

PROJECT REPORT



VIT-AP UNIVERSITY

PROJECT TITLE:

Smart Dose: Automated pill organizer with Bluetooth scheduling

Submitted by:

- 1) Devulapalli Datta Sai Srinivas - 24BES7058
- 2) Vattikuti Sri Sasank - 24BEV7035
- 3) Baswareddy Sreesanth Reddy - 24BES7040
- 4) Moka Siva Pujitha - 24BES7056
- 5) Vemuri Karthikeya - 24BEV7037

Under the Guidance of:

Dr. Ramesh A

Department of Advanced Science

Vellore Institute of Technology, Andhra Pradesh (VIT-AP University)

Abstract:

The aging global population faces a significant challenge in medication adherence, which can lead to severe health complications and increased hospitalization rates. "Smart Dose" is an innovative, cost-effective solution designed to assist the elderly and patients with complex medication regimens in managing their pill intake efficiently. This project integrates a hardware-based smart medicine box with a user-friendly mobile application via Bluetooth connectivity.

The smart medicine box, built around an Arduino Uno R3 microcontroller, features four distinct compartments, each controlled by a servo motor. A Real-Time Clock (RTC) module ensures precise timing, while an HC-05 Bluetooth module facilitates seamless communication with the mobile app developed using MIT App Inventor. The system is designed to automatically open the designated compartment at pre-scheduled times, accompanied by LED indicators, an LCD display, and an audible buzzer as reminders. The system also includes manual override buttons and a feedback mechanism to track medication intake.

The accompanying mobile application allows caregivers or users to easily configure medication schedules, including pill names, dosage times, and initial counts. The app provides real-time status updates and sends push notifications for low stock and scheduled doses. This project demonstrates a functional prototype that successfully automates and reminds for medication schedules, thereby promoting independence and improving healthcare outcomes for its users.

Index

S. No	Content Page Title	Page No.
1	Introduction	4
2	Background	5
3	Problem Definition	6
4	Objectives of Proposed Work	7
5	Methodology/Procedure	8
5.1	System Architecture	8-9
5.2	Hardware design and Components	9-10
5.3	Software Design and development	10-11
5.4	Integration and Working Principle	11-12
6	Results and Discussion	13
7	Conclusion and Future Scope	14
8	References	15
9	Appendix (Code)	16

List of Figures:

Figure No:	Figure Name	Page
1.1	Block Diagram	12
1.2	Circuit Diagram	12
1.3	Hardware Setup	13
2.1	App Interface - I	15
2.2	App Interface -II	15

Introduction:

The adherence to prescribed medication schedules is a critical factor in managing chronic illnesses, especially for the elderly population. However, cognitive decline, memory lapses, and complex multi-drug regimens often lead to missed or incorrect dosages. This non-adherence results in worsening health conditions, increased hospital visits, and higher healthcare costs. There is a pressing need for assistive technologies that can bridge this gap between medical prescription and practical consumption.

This project, titled "Smart Dose," proposes an integrated hardware and software solution to address this challenge. It comprises a smart, connected medicine box that works in tandem with a mobile application. The primary goal is to automate the reminder and dispensing process, thereby reducing the cognitive load on the user. The system not only alerts the user at the correct time but also provides a physical barrier (a closed box) that is only opened during the scheduled medication time or via manual override, ensuring intentionality in taking the medicine.

The mobile application empowers caregivers or family members to remotely set up and monitor the medication schedule. The bidirectional Bluetooth communication allows the system to send confirmation of pill intake or alerts regarding missed doses. By combining timely audio-visual alerts, physical compartment control, and digital tracking, Smart Dose aims to be a reliable companion for individuals requiring strict medication adherence, fostering independence and ensuring better health management.

Background:

The problem of medication non-adherence is well-documented, leading to a growing market of electronic pill dispensers and medication management apps. Existing solutions range from simple timer caps on pill bottles to advanced, internet-connected dispensers with cellular alerts. A review of existing technologies reveals several approaches.

Simple pill organizers with multiple compartments for days of the week are the most common low-tech solution. However, they rely entirely on the user's memory to take the medication at the correct time. Advanced electronic dispensers often feature alarms and automatic dispensing of single doses. Many of these systems are expensive and can be complex for elderly users to set up and operate.

The proliferation of smartphones has led to numerous medication reminder apps. These apps send notifications but lack a physical component to ensure the medication is actually taken. They cannot prevent double-dosing or confirm intake. Furthermore, they are ineffective for users who are not tech-savvy or do not keep their phones nearby at all times.

The Smart Dose project draws from these existing models but aims to create a more balanced and accessible solution. It leverages the ubiquitous nature of smartphones for easy configuration by a caregiver while ensuring the primary interaction for the end-user (the patient) is with a simple, dedicated hardware device. The use of Bluetooth keeps the system cost low and power-efficient compared to Wi-Fi-based solutions, and the physical design with individual boxes prevents the mixing of different medications.

Problem definition:

The core problem addressed by this project is the unreliable management of medication schedules by elderly individuals and patients with complex prescription regimens. This problem can be decomposed into several specific issues:

1. **Forgetting to Take Medication:** Users often forget whether they have taken their medication at the scheduled time, leading to missed doses or accidental double-dosing.
2. **Complex Schedules:** Managing multiple medications with different timings throughout the day (e.g., morning, afternoon, evening, night) becomes confusing and error-prone.
3. **Lack of Immediate Reminders:** While phone alarms can be used, they are non-specific and can be easily dismissed or ignored. There is a need for a more persistent, multi-sensory reminder system.
4. **Inventory Management:** Users or caregivers may lose track of the remaining quantity of pills, leading to situations where the medication runs out unexpectedly.
5. **Absence of Confirmation:** Caregivers or family members have no way of knowing if the patient has actually taken the medication, creating anxiety and potential for health risks.

Therefore, the problem is to design and develop an integrated system that automatically manages up to four different medications per day, provides robust and multi-sensory reminders, prevents access to medication outside of scheduled times (with a manual override), tracks pill inventory, and provides remote monitoring and configuration capabilities via a smartphone application.

Objectives of Proposed Work:

The primary objective of the *Smart Dose* project is to develop an intelligent and automated pill dispensing system that enhances medication adherence and minimizes human error. The project combines hardware and software to provide a reliable, low-cost, and user-friendly solution. The key objectives are:

1. To design and construct a four-compartment medicine dispenser, each controlled by a servo motor for automatic opening and closing based on scheduled timings.
2. To develop a Bluetooth-enabled mobile application using MIT App Inventor for setting medication schedules, managing pill names, timings, and tracking stock levels.
3. To implement real-time synchronization between the mobile app and the Arduino Uno microcontroller using the HC-05 Bluetooth module.
4. To integrate a Real-Time Clock (RTC) DS3231 module to ensure accurate timekeeping and timely medication alerts.
5. To provide a multi-mode alert system using LEDs, buzzers, and an LCD display for clear and effective user notifications.
6. To include a feedback mechanism that detects pill intake via push buttons and updates the status in both the device and mobile app.
7. To design an energy-efficient, compact, and affordable prototype suitable for elderly and chronically ill patients.

Methodology/Procedure:

The *Smart Dose* project integrates hardware and software to automate medication reminders. It uses Arduino Uno, RTC, servo motors, LEDs, buzzer, and Bluetooth for communication with an MIT App Inventor mobile app. The system dispenses pills at scheduled times, alerts users via sound and visuals, and updates the app on dosage status and tablet count.

System Architecture:

The Smart Dose system is designed using a modular architecture that connects the hardware medicine box with a mobile application through Bluetooth. The system comprises the following layers:

1. Input Layer:

Includes the user-configurable medication schedule and tablet information entered through the mobile application.

2. Processing Layer:

Controlled by the Arduino Uno R3 microcontroller, this layer interprets incoming data, manages servo motor actions, and handles real-time events based on the RTC module.

3. Output Layer:

Displays notifications, updates, and alerts through LEDs, an LCD display, and a buzzer.

4. Communication Layer:

Uses the HC-05 Bluetooth module to establish reliable bidirectional data transmission between the app and the Arduino board.

Component Specifications:

S. No	Component Name	Quantity	Specifications/Descriptions	Purpose
1	Arduino UNO R3	1	ATmega328P Microcontroller, 14 Digital I/O Pins, 5V logic, 16 MHz clock	Central control and processing unit
2	HC-05 Bluetooth Module	1	3.3–5V input, Serial UART communication	Wireless data communication with mobile app
3	RTC DS3231	1	I2C interface, Battery backup, ± 2 ppm accuracy	Real-time clock for dose scheduling
4	Servo Motor (SG90)	4	Operating voltage: 4.8–6V, Torque: 1.8 kg \cdot cm, Rotation: 0–180°	Controls medicine box lid movement
5	Push Buttons	5	Normally open type, 5V compatible	4 manual + 1 main for dose acknowledgment
6	LEDs	4	5mm Red LEDs, 2V forward drop	Visual indication of compartment active
7	Buzzer	1	5V, 85 dB sound output	Audio alert for medicine reminder
8	LCD Display (16x2 with I2C)	1	5V, I2C interface, 16x2 characters	Displays time, tablet name, and status
9	Jumper Wires (M-M, M-F)	30	20–22 AWG, 10–20 cm length	Electrical connections between components
10	Breadboard	1	830 points	Prototype connections without soldering
11	Cardboard/Plastic Compartments	4	Custom-built	Physical medicine storage boxes
12	Power Supply (USB/Adapter)	1	5V, ≥ 1 A	Power source for entire circuit

Abbreviations

RTC-Real Time Clock

LCD-Liquid Crystal Display

HC-05-Bluetooth Module

Connection Guide:

Component	Arduino Pin	Type of Connection	Description
RTC DS3231	SDA → A4, SCL → A5	I2C	Timekeeping input to Arduino
HC-05 Bluetooth	TX → RX (Pin 0), RX → TX (Pin 1), VCC → 5V, GND → GND	Serial	Sends and receives data between app and Arduino
Servo motor 1	Pin 2	PWM output	Controls 1st medicine box (Morning)
Servo motor 2	Pin 3	PWM output	Controls 2nd medicine box (Afternoon)
Servo motor 3	Pin 4	PWM output	Controls 3rd medicine box (Evening)
Servo motor 4	Pin 5	PWM output	Controls 4th medicine box (Night)
LED 1-4	Pins 8-11	Digital Output	Indicate which slot is active
Buzzer	Pin 7	Digital Output	Provides alert when dose time arrives
Main button	Pin 12	Digital Input	Used for main operation (open/confirm)
Manual Buttons(1-4)	A0-A3	Digital Input	Manually open each medicine box
LCD(I2C)	SDA → A4, SCL → A5	I2C	Displays time and messages
Power	5v,GND	Power supply	Powers all components

Power Distribution Chart:

Component	Voltage(V)	Current(mA)	Connection Source	Remarks
Arduino UNO	5V	50	USB/Adapter	Main controller
HC-05 Bluetooth	3.3-5	30	5V pin	Safe for direct connection
RTC DS3231	3.3-5	2	5V pin	Consumes very low power
LCD(I2C)	5	25	5V pin	Backlight and I2C logic
Servo Motors(x4)	5	200 x 4 = 800	5V pin (shared)	Operates one at a time to avoid overload
LEDs(x4)	2	20 x 4 = 80	5V via resistors	Visual indication
Buzzer	5	50	5V pin	Short bursts only
Buttons(x5)	5	<5	Pull-down configuration	Negligible current
Total Estimated Load	5V	~1.0A	USB/5V Adapter	Within safe limit of Arduino's external power

Block Diagram:

SMART DOSE: BLUETOOTH CONNECTED PILL DISPENSER & ORGANISER -
- BLOCK DIAGRAM

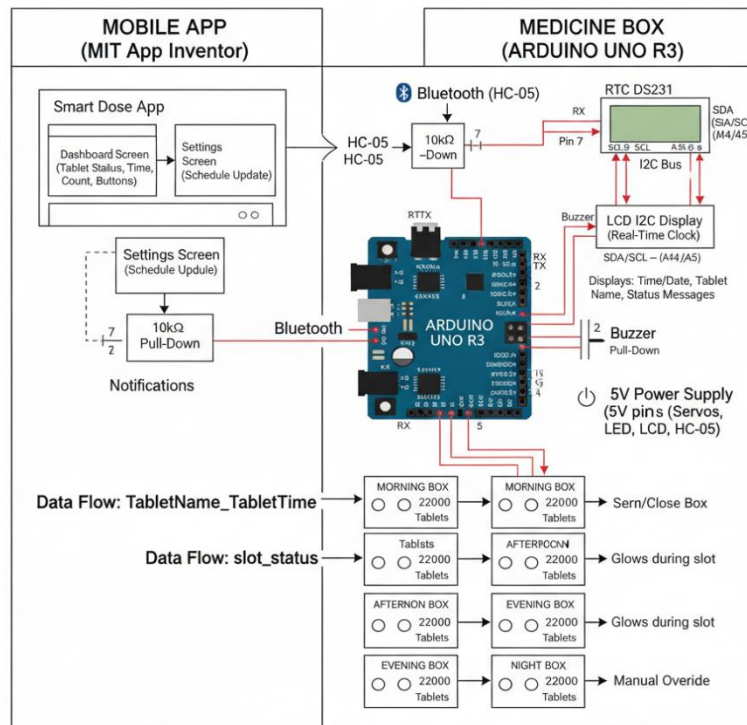


Fig :1.1

Circuit Diagram:

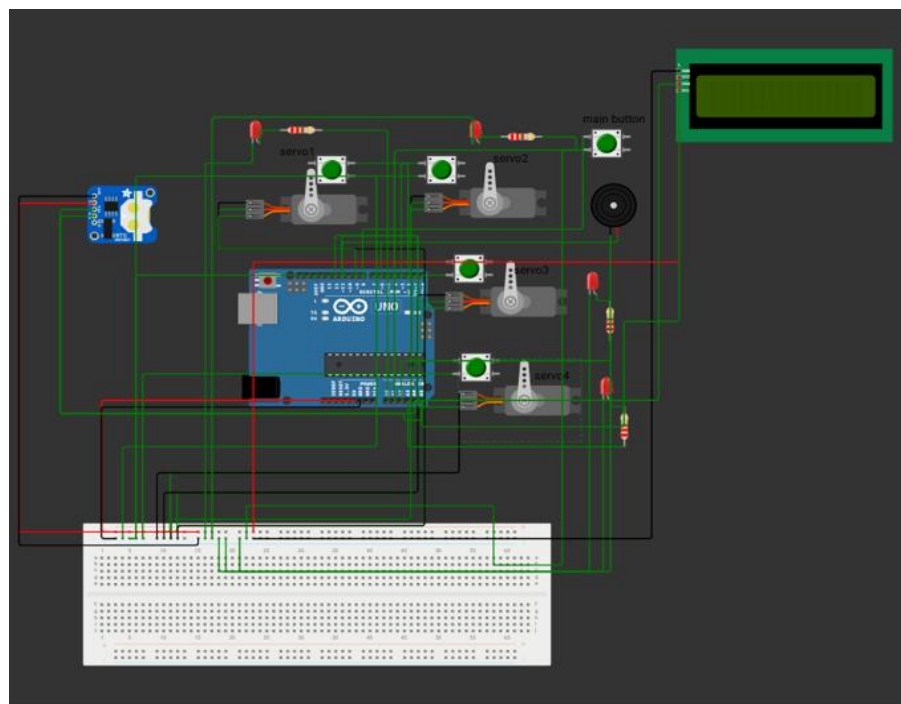


Fig :1.2

Hardware Design and Components:

The hardware design of the Smart Dose system integrates multiple electronic modules to achieve precise control and timing. The major components and their purposes are summarized below:

- **Arduino Uno R3:** Serves as the central control unit that manages servo motors, reads real-time data, and processes commands from the Bluetooth module.
- **RTC DS3231:** Maintains accurate timing for pill dispensing even when the system is powered off.
- **HC-05 Bluetooth Module:** Enables communication between the Arduino and mobile application for schedule synchronization and feedback updates.
- **Servo Motors (4 units):** Each motor controls the opening and closing of one medicine compartment corresponding to Morning, Afternoon, Evening, and Night slots.
- **LCD Display (I2C):** Shows time, tablet name, and status messages such as “Tablet Already Taken” or “No Tablet Scheduled.”
- **LED Indicators:** Visual representation of active slots—each LED glows during the designated time slot.
- **Buzzer:** Provides audio reminders at regular intervals until the tablet is taken.

Push Buttons:

- *Main Button* – Allows operation only during scheduled slots.
- *Individual Buttons* – Allow manual operation of each compartment at any time.
- **Power Supply:** The entire circuit is powered through the Arduino's 5V output, eliminating the need for an external source.

Hardware Setup:

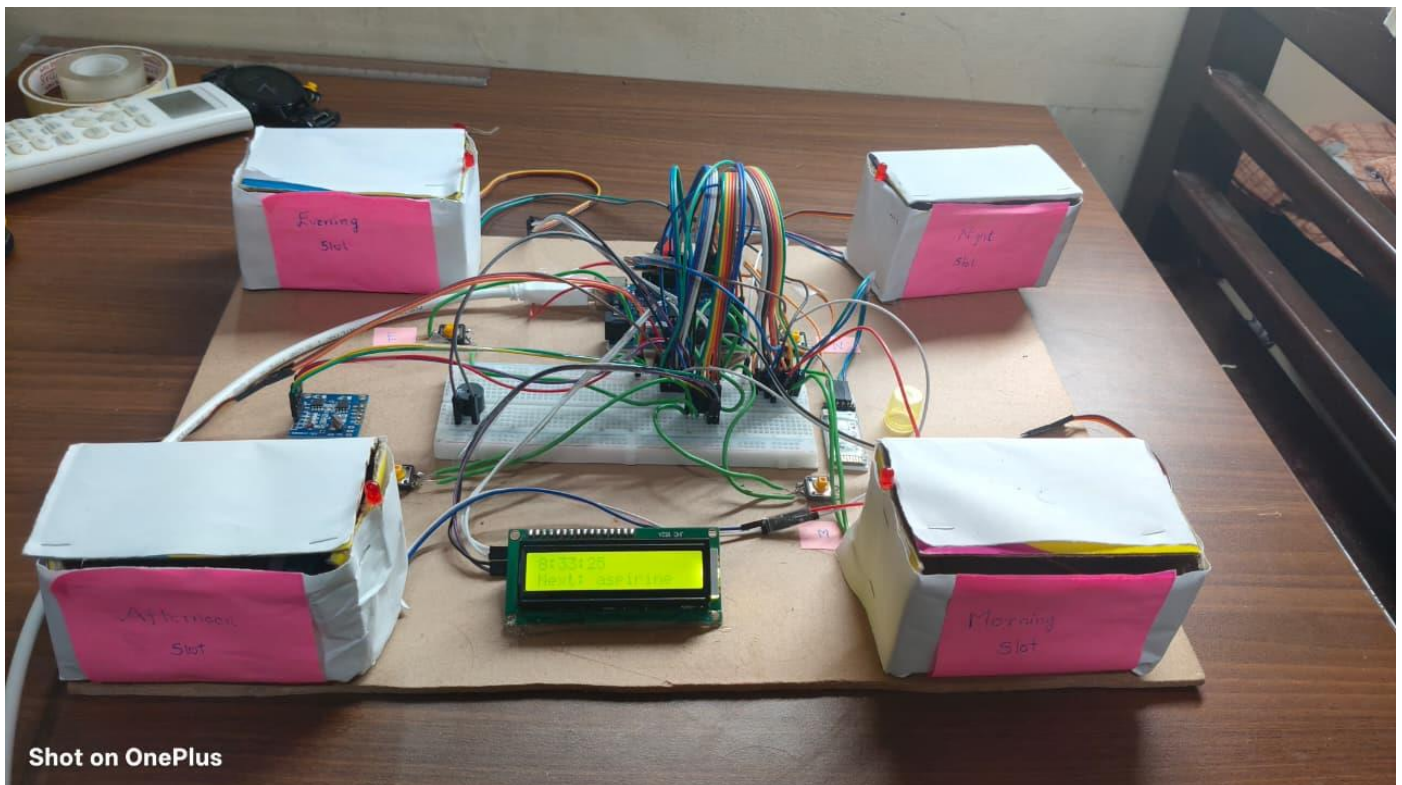


Fig :1.3

Software Design and development:

The software system comprises two primary parts: the Arduino firmware and the MIT App Inventor mobile application.

Arduino Programming:

- Written in the Arduino IDE using C/C++ language.
- Handles serial communication with the HC-05 Bluetooth module to receive tablet names and time schedules in the format *TableName_Time*.
- Parses and stores received data, activates the relevant servo motor at the scheduled time, and triggers alert signals.
- Sends real-time feedback messages such as “*Slot_opened*”, “*Slot_closed*”, or “*Slot_missed*” back to the mobile app.

Mobile Application (MIT App Inventor):

- Designed with two screens:
 - **Dashboard Screen:** Displays tablet name, timing, remaining count, and slot status with visual indicators.
 - **Settings Screen:** Allows users to configure tablet schedules and connect via Bluetooth.
- Implements push notifications to alert users of low tablet counts and upcoming schedules.
- Synchronizes tablet details and time slots dynamically when the *Save* button is pressed.

APP INTERFACE:

Screen-I

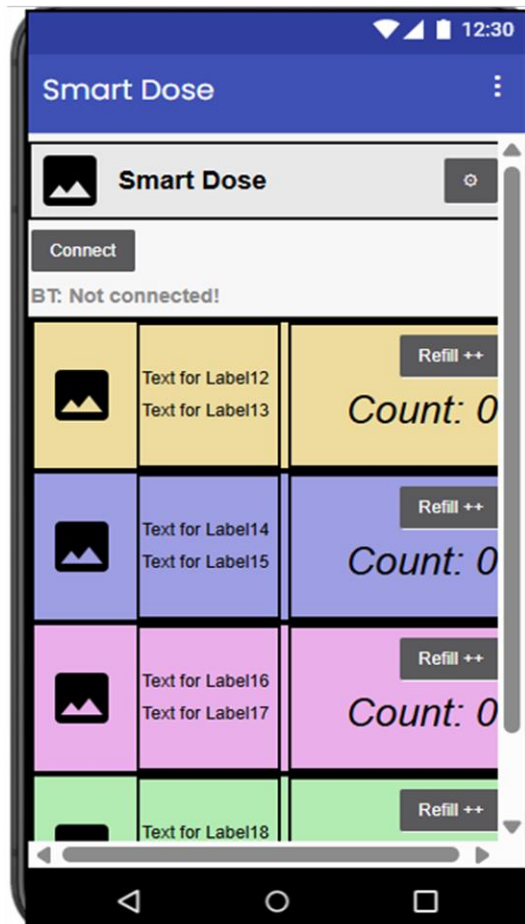


Fig :2.1

Screen-II

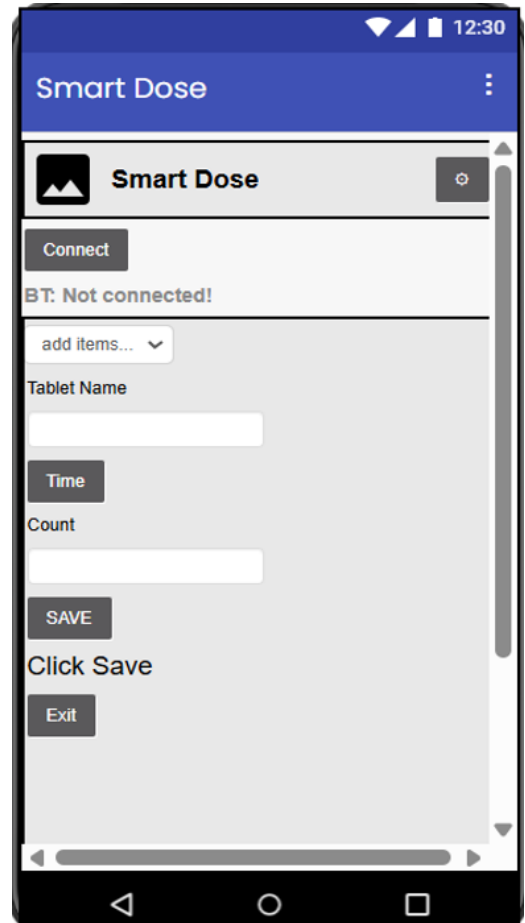


Fig :2.2

Integration and Working Principle:

Once both hardware and software components are developed, they are integrated for synchronized operation. The working principle is as follows:

1. Initialization:

The Arduino initializes all components, synchronizes time from the RTC, and establishes Bluetooth communication with the app.

2. DataTransmission:

The mobile app sends tablet name and time (e.g., *Aspirin_09:30*) to the Arduino, which parses and stores the information.

3. SlotAssignment:

Based on time, the tablet is assigned to one of the four servos (Morning, Afternoon, Evening, Night).

4. ReminderActivation:

When the current time matches the tablet's schedule, the corresponding LED glows, the servo opens the box, and the buzzer rings.

5. UserInteraction:

The user presses the main button once to open and again to close the box. If missed, the buzzer rings continuously after 30 minutes.

6. FeedbackCommunication:

The Arduino sends messages (*Slot_opened*, *Slot_closed*, or *Slot_missed*) to the app, updating the dashboard and tablet count.

This integration ensures automation, accuracy, and user convenience, making Smart Dose a practical healthcare support system.

Results and Discussion:

The Smart Dose System was successfully implemented and tested under various real-time conditions. The integration of hardware and software components ensured accurate and automated medication management.

- **Hardware Performance:**

The Arduino Uno effectively controlled all components, including the servo motors, buzzer, and LEDs. The RTC module maintained precise timing for medicine dispensing, and the HC-05 Bluetooth module enabled smooth communication with the mobile app.

- **App Functionality:**

The mobile application accurately displayed medicine schedules, allowed user configuration, and sent notifications for missed doses. Bluetooth connectivity remained stable during the tests, ensuring real-time synchronization between the app and the medicine box.

- **System Accuracy:**

The dispenser opened at the exact set times, and the buzzer provided both short and long alerts for reminders. LED indicators clearly signalled each operation, ensuring user awareness.

- **User Experience:**

The system was easy to use, with intuitive app controls and reliable operation. Real-time alerts and automatic tracking reduced the chances of missing medication doses.

Discussion:

Overall, the Smart Dose System achieved its goal of automating medicine management efficiently. Minor delays during Bluetooth reconnection were observed but did not affect the overall performance. Future improvements could include Wi-Fi connectivity and a cloud database for enhanced scalability and remote monitoring.

Conclusion and Future Scope

Conclusion:

The Smart Dose System has been successfully designed and implemented to improve medication adherence through automation and intelligent control. By combining Arduino Uno, RTC (DS3231), HC-05 Bluetooth module, servo motors, buzzer, LEDs, and an LCD display, the system provides an efficient solution for timely medicine dispensing. The integration with a mobile application built using MIT App Inventor ensures user-friendly operation, enabling users to set medicine schedules, receive alerts, and track pill counts. The real-time clock ensures high accuracy in dispensing times, while the buzzer and LED system effectively remind users to take their doses. Overall, the Smart Dose project demonstrates a reliable, low-cost, and user-centered healthcare innovation that enhances patient safety and reduces the risk of missed or incorrect doses.

Future Scope:

The project can be expanded further by integrating IoT-based cloud connectivity to allow remote monitoring by doctors or caregivers. Adding features such as SMS or app notifications, AI-based dosage adjustment, and voice commands for elderly or visually impaired users can enhance usability. Future versions may also include biometric access for secure medication control and automatic refill reminders. This system has strong potential to evolve into a comprehensive smart healthcare assistant promoting healthy living.

References

1. Arduino Official Documentation – *Arduino Uno Rev3 Technical Specifications*, available at:
<https://www.arduino.cc/en/Main/ArduinoBoardUno>
2. DS3231 RTC Module Datasheet – *Maxim Integrated Products*, available at:
<https://datasheets.maximintegrated.com/en/ds/DS3231.pdf>
3. HC-05 Bluetooth Module – *Serial Port Protocol and Configuration Guide*, available at:
<https://components101.com/modules/hc-05-bluetooth-module>
4. MIT App Inventor – *Official User Guide and Tutorials*, available at:
<https://appinventor.mit.edu/>
5. Arduino Servo Library – *Control of Servo Motors with Arduino*, available at:
<https://www.arduino.cc/en/Reference/Servo>
6. LCD Display Interfacing with Arduino – *Circuit Basics Tutorial*, available at:
<https://www.circuitbasics.com/how-to-set-up-an-lcd-display-on-an-arduino/>
7. Blynk IoT Platform (for future enhancements) – *Smart Device Connectivity Guide*, available at:
<https://blynk.io/>
8. **Rao, S. & Gupta, P. (2022). *Automation in Healthcare using IoT-based Smart Devices*. International Journal of Engineering Research, Vol. 10, No. 3, pp. 55–62.**

Appendix:

```
#include <Wire.h>

#include <LiquidCrystal_I2C.h>

#include <RTClib.h>

#include <Servo.h>

// Initialize components

LiquidCrystal_I2C lcd(0x27, 16, 2); // Change LCD address if
needed

RTC_DS3231 rtc;

// Servo objects

Servo servo1, servo2, servo3, servo4;

// Pin definitions

const int BUTTON_PIN = 7;

const int BUZZER_PIN = 8;

const int SERVO1_PIN = 9;

const int SERVO2_PIN = 10;

const int SERVO3_PIN = 11;

const int SERVO4_PIN = 12

// LED pins for each slot

const int LED1_PIN = 2;

const int LED2_PIN = 3;

const int LED3_PIN = 4;

const int LED4_PIN = 5;

// Manual override button pins

const int MANUAL_BUTTON1_PIN = A0;

const int MANUAL_BUTTON2_PIN = A1;

const int MANUAL_BUTTON3_PIN = A2;

const int MANUAL_BUTTON4_PIN = A3;

// Default time slots (will be overwritten by app data)

int SLOT_START_HOURS[4] = {9, 14, 19, 21};

int SLOT_START_MINUTES[4] = {0, 0, 0, 0};

int SLOT_END_HOURS[4] = {9, 14, 19, 21};

int SLOT_END_MINUTES[4] = {3, 3, 3, 3}; // Changed to 3
minutes duration

// Tablet names and counts for each slot

String tabletNames[4] = {"", "", "", ""};

int tabletCounts[4] = {0, 0, 0, 0};

String slotNames[4] = {"Morning", "Afternoon", "Evening",
"Night"};

// State variables for each slot

enum SystemState { SLOT_OPEN, SLOT_CLOSE,
TABLET_TAKEN, SLOT_MISSED, OUTSIDE_SLOT };

SystemState currentSlotState[4] = {SLOT_CLOSE,
SLOT_CLOSE, SLOT_CLOSE, SLOT_CLOSE};

bool slotCompleted[4] = {false, false, false, false};

bool slotActiveToday[4] = {false, false, false, false};

bool slotMissedChecked[4] = {false, false, false, false};

bool ledActive[4] = {false, false, false, false};
```

```

bool slotConfigured[4] = {false, false, false, false};

// Manual override states

bool manualOverrideActive = false;

bool servoManualState[4] = {false, false, false, false};

unsigned long lastManualButtonPress[4] = {0, 0, 0, 0};

unsigned long lastButtonPress = 0;

const unsigned long DEBOUNCE_DELAY = 300;

const unsigned long MANUAL_DEBOUNCE_DELAY =
500;

int currentActiveSlot = -1;

bool dayReset = false;

// Serial communication

String inputString = "";

bool stringComplete = false;

// LCD display state management

unsigned long lastLCDUpdate = 0;

const unsigned long LCD_UPDATE_INTERVAL = 500;

String currentLine1 = "";

String currentLine2 = "";

bool displayInitialized = false;

// --- Function Prototypes ---

void showTemporaryMessage(String line1, String line2,
unsigned long duration);

void updateLCD();

```

```

void checkSerialData();

void checkManualOverrideButtons();

void resetAllSlots();

void updateLEDFromServoState();

int getCurrentActiveSlot(DateTime now);

void handleTimeSlot(DateTime now, int slotIndex);

void handleOutsideTimeSlot(DateTime now);

void checkMissedSlotsImmediately(DateTime now);

String getTimeString(DateTime now);

int getTimeRemainingInSlot(DateTime now, int slotIndex);

String truncateString(String input, int maxLength);

String getNextSlotName(DateTime now);

void displayPreviousSlotStatus(DateTime now);

unsigned long displayMessageTimeout = 0;

bool showingTemporaryMessage = false;

String tempLine1 = "";

String tempLine2 = "";

// Buzzer timing variables

unsigned long lastMinuteBuzzerTime[4] = {0, 0, 0, 0};

bool initialBuzzerSounded[4] = {false, false, false, false};

bool minuteBuzzerActive[4] = {false, false, false, false};

void setup() {

```

```
// Initialize Serial (pins 0 and 1) for HC-05
```

```
Serial.begin(9600);
```

```
inputString.reserve(200);
```

```
// Initialize pins
```

```
pinMode(BUTTON_PIN, INPUT_PULLUP);
```

```
pinMode(BUZZER_PIN, OUTPUT);
```

```
digitalWrite(BUZZER_PIN, LOW);
```

```
// Initialize LED pins
```

```
pinMode(LED1_PIN, OUTPUT);
```

```
pinMode(LED2_PIN, OUTPUT);
```

```
pinMode(LED3_PIN, OUTPUT);
```

```
pinMode(LED4_PIN, OUTPUT);
```

```
// Initialize manual button pins
```

```
pinMode(MANUAL_BUTTON1_PIN, INPUT_PULLUP);
```

```
pinMode(MANUAL_BUTTON2_PIN, INPUT_PULLUP);
```

```
pinMode(MANUAL_BUTTON3_PIN, INPUT_PULLUP);
```

```
pinMode(MANUAL_BUTTON4_PIN, INPUT_PULLUP);
```

```
// Initialize Servos
```

```
servo1.attach(SERVO1_PIN);
```

```
servo2.attach(SERVO2_PIN);
```

```
servo3.attach(SERVO3_PIN);
```

```
servo4.attach(SERVO4_PIN);
```

```
// Initialize all servos to closed position (90 degrees)
```

```
servo1.write(90);
```

```
servo2.write(90);
```

```
servo3.write(90);
```

```
servo4.write(90);
```

```
// Turn off all LEDs initially
```

```
digitalWrite(LED1_PIN, LOW);
```

```
digitalWrite(LED2_PIN, LOW);
```

```
digitalWrite(LED3_PIN, LOW);
```

```
digitalWrite(LED4_PIN, LOW);
```

```
// Initialize LCD
```

```
lcd.init();
```

```
lcd.backlight();
```

```
lcd.clear();
```

```
// Initialize RTC
```

```
if (!rtc.begin()) {
```

```
    lcd.print("RTC Not Found!");
```

```
    while (1);
```

```
}
```

```
// Set RTC time initially
```

```
if (rtc.lostPower()) {
```

```

    rtc.adjust(DateTime(F(_DATE), F(TIME_)));

}

// Reset all slots at midnight

if (now.hour() == 0 && now.minute() == 0 && !dayReset) {

    resetAllSlots();

    dayReset = true;

}

if (now.hour() != 0) {

    dayReset = false;

}

// Initialize display

currentLine1 = "System Ready";

currentLine2 = "Waiting...";

updateLCD();

displayInitialized = true;

}

// Update LED indicators based on servo state

updateLEDFromServoState();

// Check which slot is currently active

currentActiveSlot = getCurrentActiveSlot(now);

if (currentActiveSlot != -1) {

    handleTimeSlot(now, currentActiveSlot);

} else {

    handleOutsideTimeSlot(now);

}

// Check for missed slots immediately after each slot's end
time

checkMissedSlotsImmediately(now);

// Update LCD display with throttling

updateMainDisplay(now);

void loop() {

    DateTime now = rtc.now();

    // Check for serial data from app

    checkSerialData();

    // Check for manual override buttons first (highest priority)

    checkManualOverrideButtons();

    // If manual override is active, skip automatic operations

    if (manualOverrideActive) {

        return;

    }

```

```

delay(100);

}

// Calculate time remaining

int timeRemaining = getTimeRemainingInSlot(now,
currentActiveSlot);

void updateMainDisplay(DateTime now) {

// Only show OPEN if servo is actually open

// Check if we're showing a temporary message

String stateIndicator =
(currentSlotState[currentActiveSlot] == SLOT_OPEN) ?
"OPEN" : "";

if (showingTemporaryMessage) {

if (millis() > displayMessageTimeout) {

// Build second line dynamically

// Just stop showing temporary message

newLine2 =
truncateString(tabletNames[currentActiveSlot], 8) + " C:" +
String(tabletCounts[currentActiveSlot]);

showingTemporaryMessage = false;

return; // Don't clear LCD — keep message visible until
next refresh

// Append OPEN only when needed

} else {

if (stateIndicator != "") {

newLine2 += " " + stateIndicator;

}

return; // Still showing temp message, skip regular display
updates

}

}

// Append remaining time

newLine2 += " " + String(timeRemaining) + "Min";

if (millis() - lastLCDUpdate > LCD_UPDATE_INTERVAL)
{

}

else {

// ☒ Outside slot: show current time and next slot info

lastLCDUpdate = millis();

newLine1 = getTimeString(now);

newLine1 = getTimeString(now);

if (currentActiveSlot != -1) {

// ☒ During active slot

// Determine next slot name or show "No upcoming slots"

String nextSlotMsg = getNextSlotName(now);

```

```

        if (nextSlotMsg == "None") {
            newLine2 = "No upcoming slots";
        } else {
            newLine2 = "Next Slot: " + nextSlotMsg;
        }
    }

    // Update LCD only if content changed
    if (newLine1 != currentLine1 || newLine2 != currentLine2)
    {
        currentLine1 = newLine1;
        currentLine2 = newLine2;

        updateLCD();
    }
}

String getNextSlotName(DateTime now) {
    int currentTotalMinutes = now.hour() * 60 + now.minute();

    int nextSlot = -1;

    bool anyConfigured = false;

    for (int i = 0; i < 4; i++) {
        if (!slotConfigured[i]) continue;

        anyConfigured = true;

        int slotStartTotalMinutes = SLOT_START_HOURS[i] * 60
+ SLOT_START_MINUTES[i];

        if (slotStartTotalMinutes > currentTotalMinutes) {
            nextSlot = i;
            break;
        }
    }

    if (!anyConfigured) return "None";

    if (nextSlot == -1) return "None";

    // Example: "Dolo@10:30"
    String formatted = tabletNames[nextSlot] + "@";

    if (SLOT_START_HOURS[nextSlot] < 10) formatted +=
"0";

    formatted += String(SLOT_START_HOURS[nextSlot]);

    formatted += ":";

    if (SLOT_START_MINUTES[nextSlot] < 10) formatted +=
"0";

    formatted += String(SLOT_START_MINUTES[nextSlot]);

    return formatted;
}

String getTimeString(DateTime now) {
    String timeStr = "Time: ";

    if (now.hour() < 10) timeStr += "0";

    timeStr += String(now.hour());

    timeStr += ":";

    if (now.minute() < 10) timeStr += "0";

    timeStr += String(now.minute());

    timeStr += ":";
}

```

```

    if (now.second() < 10) timeStr += "0";

    timeStr += String(now.second());

    return timeStr;
}

int getTimeRemainingInSlot(DateTime now, int slotIndex) {

    int currentTotalMinutes = now.hour() * 60 + now.minute();

    int slotEndTotalMinutes = SLOT_END_HOURS[slotIndex]
* 60 + SLOT_END_MINUTES[slotIndex];

    return slotEndTotalMinutes - currentTotalMinutes;
}

String getStatusMessage(DateTime now) {

    int currentHour = now.hour();

    int currentMinute = now.minute();

    // Count configured slots

    int configuredCount = 0;

    for (int i = 0; i < 4; i++) {

        if (slotConfigured[i]) configuredCount++;

    }

    if (configuredCount == 0) {

        return "No tablets set";

    }

    // Find next upcoming slot

    int nextSlot = -1;

    unsigned long minTimeToNext = 24 * 60; // Large value

    for (int i = 0; i < 4; i++) {

        if (!slotConfigured[i]) continue;

        int slotStartTotalMinutes = SLOT_START_HOURS[i] * 60
+ SLOT_START_MINUTES[i];

        int currentTotalMinutes = currentHour * 60 +
currentMinute;

        if (slotStartTotalMinutes > currentTotalMinutes) {

            unsigned long timeToNext = slotStartTotalMinutes -
currentTotalMinutes;

            if (timeToNext < minTimeToNext) {

                minTimeToNext = timeToNext;

                nextSlot = i;

            }

        }

        if (nextSlot != -1) {

            return "Next: " + truncateString(tabletNames[nextSlot],
12);

        }

    }

    // Check if we're after last slot

    int lastSlot = -1;

```

for (int i = 3; i >= 0; i--) {	lcd.clear();
if (slotConfigured[i]) {	lcd.setCursor(0, 0);
lastSlot = i;	lcd.print(currentLine1);
break;	lcd.setCursor(0, 1);
}	lcd.print(currentLine2);
}	}
if (lastSlot != -1) {	void showTemporaryMessage(String line1, String line2,
if (currentHour > SLOT_END_HOURS[lastSlot]	unsigned long duration) {
(currentHour == SLOT_END_HOURS[lastSlot] &&	tempLine1 = truncateString(line1, 16);
currentMinute >= SLOT_END_MINUTES[lastSlot])) {	tempLine2 = truncateString(line2, 16);
return "Day Complete";	showingTemporaryMessage = true;
}	displayMessageTimeout = millis() + duration;
}	
return "Outside Time";	lcd.clear();
}	lcd.setCursor(0, 0);
	lcd.print(tempLine1);
	lcd.setCursor(0, 1);
String truncateString(String input, int maxLength) {	lcd.print(tempLine2);
if (input.length() <= maxLength) {	}
return input;	
} else {	
return input.substring(0, maxLength);	void checkSerialData() {
}	while (Serial.available()) {
}	char inChar = (char)Serial.read();
	if (inChar == '\n' inChar == '\r') {
void updateLCD() {	if (inputString.length() > 0) {

```

        stringComplete = true;

        break;

    }

} else {

    if (isPrintable(inChar)) {

        inputString += inChar;

    }

}

}

if (stringComplete) {

    inputString.trim();

    Serial.print("Raw received: ");

    Serial.print(inputString);

    Serial.println("");

    int firstUnderscore = inputString.indexOf('_');

    if (firstUnderscore != -1) {

        int secondUnderscore = inputString.indexOf('_',
firstUnderscore + 1);

        if (secondUnderscore != -1) {

            processTimeConfiguration(inputString);

        } else {

            Serial.println("ERROR: Invalid format - missing
count");

        }

    } else {

```

```

        Serial.println("ERROR: Invalid format - no
underscores");

    }

    inputString = "";

    stringComplete = false;

}

}

void processTimeConfiguration(String data) {

    data.trim();

    Serial.print("Processing: ");

    Serial.print(data);

    Serial.println("");

    int underscoreIndex1 = data.indexOf('_');

    if (underscoreIndex1 == -1) {

        Serial.println("ERROR: Invalid format - no first
underscore");

        return;

    }

    String tabletName = data.substring(0, underscoreIndex1);

    tabletName.trim();

    int underscoreIndex2 = data.indexOf('_', underscoreIndex1 +
1);

    if (underscoreIndex2 == -1) {

```

```
Serial.println("ERROR: Invalid format - missing count");

return;

}
```

```
String timeStr = data.substring(underscoreIndex1 + 1,
underscoreIndex2);

timeStr.trim();
```

```
String countStr = data.substring(underscoreIndex2 + 1);

countStr.trim();
```

```
int colonIndex = timeStr.indexOf(':');
```

```
if (colonIndex == -1) {
```

```
    Serial.println("ERROR: Invalid time format");
```

```
    return;
```

```
}
```

```
int hour = timeStr.substring(0, colonIndex).toInt();
```

```
int minute = timeStr.substring(colonIndex + 1).toInt();
```

```
int count = countStr.toInt();
```

```
if (hour < 0 || hour > 23 || minute < 0 || minute > 59) {
```

```
    Serial.println("ERROR: Invalid time values");
```

```
    return;
```

```
}
```

```
if (count <= 0) {
```

```
Serial.println("ERROR: Invalid count value");
```

```
return;
```

```
}
```

```
int slotIndex = -1;
```

```
if (hour >= 0 && hour < 11) {
```

```
    slotIndex = 0;
```

```
} else if (hour >= 11 && hour < 16) {
```

```
    slotIndex = 1;
```

```
} else if (hour >= 16 && hour < 20) {
```

```
    slotIndex = 2;
```

```
} else {
```

```
    slotIndex = 3;
```

```
}
```

```
SLOT_START_HOURS[slotIndex] = hour;
```

```
SLOT_START_MINUTES[slotIndex] = minute;
```

```
// Set end time to start time + 3 minutes
```

```
SLOT_END_HOURS[slotIndex] = hour;
```

```
SLOT_END_MINUTES[slotIndex] = minute + 3;
```

```
// Handle minute overflow
```

```
if (SLOT_END_MINUTES[slotIndex] >= 60) {
```

```
    SLOT_END_MINUTES[slotIndex] -= 60;
```

```
    SLOT_END_HOURS[slotIndex] += 1;
```

```

if (SLOT_END_HOURS[slotIndex] >= 24) {
    SLOT_END_HOURS[slotIndex] -= 24;
}

}

}

tabletNames[slotIndex] = tabletName;

tabletCounts[slotIndex] = count;

slotConfigured[slotIndex] = true;

Serial.print("CONFIRMED: ");

Serial.print(slotNames[slotIndex]);

Serial.print(" slot set for ");

Serial.print(tabletName);

Serial.print(" at ");

Serial.print(hour);

Serial.print(":");

if (minute < 10) Serial.print("0");

Serial.print(minute);

Serial.print(" to ");

Serial.print(SLOT_END_HOURS[slotIndex]);

Serial.print(":");

if (SLOT_END_MINUTES[slotIndex] < 10)
Serial.print("0");

Serial.print(SLOT_END_MINUTES[slotIndex]);

Serial.print(" Count: ");

Serial.println(count);

// Show confirmation on LCD

String confirmLine1 = slotNames[slotIndex] + " Set";

String confirmLine2 = tabletName + " C:" + String(count);

showTemporaryMessage(confirmLine1, confirmLine2,
3000);

}

void updateLEDFromServoState() {

    DateTime now = rtc.now();

    int currentTotalMinutes = now.hour() * 60 + now.minute();

    for (int i = 0; i < 4; i++) {

        bool shouldLEDBeOn = false;

        if (!slotConfigured[i]) {

            shouldLEDBeOn = false;

        } else {

            int slotStartTotalMinutes = SLOT_START_HOURS[i] *
60 + SLOT_START_MINUTES[i];

            int slotEndTotalMinutes = SLOT_END_HOURS[i] * 60 +
SLOT_END_MINUTES[i];

            // LED stays ON if:

            // Slot is currently active (time within slot)

            // Servo is open (manual or auto)

            // Tablet not yet taken

            if ((currentTotalMinutes >= slotStartTotalMinutes &&
currentTotalMinutes < slotEndTotalMinutes &&
!slotCompleted[i]) ||

```

```

    (currentSlotState[i] == SLOT_OPEN)) {

    shouldLEDBeOn = true;

} else {

    shouldLEDBeOn = false;

}

}

switch (i) {

    case 0: digitalWrite(LED1_PIN, shouldLEDBeOn ?
HIGH : LOW); break;

    case 1: digitalWrite(LED2_PIN, shouldLEDBeOn ?
HIGH : LOW); break;

    case 2: digitalWrite(LED3_PIN, shouldLEDBeOn ?
HIGH : LOW); break;

    case 3: digitalWrite(LED4_PIN, shouldLEDBeOn ?
HIGH : LOW); break;

}

    ledActive[i] = shouldLEDBeOn; // Keep internal LED
state synced

}

}

void checkManualOverrideButtons() {

    int manualButtonPins[4] = {MANUAL_BUTTON1_PIN,
MANUAL_BUTTON2_PIN, MANUAL_BUTTON3_PIN,
MANUAL_BUTTON4_PIN};

    int ledPins[4] = {LED1_PIN, LED2_PIN, LED3_PIN,
LED4_PIN};

    for (int i = 0; i < 4; i++) {

```

```

    if (digitalRead(manualButtonPins[i]) == LOW &&

        millis() - lastManualButtonPress[i] >
MANUAL_DEBOUNCE_DELAY) {

        lastManualButtonPress[i] = millis();

        manualOverrideActive = true;

        // Toggle servo state

        servoManualState[i] = !servoManualState[i];

        // Control servo movement

        if (servoManualState[i]) {

            switch (i) {

                case 0: servo1.write(180); break;

                case 1: servo2.write(180); break;

                case 2: servo3.write(180); break;

                case 3: servo4.write(180); break;

            }

            Serial.println("MANUAL: Servo " + String(i + 1) + "
opened");

        } else {

            switch (i) {

                case 0: servo1.write(90); break;

                case 1: servo2.write(90); break;

                case 2: servo3.write(90); break;

                case 3: servo4.write(90); break;

            }

        }

    }

}

```

```

        Serial.println("MANUAL: Servo " + String(i + 1) + "
        closed");

    }

```

```

    // ☒ LED mirrors servo state (ON when OPEN, OFF
    when CLOSED)

```

```

    ledActive[i] = servoManualState[i];

```

```

    digitalWrite(ledPins[i], ledActive[i] ? HIGH : LOW);

```

```

    // Feedback

```

```

    activateBuzzer(200);

```

```

    showTemporaryMessage("Manual Mode", slotNames[i] +
    " " + (servoManualState[i] ? "OPEN" : "CLOSE"), 2000);

```

```

    }

```

```

}

```

```

// Auto exit manual mode after 3 seconds of no button
activity

```

```

static unsigned long manualModeTimer = 0;

```

```

if (manualOverrideActive) {

```

```

    bool anyButtonPressed = false;

```

```

    for (int i = 0; i < 4; i++) {

```

```

        if (digitalRead(manualButtonPins[i]) == LOW) {

```

```

            anyButtonPressed = true;

```

```

            manualModeTimer = millis();

```

```

            break;

```

```

        }

```

```

    }

```

```

    if (!anyButtonPressed && millis() - manualModeTimer >
    3000) {

```

```

        manualOverrideActive = false;

```

```

        Serial.println("Exiting Manual Mode");

```

```

        showingTemporaryMessage = false; // Stop manual message

```

```

        // Let updateMainDisplay() refresh the regular screen
        automatically

```

```

    }

```

```

    } else {

```

```

        manualModeTimer = millis();

```

```

    }

```

```

}

```

```

void resetAllSlots() {

```

```

    for(int i = 0; i < 4; i++) {

```

```

        slotCompleted[i] = false;

```

```

        slotActiveToday[i] = false;

```

```

        slotMissedChecked[i] = false;

```

```

        currentSlotState[i] = SLOT_CLOSE;

```

```

        servoManualState[i] = false;

```

```

        ledActive[i] = false;

```

```

        initialBuzzerSounded[i] = false;

```

```

        minuteBuzzerActive[i] = false;

```

```

        lastMinuteBuzzerTime[i] = 0;

```

```

    }

```

```

// Close all servos

servo1.write(90);

servo2.write(90);

servo3.write(90);

servo4.write(90);

// Turn off all LEDs

digitalWrite(LED1_PIN, LOW);

digitalWrite(LED2_PIN, LOW);

digitalWrite(LED3_PIN, LOW);

digitalWrite(LED4_PIN, LOW);

showTemporaryMessage("New Day", "All slots reset",
2000);

Serial.println("All slots reset for new day");

int getCurrentActiveSlot(DateTime now) {

    int currentHour = now.hour();

    int currentMinute = now.minute();

    for (int i = 0; i < 4; i++) {

        if (!slotConfigured[i]) continue;

        int slotStartTotalMinutes = SLOT_START_HOURS[i] * 60
+ SLOT_START_MINUTES[i];

        int slotEndTotalMinutes = SLOT_END_HOURS[i] * 60 +
SLOT_END_MINUTES[i];

        int currentTotalMinutes = currentHour * 60 +
currentMinute;

        if (currentTotalMinutes >= slotStartTotalMinutes &&
currentTotalMinutes < slotEndTotalMinutes) {

            return i;

        }
    }
}

```

```

void handleTimeSlot(DateTime now, int slotIndex) {

    slotMissedChecked[slotIndex] = false

    // Sound 3-second buzzer at start of slot (only once)

    if (!initialBuzzerSounded[slotIndex]) {

        activateBuzzer(3000); // 3-second buzzer

        initialBuzzerSounded[slotIndex] = true;

        showTemporaryMessage("Time for:",
tabletNames[slotIndex], 2000);

        Serial.println("Initial 3-second buzzer for " +
tabletNames[slotIndex]);

        // Sound 2-second buzzer every minute until tablet is taken

        if (!slotCompleted[slotIndex] && now.second() == 0) {

            unsigned long currentTime = millis();

            if (currentTime - lastMinuteBuzzerTime[slotIndex] >=
60000) { // Every minute

                activateBuzzer(2000); // 2-second buzzer

                lastMinuteBuzzerTime[slotIndex] = currentTime;

                minuteBuzzerActive[slotIndex] = true;

                Serial.println("Minute reminder buzzer for " +
tabletNames[slotIndex]);

                if (digitalRead(BUTTON_PIN) == LOW && millis() -
lastButtonPress > DEBOUNCE_DELAY) {

                    lastButtonPress = millis();

                    slotActiveToday[slotIndex] = true;

                    if (currentSlotState[slotIndex] == SLOT_CLOSE &&
!slotCompleted[slotIndex]) {

                        openSlot(slotIndex);

                        showTemporaryMessage("Take your",
tabletNames[slotIndex], 2000);

                    } else if (currentSlotState[slotIndex] == SLOT_OPEN &&
!slotCompleted[slotIndex]) {

```

```

closeSlot(slotIndex);

if (tabletCounts[slotIndex] > 0) {

    tabletCounts[slotIndex]--;

}

showTemporaryMessage("Tablet Taken", "Remaining: " +
String(tabletCounts[slotIndex]), 2000);

Serial.print(tabletNames[slotIndex]);

Serial.print(" -- taken ; remaining count : ");

Serial.println(tabletCounts[slotIndex]);

// Stop minute buzzers once tablet is taken

minuteBuzzerActive[slotIndex] = false;

} else if (slotCompleted[slotIndex]) {

    Serial.print(tabletNames[slotIndex]);

    Serial.print(" already taken ; remaining count : ");

    Serial.println(tabletCounts[slotIndex]);

    showTemporaryMessage("Already Taken",
tabletNames[slotIndex], 2000);

    activateBuzzer(2000);

void openSlot(int slotIndex) {

    currentSlotState[slotIndex] = SLOT_OPEN;

    switch(slotIndex) {

        case 0: servo1.write(180); break;

        case 1: servo2.write(180); break;

        case 2: servo3.write(180); break;

        case 3: servo4.write(180); break;

    }

    Serial.print("AUTO: ");

```

```

Serial.print(slotNames[slotIndex]);

Serial.println(" slot opened");

void closeSlot(int slotIndex) {

    currentSlotState[slotIndex] = SLOT_CLOSE;

    slotCompleted[slotIndex] = true;

    switch(slotIndex) {

        case 0: servo1.write(90); break;

        case 1: servo2.write(90); break;

        case 2: servo3.write(90); break;

        case 3: servo4.write(90); break;

    }

    Serial.print("AUTO: ");


    Serial.print(slotNames[slotIndex]);

    Serial.println(" slot closed");

void handleOutsideTimeSlot(DateTime now) {

    if (digitalRead(BUTTON_PIN) == LOW && millis() -
lastButtonPress > DEBOUNCE_DELAY) {

        lastButtonPress = millis();

        displayPreviousSlotStatus(now); //  No manual flag —
handled inside function

void displayPreviousSlotStatus(DateTime now) {

    int currentHour = now.hour();

    int currentMinute = now.minute();

    int previousSlot = -1;

    for (int i = 3; i >= 0; i--) {

        if (!slotConfigured[i]) continue;

```

```

    if (currentHour > SLOT_END_HOURS[i] || (currentHour
    == SLOT_END_HOURS[i] && currentMinute >=
    SLOT_END_MINUTES[i])) {

```

```

        previousSlot = i;

```

```

    if (previousSlot == -1) {

```

```

        showTemporaryMessage("No previous", "slot today",
        3000);

```

```

        return;

```

```

    }

```

```

    String statusLine1, statusLine2;

```

```

    if (slotCompleted[previousSlot]) {

```

```

        statusLine1 = tabletNames[previousSlot] + ": Taken";

```

```

        Serial.print(tabletNames[previousSlot]);

```

```

        Serial.print(" -- taken ; remaining count : ");

```

```

        Serial.println(tabletCounts[previousSlot]);

```

```

    } else {

```

```

        statusLine1 = tabletNames[previousSlot] + ": Missed";

```

```

        Serial.print(tabletNames[previousSlot]);

```

```

        Serial.print(" -- missed ; remaining count : ");

```

```

        Serial.println(tabletCounts[previousSlot]);

```

```

        statusLine2 = "Remaining: " +
        String(tabletCounts[previousSlot]);

```

```

        showTemporaryMessage(statusLine1, statusLine2, 3000);

```

```

    void checkMissedSlotsImmediately(DateTime now) {

```

```

        int currentHour = now.hour();

```

```

        int currentMinute = now.minute();

```

```

        for (int i = 0; i < 4; i++) {

```

```

            if (!slotConfigured[i]) continue;

```

```

            if (currentHour == SLOT_END_HOURS[i] &&
            currentMinute == SLOT_END_MINUTES[i] &&
            !slotMissedChecked[i]) {

```

```

                if (!slotCompleted[i]) {

```

```

                    currentSlotState[i] = SLOT_MISSED;

```

```

                    slotMissedChecked[i] = true;

```

```

                // Close servo if it was open

```

```

                switch(i) {

```

```

                    case 0: servo1.write(90); break;

```

```

                    case 1: servo2.write(90); break;

```

```

                    case 2: servo3.write(90); break;

```

```

                    case 3: servo4.write(90); break;

```

```

                // 5-second buzzer for missed tablet

```

```

                activateBuzzer(5000);

```

```

                Serial.print(tabletNames[i]);

```

```

                Serial.print(" -- missed ; remaining count : ");

```

```

                Serial.println(tabletCounts[i]);

```

```

                showTemporaryMessage("MISSED!", tabletNames[i],
                3000);

```

```

            } else {

```

```

                slotMissedChecked[i] = true;

```

```

        void activateBuzzer(int duration) {

```

```

            digitalWrite(BUZZER_PIN, HIGH);

```

```

            delay(duration

```

****THE END****