. This automation not only speeds up the dispensing process but also allows healthcare professionals to focus on other critical aspects of patient care.

Weaknesses

1. **Dependence on Technology:** The system's reliance on electronic components and internet connectivity means that any technical issues could disrupt its operation. For example, internet outages or hardware malfunctions can impact the system's ability to transmit data and dispense medication accurately. This dependence on technology requires robust technical support and backup plans to ensure continuous and reliable operation.
2. **Initial Costs:** Implementing the IoT-based Automated Medicine Dispensing System involves significant upfront costs. These costs include purchasing hardware components such as the Node MCU microcontroller and diaphragm pump, developing and maintaining the software, and setting up cloud infrastructure. While the long-term benefits may outweigh these initial expenses, the financial investment required for deployment can be a barrier, especially for smaller healthcare facilities.
3. **Technical Complexity:** The system's technical complexity may present challenges for healthcare providers and caregivers who are not familiar with advanced technology. Training is necessary to ensure that users can operate the system effectively and troubleshoot any issues that may arise. This learning curve may lead to initial resistance or reluctance to adopt the new technology, potentially impacting the system's successful implementation and utilization.
4. **Data Security Concerns:** Storing patient data and medication records in the cloud introduces potential data security and privacy risks. Ensuring that the system adheres to stringent cybersecurity protocols is essential to protect sensitive patient information from unauthorized access or breaches. Addressing these security concerns requires ongoing efforts to implement and update security measures, which can add to the system's overall complexity and cost.

Opportunities

1. **Expanding to New Markets:** The system has significant potential for expansion into various healthcare markets, including hospitals, outpatient clinics, long-term care facilities, and home care environments. By adapting the system to different

settings and patient needs, it can reach a broader audience and address diverse medication management challenges.

2. **Integration with Other Healthcare Systems:** There is an opportunity to integrate the automated dispensing system with other healthcare management systems and electronic health records (EHRs). This integration can enhance the system's functionality by providing a more comprehensive view of patient health data, facilitating better coordination between different aspects of patient care, and improving overall healthcare delivery.
3. **Advancements in Technology:** As technology continues to evolve, there are opportunities to further enhance the system's capabilities. Innovations in IoT devices, cloud computing, and cybersecurity can lead to improvements in system performance, reliability, and security. Staying at the forefront of technological advancements will allow the system to offer new features and maintain its competitive edge in the market.
4. **Increased Focus on Patient Safety:** With a growing emphasis on patient safety and the need for accurate medication management, the system is well-positioned to address these concerns. The increasing demand for solutions that enhance patient safety and reduce medication errors creates a favorable environment for the adoption of automated dispensing systems.

Threats

1. **Regulatory Challenges:** The automated medicine dispensing system must navigate complex healthcare regulations and standards, which can vary by region and country. Ensuring compliance with these regulations is essential for legal operation and acceptance in the market. Obtaining necessary approvals and certifications can be time-consuming and may pose a challenge to the system's deployment and scalability.
2. **Competition:** The healthcare technology market is competitive, with existing and emerging solutions offering similar or advanced functionalities. The presence of competitors may impact the system's market share and require continuous innovation to stay ahead. Keeping up with market trends and differentiating the

system through unique features or improved performance is crucial for maintaining a competitive advantage.

3. **Technological Failures:** Risks associated with technological failures, such as software bugs or hardware malfunctions, could affect the system's performance and reliability. These failures could lead to disruptions in medication dispensing, impacting patient care. Implementing robust quality assurance processes and having contingency plans in place are important to mitigate the impact of potential technological issues.
4. **User Resistance:** Resistance to adopting new technology from healthcare providers and caregivers may hinder the system's successful implementation. Concerns about reliability, changes in workflow, or unfamiliarity with the technology can contribute to reluctance. Addressing these concerns through effective training, support, and demonstrating the system's benefits is essential to achieving widespread acceptance and utilization.

CHRONOLOGY OF ACHIEVEMENTS

1. **Conceptualization and Planning (4 week):**
   - **Project Ideation:** Identified the need for an automated medicine dispensing system to enhance medication accuracy and efficiency.
   - **Feasibility Study:** Conducted research on existing solutions, market needs, and technological requirements. Established project goals and objectives.
   - **Initial Design:** Developed the preliminary system design, including the mobile application interface, cloud integration, and hardware components.
2. **Development Phase (3 week):**
   - **Hardware Development:** Acquired and assembled necessary hardware components, including the Node MCU microcontroller and diaphragm pump.
   - **Software Development:** Created the mobile application for dosage input and developed the cloud-based backend using Firebase.
3. **Integration (1 week)**
   - **Integration:** Integrated the mobile application with the Node MCU microcontroller and set up communication with Firebase for data transmission and storage.
4. **Prototype Testing (2 week):**
   - **Initial Testing:** Conducted internal testing of the prototype to identify and address any hardware and software issues. Tested the accuracy of dosage dispensing and data logging.
   - **Refinement:** Made necessary adjustments based on test results, including optimizing the user interface and improving system reliability.
5. **Final Adjustments (2 week):**

- **System Refinement:** Incorporated feedback to make final adjustments to the hardware and software.
- **Regulatory Compliance:** Ensured the system met all necessary regulatory standards.

6. **Full Deployment (1 week):**
    - **Launch:** Rolled out the fully developed system.

SUMMARY

The IoT-based Automated Medicine Dispensing System demonstrates significant advancements in medication management, integrating hardware and software technologies to improve accuracy and efficiency. Key results include precise dosage delivery facilitated by the Node MCU microcontroller and diaphragm pump, real-time monitoring and logging of dispensation events via Firebase, and an intuitive mobile application for user-friendly dosage input. The system ensures reliable medication administration and provides detailed historical records for tracking and review. Successful deployment and positive feedback from initial users validate the system's effectiveness in enhancing patient care and reducing manual errors in medication delivery.

INTERPRETATIONS

The results indicate that the Automated Medicine Dispensing System effectively addresses common issues in medication management, such as dosage inaccuracies and manual errors. The integration of IoT technology allows for real-time data transmission and precise control over medication dispensing, which improves the reliability of medication administration. The user-friendly interface of the mobile application and the comprehensive logging of dispensation events contribute to streamlined operations and better oversight. The positive feedback and successful implementation suggest that the system meets the intended goals of increasing accuracy, reducing errors, and enhancing overall efficiency in healthcare settings.

IMPLICATIONS

The results have significant implications for healthcare practices and patient safety. By automating the medication dispensing process, the system reduces the risk of human error and ensures that patients receive the correct dosage of medication. This advancement not only enhances patient safety but also improves the operational efficiency of healthcare providers. The ability to track and review medication history in real-time supports better decision-making and adherence monitoring. The success of the system underscores the potential for similar IoT-based solutions to address other challenges in healthcare, paving the way for further innovations in medication management and patient care.

Limitations

Despite its advantages, the system has limitations that must be acknowledged. The reliance on technology means that any hardware malfunctions or connectivity issues could disrupt medication dispensing. The system’s effectiveness is also dependent on accurate data input and user familiarity with the technology, which may require additional training and support. Furthermore, while the system improves accuracy and efficiency, it does not address all aspects of medication management, such as patient-specific medication interactions or adherence outside of scheduled doses. These factors should be considered when evaluating the system's overall impact and applicability.

Recommendations

1. **Enhance Technical Support:** Provide robust technical support and regular maintenance to address any hardware or software issues promptly and minimize disruptions in service.
2. **Expand Training Programs:** Offer comprehensive training and ongoing education for healthcare providers and caregivers to ensure effective use of the system and integration into existing workflows.
3. **Address Data Security:** Continuously update cybersecurity measures to protect patient data and comply with regulatory requirements, ensuring that the system remains secure against potential breaches.
4. **Explore Integration Opportunities:** Investigate opportunities to integrate the system with other healthcare management tools and electronic health records (EHRs) to enhance overall care coordination and data sharing.
5. **Conduct Ongoing Evaluation:** Regularly assess the system's performance and gather user feedback to identify areas for improvement and adapt the system to meet evolving needs and technological advancements.

PLATFORM USED: HARDWARE

1. **Node MCU (ESP8266) Microcontroller:** The Node MCU is the system's core, managing data processing and communication between components. It receives dosage instructions from the mobile application and controls the diaphragm pump accordingly. It also interfaces with Firebase for cloud storage and real-time synchronization. The microcontroller ensures precise dispensing by interpreting data inputs and executing commands to regulate medicine flow, making it essential for the system's functionality.
2. **Diaphragm Pump:** This pump is crucial for delivering the exact dosage of medication as specified by the Node MCU. It uses a diaphragm mechanism to control the flow of liquid, ensuring accuracy in dispensation. The pump's performance directly impacts the system's reliability and precision, making it a vital component for accurate medication administration.
3. **Power Supply Unit:** The power supply unit provides the necessary electrical power to the Node MCU, diaphragm pump, and other system components. It ensures stable and reliable operation by converting and regulating the voltage required by the various components. A consistent power supply is essential to prevent interruptions and maintain the system's performance and accuracy.
4. **Tubing and Connectors:** Tubing and connectors are used to transport medicine from the diaphragm pump to the dispensing point. The tubing needs to be durable and secure to prevent leaks and ensure smooth delivery. Connectors play a critical role in maintaining a stable connection between different parts of the system, facilitating accurate medication dispensing.
5. **Enclosure:** The enclosure houses the Node MCU, diaphragm pump, and other sensitive components. It protects these parts from physical damage, dust, and moisture, ensuring their longevity and reliable operation. The enclosure also aids in organizing the components, keeping them secure and providing a clean, professional appearance for the system.

6. **Buzzer:** The buzzer provides auditory feedback to users, indicating various system statuses such as successful medication dispensing or errors. It enhances user awareness by alerting them to important events or issues. The buzzer's alerts are crucial for ensuring that users are promptly informed about the system's operational state and any necessary actions.
7. **OLED 1306 Display:** The OLED 1306 display (128 X 32) provides real-time visual feedback on the system's status. It shows information such as dispensing progress, system alerts, and error messages. This display helps users monitor the system's operation and make informed decisions based on the information provided, improving overall user experience.
8. **Push Button:** The push button allows users to manually control the system, such as initiating or halting the dispensing process. It provides an additional layer of interaction, enabling users to manage the system according to their needs. This manual control option is important for flexibility and addressing specific operational requirements.

PLATFORM USED: SOFTWARE

1. **Mobile Application (MIT App Inventor):** The mobile application, developed using MIT App Inventor, serves as the interface for inputting medication dosages and monitoring the dispensing process. It allows healthcare providers to easily enter dosage information and track system performance. The user-friendly design of the app ensures that it is accessible and efficient for managing medication delivery.
2. **Firebase:** Firebase is used for cloud-based data storage and real-time synchronization between the mobile application and the Node MCU. It handles the transmission of dosage instructions and stores dispensation records securely. Firebase ensures that data is consistently updated and accessible, facilitating accurate and timely communication between system components.
3. **Google Sheets API:** The Google Sheets API is integrated to log medication dispensation data into Google Sheets. This integration provides a structured record of each dispensation event, including dosage, date, and time. The logged data can be reviewed and analyzed to monitor medication administration, track patterns, and ensure compliance with dosing schedules.
4. **Arduino IDE:** The Arduino IDE is utilized to program the Node MCU and OLED display. It provides an environment for writing, debugging, and uploading firmware code to the microcontroller. The IDE supports the development of the system's functionality by enabling precise control over hardware operations and facilitating interaction with the OLED display.
5. **Embedded C/C++:** Embedded C/C++ are the programming languages used to develop the firmware for the Node MCU microcontroller. These languages offer the precision and efficiency needed to manage hardware functions and system operations. They enable the microcontroller to process data, control the diaphragm pump, and ensure accurate medication dispensing.

MODULES OF THE PROJECT

1. User Interface Module:

The User Interface (UI) Module consists of a mobile application designed for doctors and caregivers. This app allows users to input medication details, including dosage and schedule, and view a history of dispensed medications. The interface is user-friendly, facilitating easy navigation and accurate data entry. It also provides real-time updates on the dispensing status and allows users to track medication administration effectively. The UI is critical for ensuring that healthcare providers can manage medication schedules efficiently and monitor the system's performance with ease.

2. Cloud Communication Module:

The Cloud Communication Module utilizes Firebase to handle data storage and real-time synchronization. Firebase serves as the backbone for transmitting dosage instructions from the mobile application to the Node MCU microcontroller. It ensures that data is updated in real-time, allowing for prompt and accurate medication dispensing. Additionally, Firebase provides secure storage for dispensation records, which are crucial for maintaining an accurate medication history. This module ensures that all components of the system communicate effectively and that data integrity is maintained across different platforms.

3. Control Module:

The Control Module involves programming the Node MCU microcontroller to manage system operations. This module is responsible for listening to data transmitted from Firebase and controlling the diaphragm pump based on the received dosage instructions. The Node MCU processes the data, triggers the pump, and monitors its performance to ensure accurate dispensing. It also handles communication between the hardware components and the cloud-based storage, ensuring that the system functions smoothly and responds to user inputs in real-time.

4. Dispensing Mechanism Module:

The Dispensing Mechanism Module focuses on the setup and control of the diaphragm pump. This module ensures that the pump operates correctly to dispense the precise amount of medicine as directed by the Node MCU. It includes configuring the pump for accurate flow control and integrating it with the system's control module. The diaphragm pump’s performance is critical for achieving accurate dosages, making this module essential for the reliable delivery of medication. Proper setup and calibration are necessary to avoid errors and ensure the system’s effectiveness.

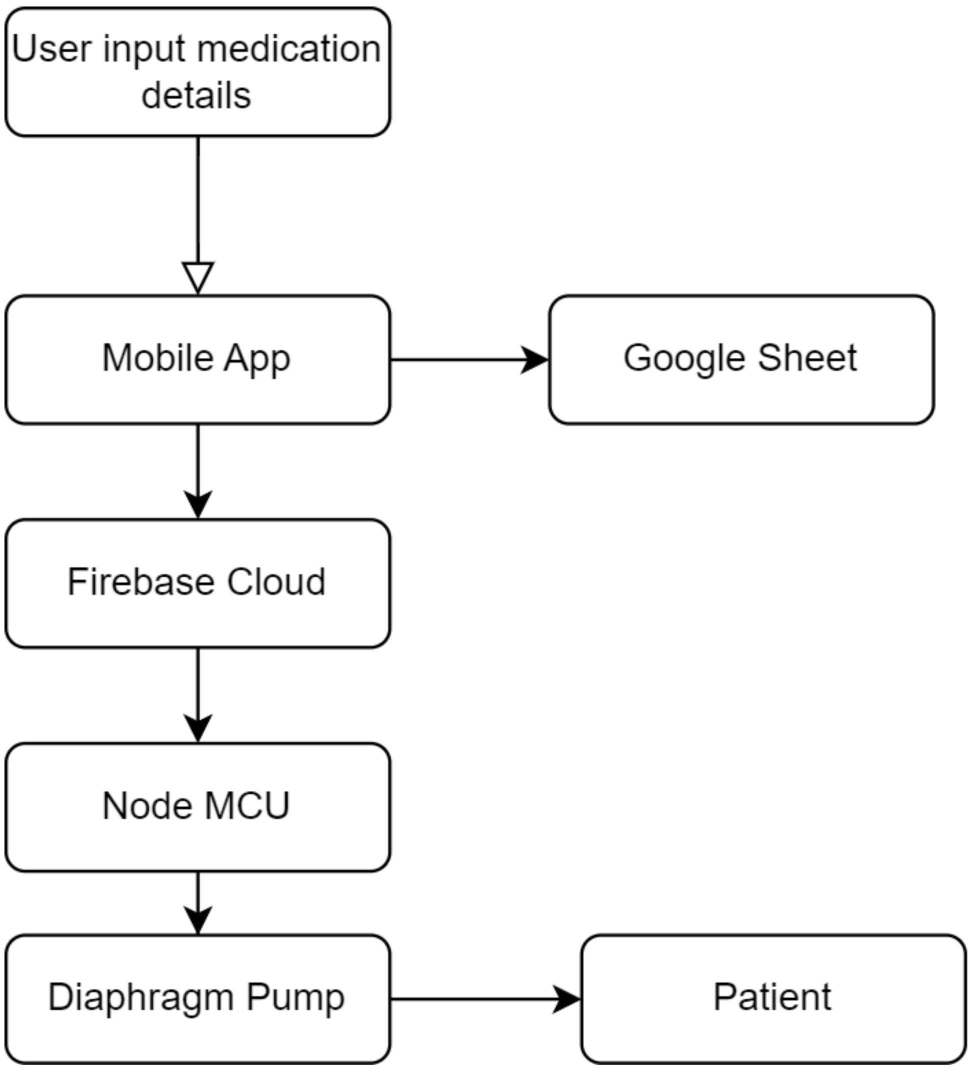
5. System Monitoring Module:

The System Monitoring Module provides feedback mechanisms to verify correct dosage delivery and overall system functionality. This includes monitoring the operational status of the diaphragm pump, ensuring that it dispenses the correct amount of medicine, and detecting any malfunctions or issues. Feedback is provided through alerts and the OLED display, allowing users to quickly identify and address any problems. This module is essential for maintaining the reliability of the dispensing process and ensuring that the system performs as intended.

6. Data Logging Module:

The Data Logging Module integrates with the Google Sheets API to record and store dispensation events. This module automatically logs details such as dosage, date, and time of each medication dispensed into a Google Sheet, creating a comprehensive historical record. The logged data is accessible for review and analysis, aiding in monitoring medication administration and ensuring compliance with prescribed schedules. This module provides valuable insights into system performance and helps track medication delivery over time.

DATA FLOW DIAGRAM



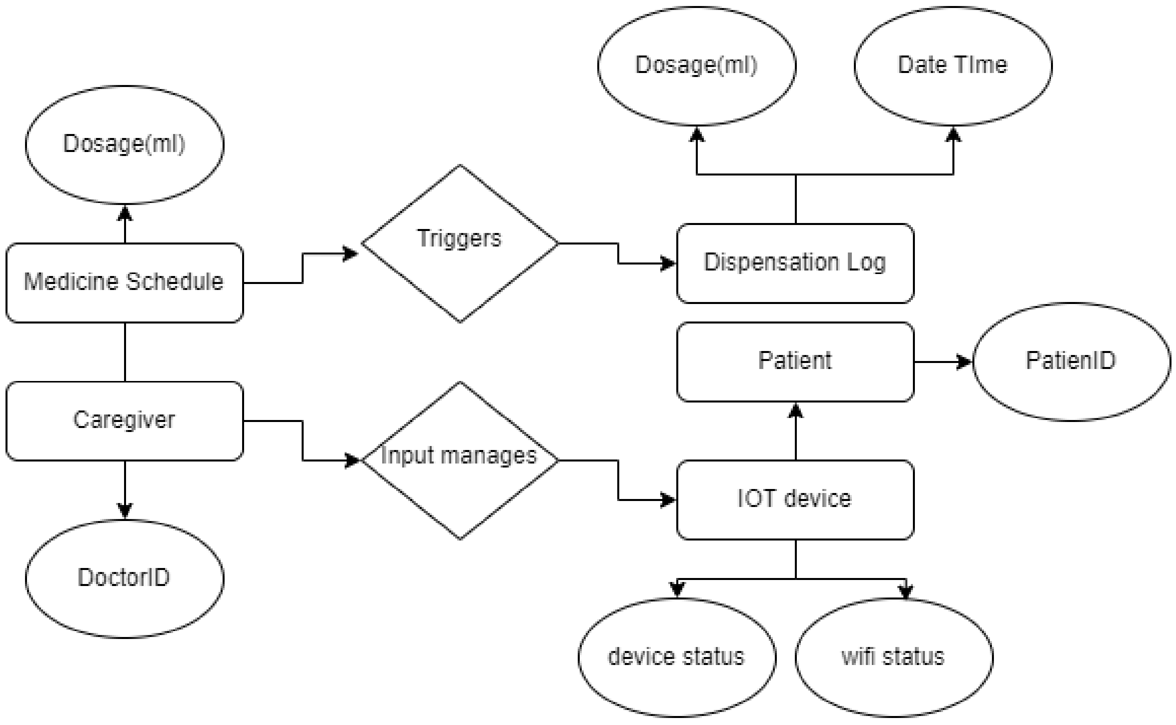
1. **User Input to Mobile App:**
   - **Action:** A doctor or caregiver inputs medication details, such as dosage and schedule, into the mobile application.
   - **Details:** The mobile app is designed with fields for entering medication information and scheduling details.
2. **Mobile App to Firebase and Google Sheets:**
   - **Action:** Once the medication details are entered, the app sends this data to Firebase for cloud storage and real-time synchronization. Simultaneously,

the data is sent to Google Sheets via the Google Sheets API for logging and historical record-keeping.

   - **Details:** Firebase handles real-time updates and synchronization, while Google Sheets provides a detailed log of dispensation events for review and compliance tracking.

3. **Firebase to Node MCU:**
   - **Action:** Firebase relays the medication details to the Node MCU microcontroller.
   - **Details:** The Node MCU continuously listens for data from Firebase. Upon receiving the medication instructions, it processes the information to control the dispensing mechanism.
4. **Node MCU to Diaphragm Pump:**
   - **Action:** Based on the received data, the Node MCU sends commands to the diaphragm pump to dispense the prescribed amount of medication.
   - **Details:** The microcontroller ensures that the pump delivers the exact dosage as instructed, controlling the flow and timing for accurate dispensing.
5. **Diaphragm Pump to Patient:**
   - **Action:** The diaphragm pump dispenses the medication to the patient.
   - **Details:** The medication is delivered through tubing from the pump to the patient, ensuring that the correct dosage reaches the intended recipient.
6. **Patient Receives Medication:**
   - **Action:** The patient receives the prescribed medication.
   - **Details:** The system ensures that the patient gets the precise amount of medication as prescribed, enhancing the accuracy and reliability of the treatment process.

Entity-Relationship Diagram



This image represents an Entity-Relationship (ER) diagram, which is used to model the relationships between different entities in a system. Here's a breakdown of the components and their relationships:

Entities and Attributes:

- Medicine Schedule:
    - Dosage (ml): Amount of medicine to be administered.
- Caregiver:
    - DoctorID: Identifier for the doctor responsible.
- Dispensation Log:
    - Dosage (ml): Amount dispensed.
    - Date Time: Timestamp of the dispensation.
- Patient:
    - PatientID: Unique identifier for the patient.

- IoT Device:
    - Device Status: Current status of the device.
    - WiFi Status: Connectivity status of the device.

Relationships:

- Triggers:
    - Links the **Medicine Schedule** (with its **Dosage (ml)** attribute) to the **Dispensation Log.**
- Input Manages:
    - Connects the **Caregiver** to the **Patient** via **PatientID**.
    - Links the **Caregiver** to the **IoT Device** which monitors **Device Status** and **WiFi Status**.

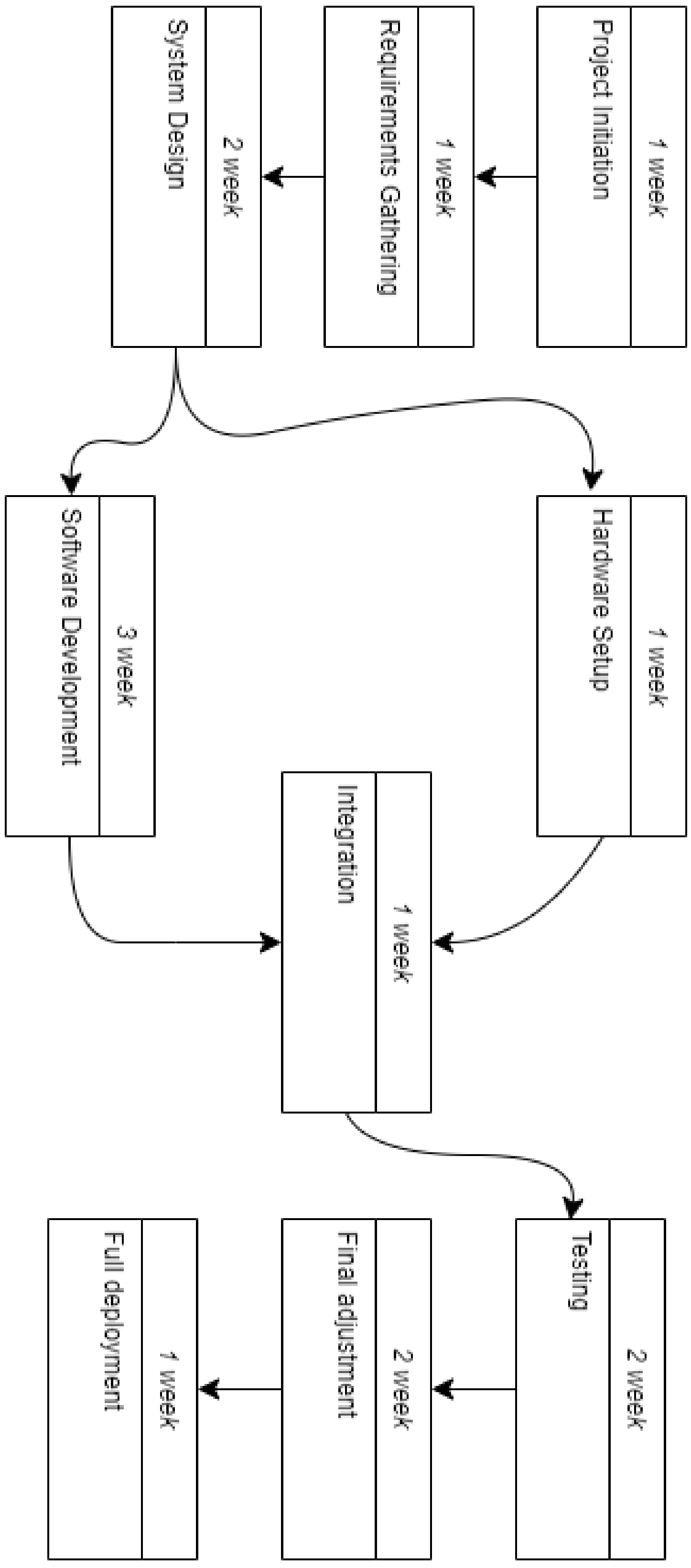
Explanation:

- The **Medicine Schedule** and **Caregiver** provide necessary information that triggers the **Dispensation Log**.
- The **Caregiver** is associated with the **DoctorID** and manages the **Patient** and the **IoT Device**.
- The **IoT Device** tracks the status and connectivity to ensure proper functioning and monitoring of the patient's medication schedule.
- The **Patient** has a unique **PatientID** which is managed by the **Caregiver**.

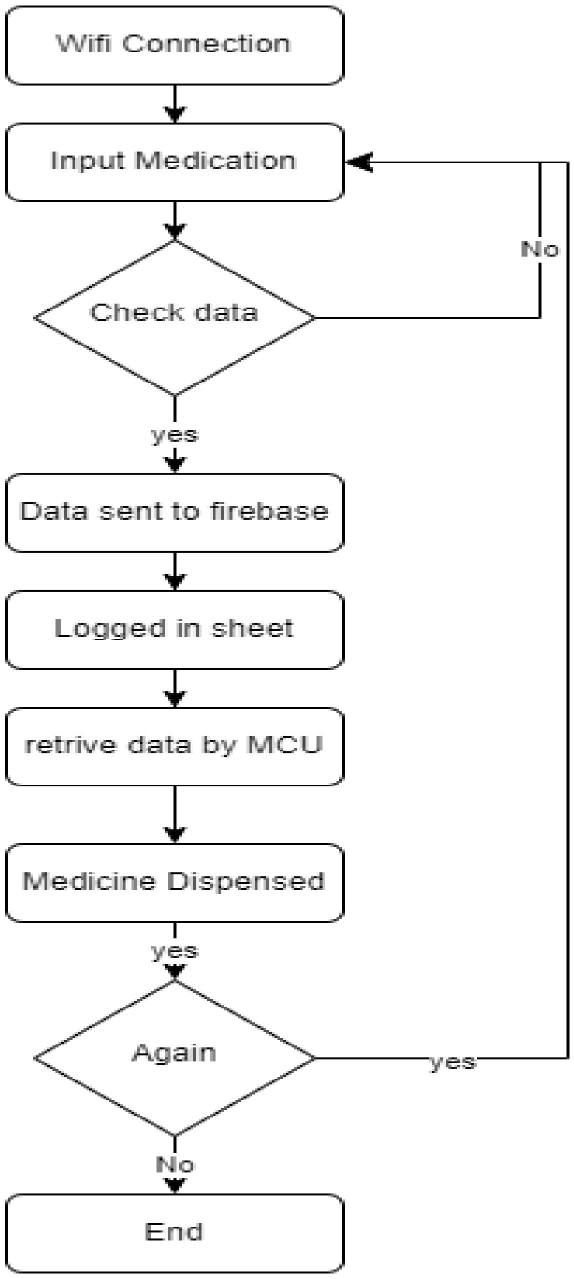
GANTT CHART

Task	**Duration**	**Start Date**	**End Date**	**Dependencies**
Project Initiation	1 week	01/04/2024	07/04/2024	-
Requirements Gathering	1 weeks	08/04/2024	14/04/2024	Project Initiation
System Design	2 weeks	15/04/2024	28/04/2024	Requirements Gathering
Hardware Setup	1 weeks	29/04/2024	04/05/2024	System Design
Software Development	3 weeks	05/05/2024	25/05/2024	System Design
Integration	1 weeks	26/05/2024	02/06/2024	Hardware Setup, Software Development
Testing	2 weeks	03/06/2024	16/06/2024	Integration
Final adjustment	2 weeks	17/06/2024	01/07/2024	Testing
Full deployment	1 week	02/07/2024	08/07/2024	Deployment

PERT CHART


1 week
Project Initiation
1 week
Requirements Gathering
2 week
System Design
1 week
Hardware Setup
3 week
Software Development
1 week
Integration
2 week
Testing
2 week
Final adjustment
1 week
Full deployment

ACTIVITY DIAGRAM



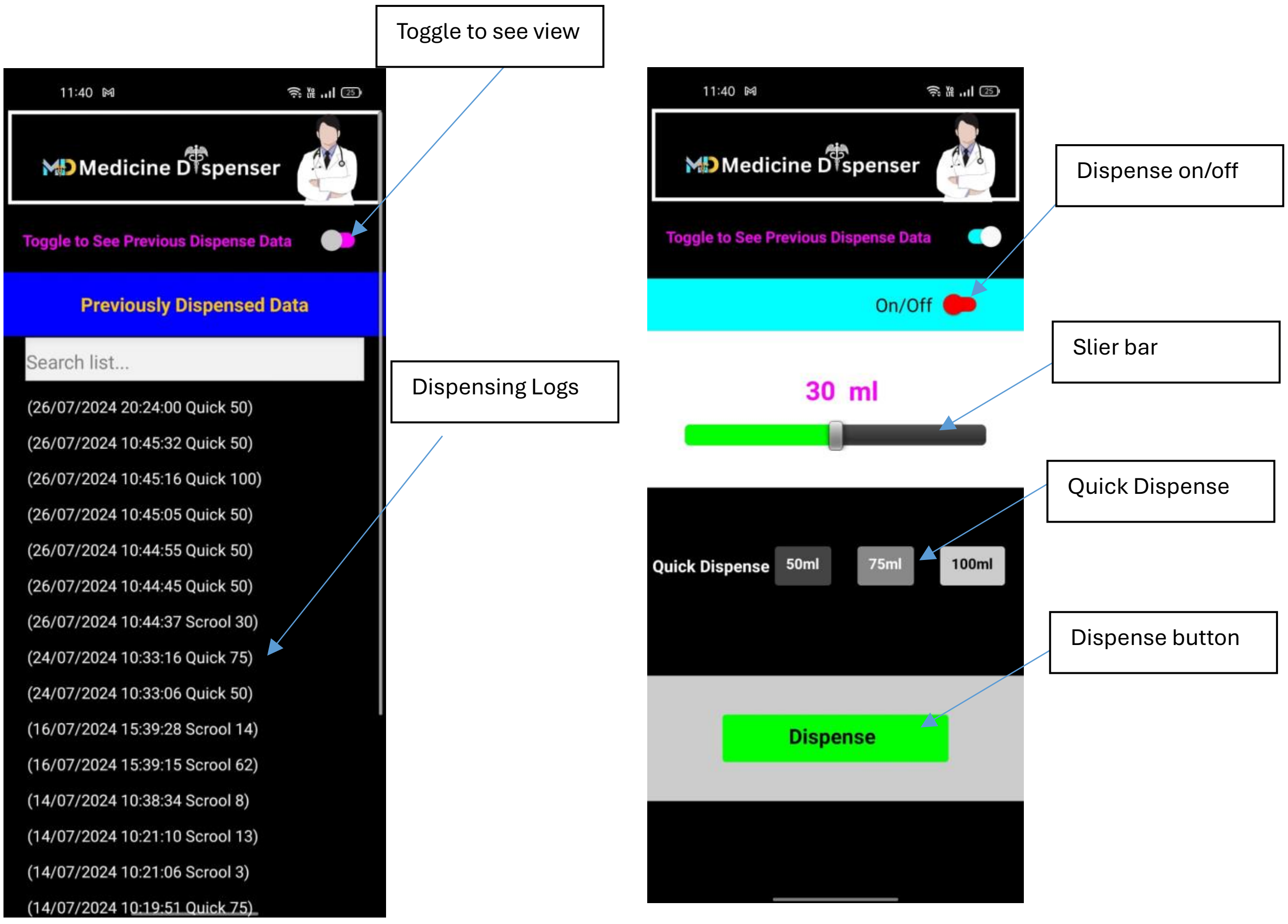
This activity diagram represents a process flow for dispensing medication with the aid of a WiFi connection and Firebase for data storage.

Here is a step-by-step explanation:

1. **WiFi Connection**: The process starts by establishing a WiFi connection.
2. **Input Medication**: The user inputs the medication data.
3. **Check Data**: The system checks the input data.
    a. If the data is incorrect, the process loops back to the "Input Medication" step.
    b. If the data is correct, the process moves to the next step.
4. **Data Sent to Firebase**: The validated data is sent to Firebase for storage.
5. **Logged in Sheet**: The data is logged in a sheet for record-keeping.
6. **Retrieve Data by MCU**: The Microcontroller Unit (MCU) retrieves the data from Firebase.
7. **Medicine Dispensed**: The medicine is dispensed based on the retrieved data.
8. **Again**: The system checks if the process should be repeated.
    a. If "Yes", the process loops back to the "Input Medication" step.
    b. If "No", the process ends.

CONTENTS OF THE PROJECT: APP

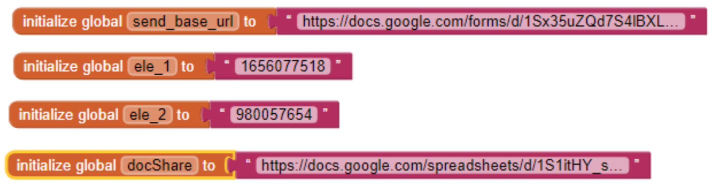
User Interface



```
(26/07/2024 20:24:00 Quick 50)
(26/07/2024 10:45:32 Quick 50)
(26/07/2024 10:45:16 Quick 100)
(26/07/2024 10:45:05 Quick 50)
(26/07/2024 10:44:55 Quick 50)
(26/07/2024 10:44:45 Quick 50)
(26/07/2024 10:44:37 Scrool 30)
(24/07/2024 10:33:16 Quick 75)
(24/07/2024 10:33:06 Quick 50)
(16/07/2024 15:39:28 Scrool 14)
(16/07/2024 15:39:15 Scrool 62)
(14/07/2024 10:38:34 Scrool 8)
(14/07/2024 10:21:10 Scrool 13)
(14/07/2024 10:21:06 Scrool 3)
(14/07/2024 10:19:51 Quick 75)
```

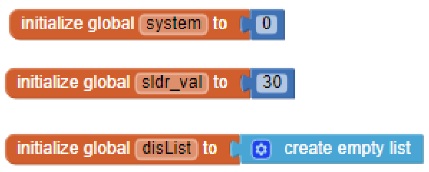
The about images is the Screen shot of app running in a mobile phone.

Code Explanation



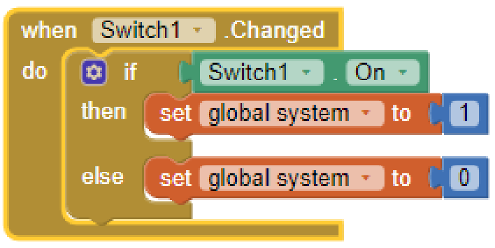
```
initialize global send_base_url to " https://docs.google.com/forms/d/1Sx35uZQd7S4lBXL... "
initialize global ele_1 to " 1656077518 "
initialize global ele_2 to " 980057654 "
initialize global docShare to " https://docs.google.com/spreadsheets/d/1S1itHY_s... "
```

- The **initial global variable** contains the base URL, the Google Sheet URL, and element IDs in `ele_1` and `ele_2` for the system.
- `send_base_url` and `docShare` contain the URLs for the Google Sheet API used for fetching and updating data.



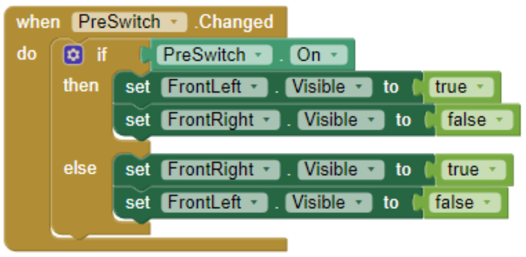
```
initialize global system to 0
initialize global sldr_val to 30
initialize global disList to create empty list
```

- The **system variable** contains the state of the toggle used to turn the dispense button on or off.
- **sldr_val** is the variable that stores the value of the slider bar, which is set to 30 by default.
- **disList** is a list that fetches and stores the previously dispensed data from the Google Sheet API.



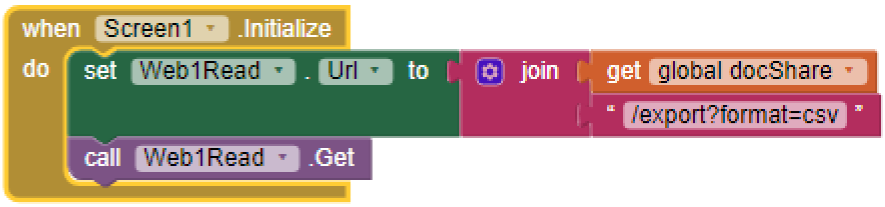
```
when Switch1 .Changed
do  if Switch1 . On
    then  set global system to 1
    else  set global system to 0
```

It is a block of code that uses a trigger. When **Switch1** changes, it checks its value and inverts the value accordingly.



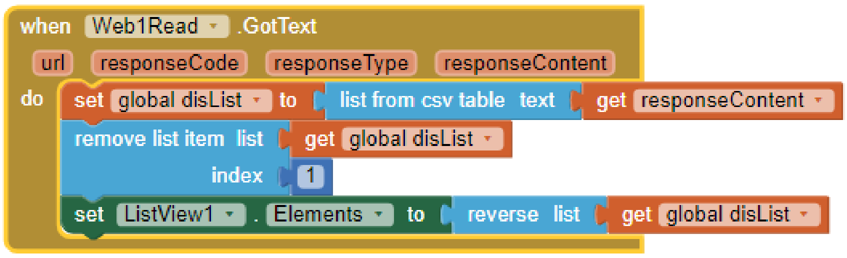
```
when PreSwitch .Changed
do  if  PreSwitch . On
    then  set FrontLeft . Visible to  true
          set FrontRight . Visible to  false
    else  set FrontRight . Visible to  true
          set FrontLeft . Visible to  false
```

- It is the top toggle button called **"PreSwitch"** logic, which changes the view between dispensing logs and the main app view



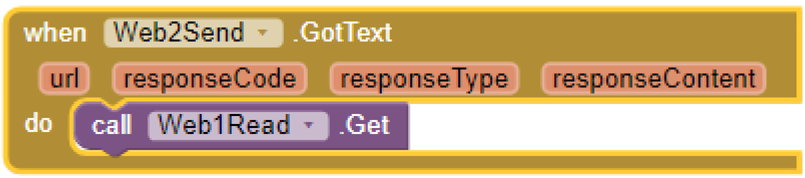
```
when Screen1 .Initialize
do  set Web1Read . Url to  join  get global docShare
                                 " /export?format=csv "
    call Web1Read .Get
```

- When **Screen1.Initialize**:
  - This is an event block that triggers when the screen named "Screen1" initializes (i.e., when the app starts and this screen is loaded).
- Set **Web1Read.Url** to join:
  - This block sets the URL for the Web1Read component. The join block concatenates strings and variables to form this URL.
- The join block consists of:
  - get global **docShare**: This retrieves the value stored in the global variable **docShare**. This variable contains a base URL or part of a URL.
  - "/export?format=csv": This is a string that specifies the path and query parameters to be appended to the base URL.
- Call **Web1Read.Get**:
  - After setting the URL, the Web1Read.Get method is called. This method sends a request to the URL that was just set.
  - In this case, it is sending a request to download or access data in CSV format from the specified URL.



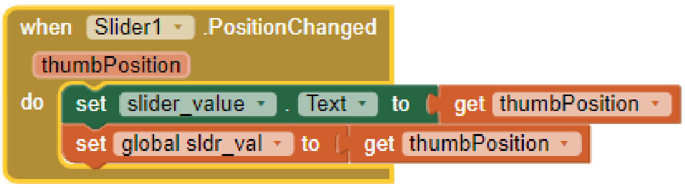
```
when Web1Read .GotText
  url  responseCode  responseType  responseContent
do  set global disList to  list from csv table  text  get responseContent
    remove list item  list  get global disList
                      index  1
    set ListView1 . Elements to  reverse list  get global disList
```

1. **Event Trigger (when Web1Read.GotText)**:
    - This block of code starts when the web component Web1Read receives a response from a web request, which includes url, responseCode, responseType, and responseContent.
2. **Set Global Variable (set global disList)**:
    - The responseContent is converted into a list from CSV table text and assigned to the global variable **disList**.
        - list from csv table text get responseContent: This block takes the CSV formatted responseContent and converts it into a list.
3. **Remove First List Item**:
    - The first item in the list (global disList) is removed.
        - remove list item list get global disList index 1: This block removes the item at index 1 (the first item) from global disList.
4. **Set ListView Elements**:
    - The elements of ListView1 are set to the reversed global disList.
        - set ListView1.Elements to reverse list get global disList: This block reverses global disList and sets it as the elements of ListView1 reverse so we can see the latest logs first.



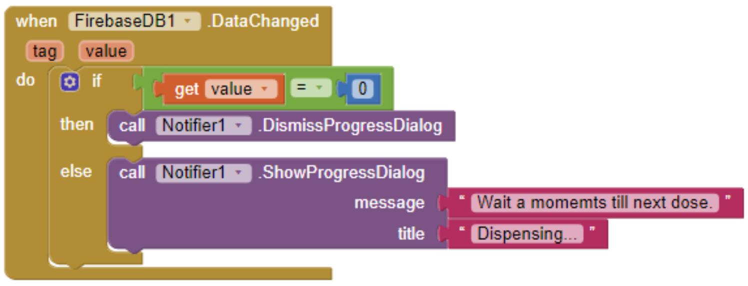
```
when Web2Send .GotText
  url  responseCode  responseType  responseContent
do  call Web1Read .Get
```

- When Web2Send receive request then it call Web1Read and pass the data to it.



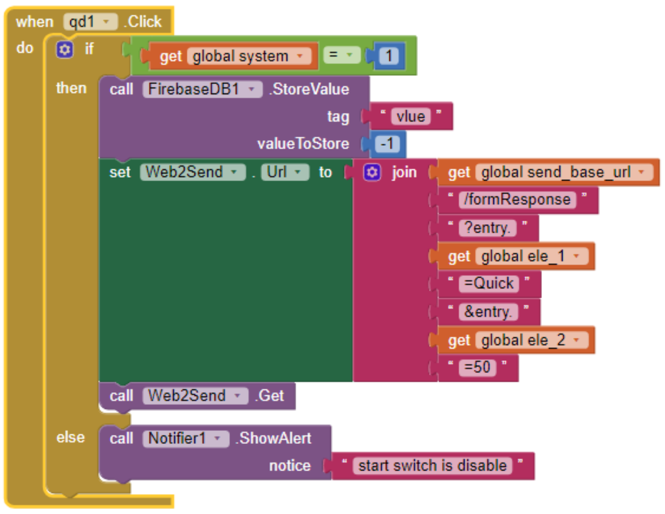
```
when Slider1 .PositionChanged
  thumbPosition
do  set slider_value . Text to  get thumbPosition
    set global sldr_val to  get thumbPosition
```

- It is the code which fetches data of slider variable and put it to the “sldr_val” which is used to set medicine dosages.



```
when FirebaseDB1 .DataChanged
  tag  value
do  if   get value = 0
    then call Notifier1 .DismissProgressDialog
    else call Notifier1 .ShowProgressDialog
              message  " Wait a momemts till next dose. "
              title    " Dispensing... "
```

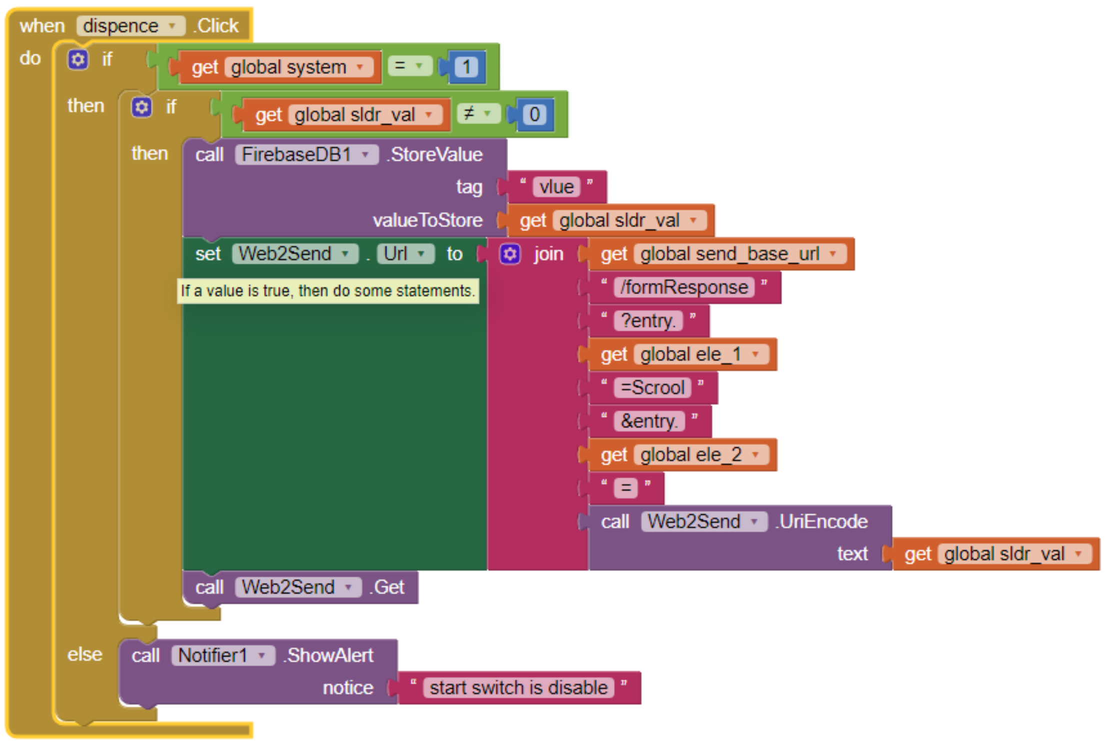
- This code always runs when there is a data change in Firebase.
- When the data is not 0, it means the device is dispensing medicine, so it shows a popup.
- The popup disappears when the dosage is completed, and the Firebase data becomes 0.



```
when qd1 .Click
do  if   get global system = 1
    then call FirebaseDB1 .StoreValue
              tag          " vlue "
              valueToStore -1
         set Web2Send . Url to  join  get global send_base_url
                                      " /formResponse "
                                      " ?entry. "
                                      get global ele_1
                                      " =Quick "
                                      " &entry. "
                                      get global ele_2
                                      " =50 "
         call Web2Send .Get
    else call Notifier1 .ShowAlert
              notice  " start switch is disable "
```

1. **Event Trigger (when qd1.Click)**:
   - This block starts when the button with the identifier **qd1** is clicked.
2. **Condition Check (if get global system = 1)**:
   - It checks if the global variable system is equal to 1 which is the toggle button to on/off dispensing button.

3. **If Condition is True**:
   - **Store Value in FirebaseDB**:
     - FirebaseDB1.StoreValue is called to store the value **-1(50ml)** with the tag **vlue** in the Firebase database.
     - For qd2 its store -2**(75ml)**
     - For qd3 its store -3**(100ml)**
   - **Set Web2Send URL**:
     - Web2Send.Url is set using a concatenation of various parts:
       - get global **send_base_url**: Base URL for sending data.
       - "/formResponse": URL endpoint.
       - "?entry." & get global ele_1 & "=" & get global Quick & "&entry." & get global ele_2 & "=50": Query parameters. This concatenates several global variables and fixed strings to form a complete URL.
   - **Send Web Request**:
     - Web2Send.Get is called to send the HTTP GET request to the constructed URL.
4. **If Condition is False**:
   - **Show Alert**:
     - Notifier1.ShowAlert is called with the notice "start switch is disable," indicating that the system is not enabled.



```
when dispence .Click
do  if   get global system = 1
    then  if   get global sldr_val ≠ 0
          then  call FirebaseDB1 .StoreValue
                                  tag  " vlue "
                         valueToStore  get global sldr_val
                set Web2Send . Url to  join  get global send_base_url
                                             " /formResponse "
                                             " ?entry. "
                                             get global ele_1
                                             " =Scrool "
                                             " &entry. "
                                             get global ele_2
                                             " = "
                                             call Web2Send .UriEncode
                                                            text  get global sldr_val
                If a value is true, then do some statements.
                call Web2Send .Get
    else  call Notifier1 .ShowAlert
                          notice  " start switch is disable "
```

Explanation

1. **Event Trigger (when dispence.Click)**:
   - This block of code starts when the button with the identifier dispence is clicked.
2. **Condition Check (if get global system = 1)**:
   - It checks if the global variable system is equal to 1.
3. **Nested Condition Check (if get global sldr_val ≠ 0)**:
   - It checks if the global variable sldr_val is not equal to 0.
4. **If Both Conditions are True**:
   - **Store Value in FirebaseDB**:
     - FirebaseDB1.StoreValue is called to store the value of global sldr_val with the tag vlue in the Firebase database.
   - **Set Web2Send URL**:
     - Web2Send.Url is set using a concatenation of various parts:
     - get global send_base_url: Base URL for sending data.
     - "/formResponse": URL endpoint.
     - "?entry." & get global ele_1 & "=Scrool&entry." & get global ele_2 & "=" & Web2Send.UriEncode(get global sldr_val)

        - This concatenates several global variables and fixed strings to form a complete URL. The UriEncode block ensures that the value of global sldr_val is properly encoded for inclusion in a URL.
    - **Send Web Request**:
        - Web2Send.Get is called to send the HTTP GET request to the constructed URL.
5. **If Either Condition is False**:
    - **Show Alert**:
        - Notifier1.ShowAlert is called with the notice "start switch is disable," indicating that the system is not enabled.

CONTENTS OF THE PROJECT: Node MCU

Code of NODE MCU

```
#include <WiFiManager.h>
#include <Firebase_ESP_Client.h>
/* for OLED display */
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

/* -------------------------------------------------------------
---------------------------------------- */
/* OLED setup */
#define SCREEN_WIDTH 128  // OLED display width, in pixels
#define SCREEN_HEIGHT 32  // OLED display height, in pixels

#define OLED_RESET -1        // Reset pin # (or -1 if sharing
Arduino reset pin)
#define SCREEN_ADDRESS 0x3C  ///< See datasheet for Address;
0x3D for 128x64, 0x3C for 128x32


/* -------------------------------------------------------------
---------------------------------------- */

/* Firebase defines */
/* 2. Define the API Key */
#define API_KEY "your_key"

/* 3. Define the RTDB URL */
#define DATABASE_URL "your_url-rtdb.firebaseio.com/"

/* 4. Define the user Email and password that alreadey
registerd or added in your project */
#define USER_EMAIL "your_user_name@gmail.com"
#define USER_PASSWORD "123456"

/* -------------------------------------------------------------
---------------------------------------- */
```

```
/* oled initialize and all bitmaps */
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);

const unsigned char PROGMEM init_logo[] = {
  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x00, 0x00, 0x7f,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff………., 0xff, 0xff, 0xe0,
0x00, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff
};

static const uint8_t PROGMEM wifi_logo[] = {
  0xff, 0xff, 0xff, 0xff, 0xff, …………………
  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff
};


// 'succes21', 128x32px
const unsigned char PROGMEM bitmap_succes[] = {
  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0…………………
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff
};



/* ------------------------------------------------------------
----------------------------------------- */
// Define Firebase Data object
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;
WiFiManager wm;


//user created variable
unsigned long sendDataPrevMillis = 0;
String sValue;
int val;
int speaker = D3;
int pus_btn = D4;
int motor = D6;
int prev_btn_state = HIGH;
int curr_btn_state;
/* -------------------------------------------------------------
----------------------------------------- */


void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(speaker, OUTPUT);
  pinMode(motor,OUTPUT);
  pinMode(pus_btn, INPUT_PULLUP);

  digitalWrite(LED_BUILTIN, LOW);

  Serial.begin(115200);



  //display begins
  connectDisplay();
  initDisplay();
  track1();
  track1();
  track1();
  track1();
  track1();
  track1();
  wifiConnecting();

  if (digitalRead(pus_btn) == LOW) {
    wm.resetSettings();
    ESP.restart();
  }


  /* WiFi manager function */
  setupWiFiManager();

  //if wifi is connected the print wifi is connected
  wifiConnected();
  /* Assign the api key (required) */
  config.api_key = API_KEY;

  /* Assign the user sign in credentials */
  auth.user.email = USER_EMAIL;
  auth.user.password = USER_PASSWORD;

  /* Assign the RTDB URL (required) */
  config.database_url = DATABASE_URL;

  // Since v4.4.x, BearSSL engine was used, the SSL buffer
need to be set.
  // Large data transmission may require larger RX buffer,
otherwise connection issue or data read time out can be
occurred.
  fbdo.setBSSLBufferSize(4096 /* Rx buffer size in bytes from
512 - 16384 */, 1024 /* Tx buffer size in bytes from 512 -
16384 */);

  // Limit the size of response payload to be collected in
FirebaseData
  fbdo.setResponseSize(2048);

  Firebase.begin(&config, &auth);

  Firebase.setDoubleDigits(5);

  config.timeout.serverResponse = 10 * 1000;
}


void loop() {
  curr_btn_state = digitalRead(pus_btn);

  /* if wifi is disconnected by any mean try to reconnect by
restarting the esp. */
  while (WiFi.status() != WL_CONNECTED) {
    Serial.println("wifi disconnected,  connecting..");
    Serial.print(".");
    ESP.restart();
    delay(500);
  }
  if ((prev_btn_state == HIGH) && (curr_btn_state == LOW)) {
    wm.resetSettings();
    ESP.restart();
  }
  prev_btn_state = curr_btn_state;


  if (Firebase.ready() && (millis() - sendDataPrevMillis >
1000 || sendDataPrevMillis == 0)) {
    sendDataPrevMillis = millis();

    if (Firebase.RTDB.getString(&fbdo, "/vlue")) {
      if (fbdo.dataType() == "string") {
        sValue = fbdo.to<String>();
        val = sValue.toInt();
        Serial.println(val);


        if (val != 0) {
          if (val < 0) quickDispence(val);
          else scrlDispence(val);

          Serial.println(" medecine despenses successfullyy
value setting to zero");
          Firebase.RTDB.setString(&fbdo, "/vlue", "0");
          Serial.println("the value is set to zero");
        }
      }
    } else {
      Serial.println(fbdo.errorReason().c_str());
    }
  }
}


void setupWiFiManager() {
  track3();
  WiFi.mode(WIFI_STA);
  // wm.resetSettings();
  WiFi.forceSleepWake();
  WiFi.setSleepMode(WIFI_NONE_SLEEP);


  wm.setConnectTimeout(180);
  wm.setConnectRetries(100);
  if (wm.getWiFiIsSaved()) wm.setEnableConfigPortal(false);

  bool res;
  res = wm.autoConnect("AutoConnectAP");  // anonymous ap

  if (!res) {
    Serial.println("Failed to connect");
    ESP.restart();
  } else {
    //if wifi is  connected
    Serial.println("connected... :)");
    track2();
  }
}


void dd(int quan) {
  track3();
  track3();
  int x = quan / 20;
  int error = x * 2;
  quan -= error;
  int to_time = (quan * 100 /* *1666 */);
  /*digitalWrite(po,HIGH);
      digitalWrite(ne,LOW);
      delay(to_time);
      digitalWrite(po,LOW);*/

  digitalWrite(motor, HIGH);
  delay(to_time);
  digitalWrite(motor, LOW);
}

void quickDispence(int a) {
  if (a == -1) {
    formatMsg(50, 2);
    dd(50);
    Serial.println("quick despence 50ml");
  } else if (a == -2) {
    formatMsg(75, 2);
    dd(75);
    Serial.println("quick despence 75ml");
  } else if (a == -3) {
    formatMsg(100, 2);
    dd(100);
    Serial.println("quick despence 100ml");
  }
  dispense_successful();
}


void scrlDispence(int a) {
  formatMsg(a, 1);
  dd(a);
  dispense_successful();
  Serial.print("dispensing ");
  Serial.print(a);
  Serial.println("ml via scroll method in normal speed");
}


/* --------------------------------------------------------------
----------------------------------------- */

void connectDisplay() {
  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V
internally
  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;)
      ;  // Don't proceed, loop forever
  }
  // Clear the buffer
  display.clearDisplay();
}

void initDisplay() {
  display.clearDisplay();
  display.drawBitmap(0, 0, init_logo, 128, 64, WHITE);
  display.display();
  delay(2000);
}

void wifiConnecting() {
  displayMessage("Connect AutoConnectAP for further setup.",
1, 0, 0);
  delay(6000);

  //showing bitmap and then scrool it
  display.clearDisplay();
  display.drawBitmap(0, 0, wifi_logo, 128, 64, WHITE);
  display.display();
  display.startscrollright(0x00, 0x0F);
}

void wifiConnected() {
  display.clearDisplay();
  display.stopscroll();

  displayMessage("Connected", 2, 15, 0);
  delay(2000);

  displayMessage("Ready to dispence...", 1, 0, 10);
}

void displayMessage(const char* message, int textSize, int x,
int y) {
  display.clearDisplay();
  display.setTextSize(textSize);  // Set text size based on
the argument
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(x, y);
  display.println(message);  // Display the message based on
the argument
  display.display();
}

void formatMsg(int vol, int method) {
  //1:scrool
  display.clearDisplay();

  display.setTextSize(2);  // Set text size based on the
argument
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(32, 0);

  display.println((String(vol) + "ml").c_str());

  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  String msg = "  dispensing via";
  if (method == 1) {
    msg += "\n  SCROOL";
  } else {
    msg += "\n  QUICK";
  }
  msg += " method...";
  display.println(msg.c_str());
  display.display();
}

void dispense_successful() {
  display.clearDisplay();
  display.drawBitmap(0, 0, bitmap_succes, 128, 64, WHITE);
  display.display();
  track4();
  displayMessage("Ready to dispence...", 1, 0, 10);
}

void playTone(int pin, int duration, int after_duration) {
  digitalWrite(pin, HIGH);
  delay(duration);
  digitalWrite(pin, LOW);
  delay(after_duration);
}


void track1() {
  playTone(speaker, 10, 5);
  playTone(speaker, 20, 15);
  playTone(speaker, 20, 15);
  playTone(speaker, 10, 5);
```

```
}

void track2() {
  playTone(speaker, 20, 100);
  playTone(speaker, 40, 10);
  playTone(speaker, 40, 10);
  playTone(speaker, 20, 100);
}

void track3() {
  playTone(speaker, 30, 10);
  playTone(speaker, 10, 100);
  playTone(speaker, 30, 10);
}

void track4() {
  playTone(speaker, 1000, 100);
  playTone(speaker, 5, 5);
  playTone(speaker, 30, 50);
  playTone(speaker, 20, 20);
  playTone(speaker, 20, 20);
}
```

EXPLANATION:

Library Inclusions

The code starts by including necessary libraries:

- **WiFiManager.h**: To handle Wi-Fi connection management.
- **Firebase_ESP_Client.h**: To interact with Firebase.
- **SPI.h and Wire.h**: For SPI and I2C communication, necessary for OLED display.
- **Adafruit_GFX.h and Adafruit_SSD1306.h**: To manage graphics on the OLED display.

OLED Display Setup

Defines the width, height, reset pin, and screen address for the OLED display. These parameters are essential for initializing and controlling the display.

Firebase Configuration

The code defines several constants required to connect to Firebase:

- **API_KEY**: The API key for accessing Firebase services.
- **DATABASE_URL**: The URL of the Firebase Realtime Database.
- **USER_EMAIL and USER_PASSWORD**: Credentials for logging into Firebase.

OLED Display Initialization

Initializes the OLED display using the Adafruit_SSD1306 library. The OLED display is connected via I2C (using the Wire library), and its properties are defined earlier.

Bitmap Definitions

The code includes bitmap definitions for various images to be displayed on the OLED screen. These images are stored in PROGMEM to save RAM.

- **init_logo**: A bitmap array representing an initial logo.
- **wifi_logo**: A bitmap array for displaying a Wi-Fi logo.
- **bitmap_succes**: A bitmap array for displaying a success message.

Libraries and Objects

Firebase Data and Configurations:

- **FirebaseData fbdo**: Object for Firebase operations.
- **FirebaseAuth auth**: Object for Firebase authentication.
- **FirebaseConfig config**: Object for Firebase configuration.
- **WiFiManager wm**: Object to manage WiFi connections.

User-Created Variables:

- **unsigned long sendDataPrevMillis = 0**: Keeps track of time for sending data.
- **String sValue**: Stores string values from Firebase.
- **int val**: Stores integer values converted from the string.
- **int speaker = D3**: GPIO pin for the speaker.
- **int pus_btn = D4**: GPIO pin for the push button.

- **int motor = D6**: GPIO pin for the motor.
- **int prev_btn_state = HIGH**: Stores the previous state of the button.
- **int curr_btn_state**: Stores the current state of the button.

Setup Function

Pin Modes:

- Initialize the built-in LED pin, speaker pin, and motor pin as outputs.
- Initialize the push button pin as input with an internal pull-up resistor.

Initial States:

- Turn off the built-in LED.
- Start serial communication at 115200 baud rate.

Display Setup:

- Initialize the display and show the WiFi connection process.

WiFi Connection:

- If the push button is pressed during startup, reset WiFi settings and restart Node MCU.
- Call function to handle WiFi connections.
- Confirm WiFi connection.

Firebase Configuration:

- Set API key, user credentials, and database URL.
- Configure SSL buffer sizes and response payload size.
- Initialize Firebase with configuration and authentication.
- Set double-digit precision and server response timeout.

Loop Function

WiFi Reconnection:

- If WiFi is disconnected, print a message and restart Node MCU.

Button State Handling:

- Read the current state of the push button.
- If the button state changes from HIGH to LOW, reset WiFi settings and restart Node MCU.
- Update the previous button state.

Firebase Data Handling:

- If Firebase is ready and enough time has passed, update the timestamp.

- Retrieve string value from Firebase.
- Convert the string to integer and print it.
- If the value is not zero, dispense medicine.
- Set the value to zero in Firebase after dispensing.
- Print error if Firebase operation fails.

WiFi Manager Setup Function

WiFi Manager Configuration:

- Configure WiFi mode and settings.
- Set connection timeout and retries.
- Disable config portal if WiFi is saved.
- Attempt to connect to WiFi.
- If connection fails, restart Node MCU. If successful, print message and call track2().

Medicine Dispensing Functions

Dispensing Function:

- Control the motor for dispensing medicine.
- Calculate dispensing time.
- Activate motor for the calculated time.

Quick Dispense Function:

- Quickly dispense predefined amounts.
- Dispense 50 ml, 75 ml, or 100 ml based on input.
- Indicate successful dispensing.

Scroll Dispense Function:

- Dispense specified amount at normal speed.

Display Functions

Display Initialization:

- Initialize the display.
- Check display initialization and halt if failed.
- Clear the display buffer.

Display Messages:

- Display initial logo.
- Show WiFi connecting message.
- Show WiFi connected message.
- Display custom messages.

Format Messages for Dispensing:

- Format messages for different dispensing methods.

Dispense Successful Message:

- Show successful dispensing message.

Sound Functions

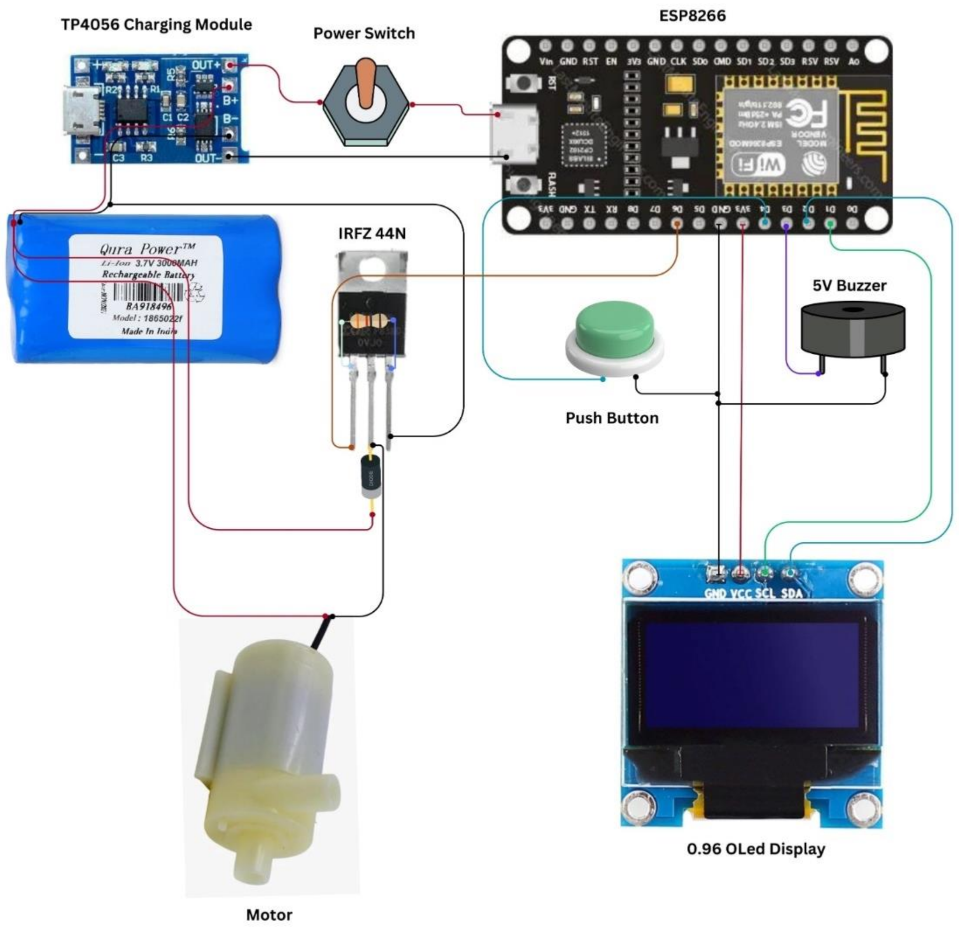
Play Tones:

- Play tones on the speaker.
- Define and play sequences for track 1, track 2, track 3, and track 4.

Summary

This code handles the setup and operation of an automated medicine dispensing system. It connects to WiFi, integrates with Firebase for receiving dispensing commands, controls hardware components (motor, speaker, display), and provides user feedback through the display and speaker. The system ensures accurate dispensing of medicine based on commands received from the Firebase database and displays relevant messages and statuses to the user.

CONTENTS OF THE PROJECT: CIRCUIT



This is the circuit diagram of Medicine dispensing system.

1. **Power Supply**:
   - A rechargeable lithium-ion battery (3.7V) provides power to the system.
   - The battery is connected to a TP4056 charging module for charging purposes.
   - A power switch is included to turn the system on and off.
2. **Microcontroller (ESP8266)**:
   - The ESP8266 microcontroller is the brain of the system. It is powered through the TP4056 module.
   - The ESP8266 has several GPIO (General Purpose Input/Output) pins connected to various components.
3. **IRFZ 44N MOSFET**:

   - This MOSFET is used to control the motor. It acts as a switch that allows high-current loads to be driven by the microcontroller’s low-power output.
4. **Motor**:
   - The motor is connected to the drain of the MOSFET and to the ground. When the MOSFET is activated, the motor runs.
5. **Push Button**:
   - A push button is connected to a GPIO pin on the ESP8266. This can be used to trigger actions such as dispensing medicine.
6. **5V Buzzer**:
   - A buzzer is connected to a GPIO pin on the ESP8266. It can provide audio alerts or notifications.
7. **0.96” OLED Display**:
   - The OLED display is connected to the ESP8266 via I2C (SDA and SCL pins). This display can show information such as time, dosage schedules, or other relevant data.

Connections:

1. **Battery to TP4056:**
   a. The battery's positive terminal is connected to the B+ terminal on the TP4056.
   b. The battery's negative terminal is connected to the B- terminal on the TP4056.
   c. The OUT+ and OUT- terminals of the TP4056 provide power to the ESP8266 and the rest of the components.
2. **TP4056 to Power Switch and ESP8266:**
   a. The OUT+ terminal of the TP4056 is connected to one side of the power switch.
   b. The other side of the power switch is connected to the VCC pin of the ESP8266.
   c. The OUT- terminal of the TP4056 is connected to the GND pin of the ESP8266.
3. **ESP8266 to Push Button:**
   a. One side of the push button is connected to a GPIO pin on the ESP8266.
   b. The other side of the push button is connected to the ground.
4. **ESP8266 to Buzzer**:
   a. The positive terminal of the buzzer is connected to a GPIO pin on the ESP8266.
   b. The negative terminal of the buzzer is connected to the ground.
5. **ESP8266 to OLED Display:**
   a. The SDA pin of the OLED display is connected to the corresponding SDA GPIO pin on the ESP8266.
   b. The SCL pin of the OLED display is connected to the corresponding SCL GPIO pin on the ESP8266.
   c. The VCC and GND pins of the OLED display are connected to the VCC and GND pins of the ESP8266.

6. **ESP8266 to IRFZ 44N MOSFET**:
    a. The gate of the MOSFET is connected to a GPIO pin on the ESP8266.
    b. The drain of the MOSFET is connected to one terminal of the motor.
    c. The source of the MOSFET is connected to the ground.
7. **Motor:**
    a. The other terminal of the motor is connected to the battery's positive terminal through the MOSFET.

ITS RELEVANCE AND IMPLICATION IN COMPANY

The IoT-based Automated Medicine Dispensing System holds significant relevance and implications for healthcare companies. By integrating IoT technology with medication administration, this system enhances operational efficiency, ensuring that patients receive the correct dosage of medication at the right time. This precision is critical in reducing medication errors, which are a common and costly issue in healthcare settings.

For companies, implementing this system can lead to improved patient outcomes and satisfaction, as it ensures consistent and accurate medication delivery. It also streamlines the workflow for healthcare providers, reducing the time and effort required to manually dispense medications. This allows healthcare professionals to focus more on patient care rather than administrative tasks.

The data logging feature, which records each dispensation event in Google Sheets, provides valuable insights into patient medication patterns and adherence. This information can be used to optimize treatment plans, identify potential issues early, and improve overall patient management.

Moreover, the use of a cloud-based system like Firebase ensures scalability and remote access, making it suitable for various healthcare settings, from hospitals to home care. Overall, the adoption of this system can lead to enhanced efficiency, reduced costs, and better healthcare outcomes, making it a valuable investment for healthcare companies.

FINDING

The findings from the implementation of the IoT-based Automated Medicine Dispensing System reveal several key benefits and insights. Firstly, the system significantly improves the accuracy of medication administration. By automating the process of dispensing medicine based on precise inputs from healthcare providers, it virtually eliminates human errors related to dosage, timing, and medication type.

Secondly, the integration with cloud services like Firebase allows for real-time monitoring and data logging. This feature ensures that every dispensation event is recorded with the exact date, time, and amount of medication dispensed. Such detailed records are invaluable for ensuring compliance with medical guidelines and for conducting audits or reviews of patient care.

Additionally, the historical data accessible through the mobile application offers a comprehensive view of a patient's medication history. This can help in identifying trends, improving treatment plans, and enhancing patient adherence to prescribed therapies. The ease of access to this data also supports better communication and coordination among healthcare providers.

Furthermore, the system's ability to reduce the workload of healthcare professionals by automating routine tasks allows them to focus more on direct patient care. This leads to better resource allocation and potentially improved patient outcomes. Overall, the findings highlight the system's role in enhancing efficiency, accuracy, and quality of healthcare delivery.

CONCLUSION

In conclusion, the IoT-based Automated Medicine Dispensing System demonstrates significant potential to transform healthcare delivery. By automating the medication administration process, it enhances accuracy, reduces human errors, and ensures timely delivery of medicines. The integration with cloud services like Firebase for real-time monitoring and data logging provides invaluable insights into patient medication patterns, aiding in better treatment planning and adherence.

The system's ability to record and access detailed medication histories improves communication and coordination among healthcare providers, leading to more informed and efficient patient care. Additionally, by reducing the administrative burden on healthcare professionals, the system allows them to allocate more time to direct patient care, thus improving overall patient outcomes.

This innovative solution not only optimizes operational efficiency but also significantly enhances the quality of care, making it a valuable investment for healthcare institutions aiming to improve patient safety and treatment effectiveness.

LIMITATION OF THE PROJECT

1. **Internet Dependence**:
   - Relies on a stable internet connection for real-time data transfer and monitoring.
   - Connectivity issues can delay medication dispensation and data logging.
2. **Technical Expertise**:
   - Initial setup and maintenance require technical skills.
   - May pose a barrier for healthcare facilities with limited technological resources.
3. **System Integration**:
   - Challenges in integrating with existing healthcare systems.
   - Ensuring data security and privacy is critical.
4. **Input Accuracy**:
   - Dependent on accurate input of medication dosages by healthcare providers.
   - Errors in input can lead to incorrect medication dispensation.
5. **Cost**:
   - Implementation costs, including hardware and software, might be high.

FURTHER ENHANCEMENT

1. **Offline Functionality**:
   - Develop offline capabilities to ensure continuous operation during internet outages.
   - Synchronize data with the cloud once connectivity is restored.
2. **User-Friendly Interface**:
   - Simplify the setup and maintenance process to reduce the need for technical expertise.
   - Provide comprehensive training and support for healthcare staff.
3. **Enhanced Security**:
   - Implement advanced security measures to protect patient data and ensure privacy.
   - Regularly update the system to address emerging security threats.
4. **AI and Machine Learning**:
   - Integrate AI to analyze medication patterns and predict patient needs.
   - Use machine learning to improve dosage accuracy and personalize treatment plans.
5. **Scalability**:
   - Design the system to be easily scalable for different healthcare settings.
   - Ensure compatibility with various medical devices and healthcare software.
6. **Cost Reduction**:
   - Explore cost-effective hardware and software solutions to lower implementation expenses.
   - Offer flexible pricing models for smaller healthcare providers.

7. **Real-Time Alerts**:
    - Implement real-time alerts for missed or delayed doses to ensure timely medication administration.
    - Notify healthcare providers and caregivers of any issues immediately.
8. **Comprehensive Reporting**:
    - Enhance reporting features to provide detailed analytics and insights.
    - Allow customizable reports for better patient management and decision-making.
9. **Remote Monitoring**:
    - Enable remote monitoring capabilities for healthcare providers and caregivers.
    - Allow for adjustments to medication schedules and dosages remotely.
10. **Integration with EHRs**:
    - Integrate with Electronic Health Records (EHR) systems for seamless data exchange.
    - Ensure comprehensive and up-to-date patient medical histories.