Código del Proyecto

A continuación, se presenta el fragmento de código correspondiente al Prototipo de Sistema Asistido para el Control de Medicación en Adultos Mayores.

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <math.h>
#include <RtcDS1302.h>
#include "BluetoothSerial.h"
#include <ESP32Servo.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define OLED_SDA 21
#define OLED_SCL 22
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

#define CLK_PIN 5
#define DAT_PIN 4
#define RST_PIN 2
#define BUZZER_PIN 25
#define LED_PIN 13
#define SERVO_PIN 18
#define STATUS_LED_PIN 2

ThreeWire myWire(DAT_PIN, CLK_PIN, RST_PIN);
RtcDS1302<ThreeWire> Rtc(myWire);

BluetoothSerial SerialBT;

Servo servoMotor;

// volatile bool buttonPressed = false; // No longer needed
int currentScreen = 0;
unsigned long lastScreenChangeTime = 0; // New variable for automatic screen
switching
const unsigned long SCREEN_CHANGE_INTERVAL = 12000; // 12 seconds

int eyeWidth = 24;
int eyeHeight = 24;
int leftEyeX = (SCREEN_WIDTH / 2) - (eyeWidth + 10);
```

```
int rightEyeX = (SCREEN_WIDTH / 2) + 10;
int eyeY = 8;

int targetOffsetX = 0;
int targetOffsetY = 0;
int moveSpeed = 5;
int blinkState = 0;
int blinkDelay = 4000;

unsigned long lastBlinkTime = 0;
unsigned long moveTime = 0;
unsigned long lastExpressionChange = 0;
int expressionType = 0;

bool estadoAnteriorBluetooth = false;
String mensajeBluetooth = "";
String nombrePastilla = "";
String textoHora = ""; // Ahora almacena la hora de la alarma formateada
desde Bluetooth
bool alarmaActiva = false;
bool alarmaEnCurso = false;
int ultimaHoraAlarma = -1;
int ultimoMinutoAlarma = -1;

unsigned long ultimaAccionAlarma = 0;
int repeticionesAlarma = 0;
bool esperandoProximoCiclo = false;

// Bitmap del icono de campana (8x8 píxeles)
const unsigned char PROGMEM bell_icon[] =
{
  0x04, 0x0E, 0x0E, 0x0E, 0x1F, 0x00, 0x04, 0x00
};

void drawHappyEye(int x, int y) {
  display.fillRoundRect(x, y, eyeWidth, eyeHeight, 4, WHITE);
}

void drawLaughingEyes(int x, int y) {
  display.drawLine(x, y + eyeHeight / 2, x + eyeWidth / 2, y + eyeHeight / 2
+ 3, WHITE);
  display.drawLine(x + eyeWidth / 2, y + eyeHeight / 2 + 3, x + eyeWidth, y
+ eyeHeight / 2, WHITE);
}
```

```
void drawEyebrows(int leftX, int rightX, int y) {
  display.drawLine(leftX, y - 4, leftX + eyeWidth, y - 6, WHITE);
  display.drawLine(rightX, y - 6, rightX + eyeWidth, y - 4, WHITE);
}

void drawSmilingMouth(int offsetX, int offsetY) {
  int centerX = SCREEN_WIDTH / 2 + offsetX / 2;
  int baseY = 52 + offsetY / 4;

  for (int x = -12; x <= 12; x++) {
    int y = (x * x) / 18;
    display.drawPixel(centerX + x, baseY - y, WHITE);
  }
}

void runFaceAnimation() {
  unsigned long currentTime = millis();

  if (currentTime - lastExpressionChange > 5000) {
    expressionType = random(0, 3);
    lastExpressionChange = currentTime;
  }

  if (blinkState == 0) {
    if (currentTime - lastBlinkTime > blinkDelay) {
      blinkState = 1;
      lastBlinkTime = currentTime;
    }
  } else {
    if (currentTime - lastBlinkTime > 150) {
      blinkState = 0;
      lastBlinkTime = currentTime;
      blinkDelay = random(3000, 7000);
    }
  }

  if (blinkState == 0) {
    if (currentTime - moveTime > random(800, 1500)) {
      int m = random(0, 8);
      switch (m) {
        case 0:
          targetOffsetX = -5;
          targetOffsetY = 0;
          break;
        case 1:
```

```
        targetOffsetX = 5;
        targetOffsetY = 0;
        break;
      case 2:
        targetOffsetX = -5;
        targetOffsetY = -4;
        break;
      case 3:
        targetOffsetX = 5;
        targetOffsetY = -4;
        break;
      case 4:
        targetOffsetX = -5;
        targetOffsetY = 4;
        break;
      case 5:
        targetOffsetX = 5;
        targetOffsetY = 4;
        break;
      default:
        targetOffsetX = 0;
        targetOffsetY = 0;
        break;
    }
    moveTime = currentTime;
  }
}

static int offsetX = 0;
static int offsetY = 0;
offsetX += (targetOffsetX - offsetX) / moveSpeed;
offsetY += (targetOffsetY - offsetY) / moveSpeed;

display.clearDisplay();

if (blinkState == 0) {
  if (expressionType == 2) {
    drawLaughingEyes(leftEyeX + offsetX, eyeY + offsetY);
    drawLaughingEyes(rightEyeX + offsetX, eyeY + offsetY);
  } else {
    drawHappyEye(leftEyeX + offsetX, eyeY + offsetY);
    drawHappyEye(rightEyeX + offsetX, eyeY + offsetY);
  }
  drawEyebrows(leftEyeX + offsetX, rightEyeX + offsetX, eyeY + offsetY);
} else {
```

```
    display.fillRect(leftEyeX + offsetX, eyeY + offsetY + eyeHeight / 2 - 2,
eyeWidth, 4, WHITE);
    display.fillRect(rightEyeX + offsetX, eyeY + offsetY + eyeHeight / 2 -
2, eyeWidth, 4, WHITE);
  }

  drawSmilingMouth(offsetX, offsetY);
}

void moverServoPorComp(int comp) {
  int angulo = 0;
  if (comp == 1)
    angulo = 34;
  else if (comp == 2)
    angulo = 96;
  else if (comp == 3)
    angulo = 168;
  else
    angulo = 0;
  servoMotor.write(angulo);
}

void mostrarPastilla() {
  display.clearDisplay();
  display.setTextColor(WHITE);

  // La función mostrarPastilla se encarga de mostrar el nombre de la
pastilla
  // centrado y más pequeño cuando la alarma está activa.
  // CAMBIO AQUÍ: Tamaño de fuente cambiado a 1 para el nombre de la
pastilla durante la alarma
  display.setTextSize(1);
  String nombreMayuscula = nombrePastilla; // Crear una copia para manipular
  nombreMayuscula.toUpperCase(); // Convertir a mayúsculas
  int16_t x1, y1;
  uint16_t w1, h1;
  display.getTextBounds(nombreMayuscula, 0, 0, &x1, &y1, &w1, &h1);
  display.setCursor((SCREEN_WIDTH - w1) / 2, (SCREEN_HEIGHT - h1) / 2); //
Centrado
  display.println(nombreMayuscula);
}

void iniciarAlarma() {
  alarmaEnCurso = true;
  repeticionesAlarma = 0;
```

```
  ultimaAccionAlarma = millis();
  esperandoProximoCiclo = false;
  digitalWrite(LED_PIN, HIGH);
  // Llamar a mostrarPastilla para mostrar el nombre de la pastilla al
iniciar la alarma
  mostrarPastilla();
}

void procesarAlarma() {
  unsigned long tiempoActual = millis();
  if (alarmaEnCurso) {
    if (!esperandoProximoCiclo) {
      for (int i = 0; i < 3; i++) {
        digitalWrite(BUZZER_PIN, HIGH);
        delay(100);
        digitalWrite(BUZZER_PIN, LOW);
        delay(100);
      }
      esperandoProximoCiclo = true;
      ultimaAccionAlarma = tiempoActual;
      repeticionesAlarma++;
    } else {
      if (tiempoActual - ultimaAccionAlarma >= 3000) {
        esperandoProximoCiclo = false;
      }
    }

    if (repeticionesAlarma >= 5) {
      alarmaEnCurso = false;
      digitalWrite(LED_PIN, LOW);
      servoMotor.write(0);
      Serial.println("Alarma finalizada.");
    }
    // Mostrar continuamente el nombre de la pastilla mientras la alarma
esté activa
    mostrarPastilla();
  }
}

void dibujarBluetoothIcono(bool conectado) {
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  int x = SCREEN_WIDTH - 6;
  int y = 0;
  display.setCursor(x, y);
```

```
  display.print(conectado ? "V" : "X");
}

String obtenerHora12(int hour) {
  if (hour == 0)
    hour = 12;
  else if (hour > 12)
    hour -= 12;
  return String(hour);
}

String obtenerAmPm(int hour) {
  return (hour < 12) ? "AM" : "PM";
}

bool parsearMensajeBluetooth(const String& mensaje, String& horaTexto,
String& pastilla) {
  int h, m, comp;
  char nombre[30];

  int num_parsed = sscanf(mensaje.c_str(), "%d:%d:%d:%29[^\n]", &h, &m,
&comp, nombre);
  Serial.print("Debug Parseo: ");
  Serial.print("Mensaje original: '");
  Serial.print(mensaje);
  Serial.print("'");
  Serial.print(", Elementos parseados: ");
  Serial.print(num_parsed);
  if (num_parsed == 4) {
    pastilla = String(nombre);

    int hora12 = h;
    String ampm = obtenerAmPm(h);
    if (hora12 == 0)
      hora12 = 12;
    else if (hora12 > 12)
      hora12 -= 12;

    char bufferHora[16];
    snprintf(bufferHora, sizeof(bufferHora), "%d:%02d %s", hora12, m,
ampm.c_str());
    horaTexto = String(bufferHora); // Almacenar la hora de la alarma
formateada aquí

    Serial.print(", H: ");
```

```
    Serial.print(h);
    Serial.print(", M: ");
    Serial.print(m);
    Serial.print(", Comp: ");
    Serial.print(comp);
    Serial.print(", Nombre: ");
    Serial.println(nombre);
    return true;
  }
  Serial.println(", Parseo Fallido.");
  return false;
}

bool coincidenHoras(int rtcHora, int rtcMinuto, const String& mensaje) {
  String horaTextoTemp; // Variable temporal, no la global textoHora
  String pastillaTemp;
  Serial.print("Debug Coincidir Horas: ");
  Serial.print("RTC ");
  Serial.print(rtcHora);
  Serial.print(":");
  Serial.print(rtcMinuto);
  Serial.print(", Mensaje: '");
  Serial.print(mensaje);
  Serial.print("'");

  if (parsearMensajeBluetooth(mensaje, horaTextoTemp, pastillaTemp)) {
    nombrePastilla = pastillaTemp; // Actualizar el nombre global de la
pastilla
    textoHora = horaTextoTemp;      // Actualizar la hora global de la
alarma para mostrar
    int h_msg, m_msg, comp_msg;
    sscanf(mensaje.c_str(), "%d:%d:%d:", &h_msg, &m_msg, &comp_msg);

    Serial.print(", Alarma MSG ");
    Serial.print(h_msg);
    Serial.print(":");
    Serial.print(m_msg);
    Serial.print(", Resultado: ");
    Serial.println((rtcHora == h_msg && rtcMinuto == m_msg) ? "TRUE" :
"FALSE");
    return (rtcHora == h_msg && rtcMinuto == m_msg);
  }
  Serial.println(", No se pudo parsear el mensaje para coincidencia.");
  return false;
}
```

```
void mostrarTextoCentrado(String texto, int yInicial, int textSize = 1) {
  int16_t x1, y1;
  uint16_t w, h;

  display.setTextSize(textSize);
  display.getTextBounds(texto, 0, 0, &x1, &y1, &w, &h);
  // Asegurarse de que el texto quepa en el ancho de la pantalla, si no,
reducir el tamaño
  while (w > SCREEN_WIDTH && textSize > 0) {
    textSize--;
    display.setTextSize(textSize);
    display.getTextBounds(texto, 0, 0, &x1, &y1, &w, &h);
  }

  display.setCursor((SCREEN_WIDTH - w) / 2, yInicial);
  display.println(texto);
}

void runClockScreen() {
  RtcDateTime now = Rtc.GetDateTime();

  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);

  // Fecha en la parte superior izquierda
  char fecha[17];
  snprintf(fecha, sizeof(fecha), "%02u/%02u/%04u", now.Day(), now.Month(),
now.Year());
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.println(fecha);

  // Icono de Bluetooth en la parte superior derecha
  bool conectado = SerialBT.hasClient();
  dibujarBluetoothIcono(conectado);
  digitalWrite(STATUS_LED_PIN, conectado ? HIGH : LOW);

  // Hora de la alarma con icono de campana (si hay un mensaje Bluetooth
presente)
  if (mensajeBluetooth.length() > 0) {
    display.drawBitmap(65, 0, bell_icon, 8, 8, WHITE); // Icono de campana
    display.setTextSize(1); // Mismo tamaño que la fecha
    display.setCursor(75, 0); // Posición al lado del icono de campana
    display.println(textoHora); // Usar la hora de la alarma pre-formateada
```

```
  }

  // Visualización grande de la hora en tiempo real (RTC)
  String ampm = obtenerAmPm(now.Hour());
  int displayHour = now.Hour();
  if (displayHour == 0) displayHour = 12;
  else if (displayHour > 12) displayHour -= 12;

  char bigTime[10];
  snprintf(bigTime, sizeof(bigTime), "%d:%02u", displayHour, now.Minute());

  display.setTextSize(3); // Fuente más grande para la hora principal
  int16_t x1_time, y1_time;
  uint16_t w_time, h_time;
  display.getTextBounds(bigTime, 0, 0, &x1_time, &y1_time, &w_time,
&h_time);

  int timeX = (SCREEN_WIDTH - w_time) / 2;
  int timeY = SCREEN_HEIGHT / 2 - h_time / 2 - 5;

  display.setCursor(timeX, timeY);
  display.println(bigTime);

  // AM/PM en fuente más pequeña al lado de la hora
  display.setTextSize(1);
  int16_t x1_ampm, y1_ampm;
  uint16_t w_ampm, h_ampm;
  display.getTextBounds(ampm, 0, 0, &x1_ampm, &y1_ampm, &w_ampm, &h_ampm);

  int ampmX = timeX + w_time + 2;
  int ampmY = timeY + h_time - h_ampm - 2;

  display.setCursor(ampmX, ampmY);
  display.println(ampm);

  // Nombre de la pastilla debajo de la hora del RTC (más pequeño y
centrado)
  if (nombrePastilla.length() > 0) {
    display.setTextColor(WHITE);
    String pillNameUpper = nombrePastilla; // Crear una copia local
    pillNameUpper.toUpperCase();            // Convertir a mayúsculas
    // Posición Y ajustada para estar debajo de la visualización principal
de la hora, y tamaño de texto más pequeño (1)
    mostrarTextoCentrado(pillNameUpper, timeY + h_time + 2, 1); // Tamaño de
texto más pequeño (1)
```

```
  }
}

void handleBluetoothConnectionSound() {
  bool conectado = SerialBT.hasClient();
  if (conectado != estadoAnteriorBluetooth) {
    digitalWrite(BUZZER_PIN, HIGH);
    delay(100);
    digitalWrite(BUZZER_PIN, LOW);
    delay(100);
    if (conectado) {
      digitalWrite(BUZZER_PIN, HIGH);
      delay(100);
      digitalWrite(BUZZER_PIN, LOW);
      Serial.println("Bluetooth CONECTADO.");
    } else {
      Serial.println("Bluetooth DESCONECTADO.");
    }
    estadoAnteriorBluetooth = conectado;
  }
}

void handleBluetoothMessages() {
  if (SerialBT.available()) {
    mensajeBluetooth = SerialBT.readStringUntil('\n');
    mensajeBluetooth.trim();
    Serial.print("Mensaje BT recibido: '");
    Serial.print(mensajeBluetooth);
    Serial.println("'");
    alarmaActiva = false;
    Serial.println("alarmaActiva reseteada a FALSE tras nuevo mensaje BT.");
  }
}

void checkAndTriggerAlarm() {
  RtcDateTime now = Rtc.GetDateTime();

  if (mensajeBluetooth.length() > 0 && !alarmaEnCurso) {
    if (coincidenHoras(now.Hour(), now.Minute(), mensajeBluetooth)) {
      if (!alarmaActiva || now.Hour() != ultimaHoraAlarma || now.Minute() !=
ultimoMinutoAlarma) {
        Serial.println("¡Coincidencia de alarma! Iniciando...");
        iniciarAlarma();

        int h, m, comp;
```

```
        sscanf(mensajeBluetooth.c_str(), "%d:%d:%d:", &h, &m, &comp);
        moverServoPorComp(comp);
        Serial.print("Servo movido a compartimento: ");
        Serial.println(comp);

        alarmaActiva = true;
        ultimaHoraAlarma = now.Hour();
        ultimoMinutoAlarma = now.Minute();
      } else {
        Serial.println("Alarma ya activa o ya sonó este minuto. Esperando el
siguiente.");
      }
    } else {
      if (alarmaActiva) {
        Serial.println("Horas no coinciden. Reseteando alarmaActiva a
FALSE.");
        alarmaActiva = false;
      }
    }
  }
}


void setup() {
  Serial.begin(115200);
  Wire.begin(OLED_SDA, OLED_SCL);

  SerialBT.begin("SaludTime");
  Serial.println("Bluetooth listo para emparejar.");

  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println("¡ERROR: No se detectó pantalla OLED!");
    digitalWrite(STATUS_LED_PIN, HIGH);
    while (true);
  }
  display.clearDisplay();
  display.display();
  Serial.println("Pantalla OLED inicializada.");

  Rtc.Begin();
  RtcDateTime compiled = RtcDateTime(__DATE__, __TIME__);

  char bufferRTC[20];
  snprintf(bufferRTC, sizeof(bufferRTC), "%02u/%02u/%04u %02u:%02u:%02u",
           compiled.Day(), compiled.Month(), compiled.Year(),
```

```
            compiled.Hour(), compiled.Minute(), compiled.Second());
  Serial.print("Hora de compilacion: ");
  Serial.println(bufferRTC);

  if (!Rtc.IsDateTimeValid()) {
    Serial.println("RTC no válido, estableciendo hora de compilacion.");
    Rtc.SetDateTime(compiled);
  }
  if (Rtc.GetIsWriteProtected()) {
    Serial.println("RTC con proteccion de escritura, deshabilitando.");
    Rtc.SetIsWriteProtected(false);
  }
  if (!Rtc.GetIsRunning()) {
    Serial.println("RTC no esta corriendo, iniciandolo.");
    Rtc.SetIsRunning(true);
  }
  if (Rtc.GetDateTime() < compiled) {
    Serial.println("RTC con hora antigua, actualizando a la de
compilacion.");
    Rtc.SetDateTime(compiled);
  }

  RtcDateTime nowRTC = Rtc.GetDateTime();
  snprintf(bufferRTC, sizeof(bufferRTC), "%02u/%02u/%04u %02u:%02u:%02u",
            nowRTC.Day(), nowRTC.Month(), nowRTC.Year(),
            nowRTC.Hour(), nowRTC.Minute(), nowRTC.Second());
  Serial.print("Hora actual del RTC: ");
  Serial.println(bufferRTC);

  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(LED_PIN, OUTPUT);
  pinMode(STATUS_LED_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, LOW);
  digitalWrite(LED_PIN, LOW);
  digitalWrite(STATUS_LED_PIN, LOW);
  Serial.println("Pines configurados. Probando Zumbador y LED...");
  digitalWrite(LED_PIN, HIGH);
  for (int i = 0; i < 3; i++) {
    digitalWrite(BUZZER_PIN, HIGH);
    delay(100);
    digitalWrite(BUZZER_PIN, LOW);
    delay(100);
  }
  digitalWrite(LED_PIN, LOW);
  Serial.println("Prueba de Zumbador y LED finalizada.");
```

```
  servoMotor.attach(SERVO_PIN);
  servoMotor.write(0);
  Serial.println("Servomotor inicializado.");

  lastScreenChangeTime = millis(); // Initialize the screen change timer
}

void loop() {
  unsigned long currentTime = millis();

  // Automatic screen switching logic
  if (!alarmaEnCurso && (currentTime - lastScreenChangeTime >=
SCREEN_CHANGE_INTERVAL)) {
    currentScreen = (currentScreen + 1) % 2;
    lastScreenChangeTime = currentTime;
    display.clearDisplay();
    Serial.print("Cambio de pantalla automatico a: ");
    Serial.println(currentScreen == 0 ? "Cara" : "Reloj");
  }

  handleBluetoothConnectionSound();
  handleBluetoothMessages();
  checkAndTriggerAlarm();
  procesarAlarma();

  // Si la alarma está sonando activamente, muestra el nombre de la pastilla
centrado y más pequeño
  if (alarmaEnCurso) {
    mostrarPastilla();
  } else { // De lo contrario, ejecuta las animaciones/pantallas normales
    if (currentScreen == 0) {
      runFaceAnimation();
    } else {
      runClockScreen(); // Esto ahora mostrará la hora de la alarma y el
nombre de la pastilla si están disponibles
    }
  }

  display.display();

  static unsigned long lastDebugPrintTime = 0;
  if (currentTime - lastDebugPrintTime >= 1000) {
    RtcDateTime now = Rtc.GetDateTime();
    char bufferRTC_loop[20];
```

```
    snprintf(bufferRTC_loop, sizeof(bufferRTC_loop),
"%02u/%02u/%04u %02u:%02u:%02u",
             now.Day(), now.Month(), now.Year(),
             now.Hour(), now.Minute(), now.Second());
    Serial.print("RTC: ");
    Serial.print(bufferRTC_loop);
    Serial.print(" - Mensaje BT: '");
    Serial.print(mensajeBluetooth);
    Serial.print("' - Alarma Activa (Bandera): ");
    Serial.print(alarmaActiva ? "TRUE" : "FALSE");
    Serial.print(" - Alarma En Curso (Sonando): ");
    Serial.print(alarmaEnCurso ? "TRUE" : "FALSE");
    Serial.print(" - Pantalla Actual: ");
    Serial.println(currentScreen == 0 ? "Cara" : "Reloj");
    lastDebugPrintTime = currentTime;
  }
}
```

TABLA DE COMPONENTES

Componente	Descripción	Cantidad	Notas
ESP32	Microcontrolador principal con conectividad Bluetooth integrada	1	Ejecuta la lógica del sistema
Pantalla OLED SSD1306 (128x64, I2C)	Display gráfico para mostrar hora, alarmas, íconos y animaciones	1	Usa pines SDA (21) y SCL (22)
RTC DS1302	Módulo de reloj en tiempo real para mantener la hora actual	1	Conexión por 3 hilos (CLK, DAT, RST)
Buzzer	Emite señales acústicas cuando la alarma está activa	1	Conectado al pin 25
LED indicador	Indica visualmente el estado de la alarma	1	Conectado al pin 13
LED de estado Bluetooth	Señala el estado de conexión Bluetooth	1	Conectado al pin 2

Componente	Descripción	Cantidad	Notas
Servomotor SG90 / similar	Mueve el compartimento de las pastillas al activarse la alarma	1	Conectado al pin 18