1) Arduino / ESP32 sketch — distance + DFPlayer + Firebase

```
/* distance_dfplayer_firebase.ino
  Sanitized: all personal credentials replaced with placeholders.
*/

#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Arduino.h>
#include <DFRobotDFPlayerMini.h>
#include <WiFi.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <FirebaseESP32.h>
#include <esp_now.h>
#include <TimeLib.h>

// I2C LCD configuration
LiquidCrystal_I2C lcd(0x27, 20, 4); // Address 0x27

// Ultrasonic sensor pins
#define TRIG_PIN 4
#define ECHO_PIN 2

// Push buttons for input
#define BUTTON_HASH 12 // Pin for YES button
#define BUTTON_STAR 13 // Pin for NO button
```

```
// Audio player serial configuration
HardwareSerial mySerial(1);
DFRobotDFPlayerMini myDFPlayer;

// Constants for distance detection
const unsigned long interval = 15000; // 15 seconds in milliseconds
unsigned long startMillis;
bool objectDetected = false;
bool isPlaying = false;
bool measureDistance = false;
int previousLaserState = -1; // To store previous laser state

// WiFi and NTP client configuration
const char* ssid = "PLACEHOLDER_YOUR_SSID";
const char* password = "PLACEHOLDER_YOUR_WIFI_PASSWORD";

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org", 19800, 60000); // Offset for Sri Lanka (UTC +5:30)

/// Firebase placeholders
#define FIREBASE_HOST "PLACEHOLDER_YOUR_FIREBASE_HOST"
#define FIREBASE_API_KEY "PLACEHOLDER_YOUR_FIREBASE_API_KEY"

FirebaseData firebaseData;
FirebaseConfig firebaseConfig;
FirebaseAuth firebaseAuth;
int recordCounter = 0;
```

```
void setup() {
  lcd.init();
  lcd.backlight();

  lcd.print("PRACTICE MAKES SKILL");
  lcd.setCursor(0, 1);
  lcd.print(" ");
  lcd.setCursor(0, 2);
  delay(10000);

  // Initialize serial communication
  Serial.begin(115200);

  // Configure ultrasonic sensor pins
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  // Configure button pins
  pinMode(BUTTON_HASH, INPUT_PULLUP);
  pinMode(BUTTON_STAR, INPUT_PULLUP);

  // Initialize the DFPlayer Mini
  mySerial.begin(9600, SERIAL_8N1, 18, 19);
  Serial.println("Initializing DFPlayer Mini...");

  if (!myDFPlayer.begin(mySerial)) {
    Serial.println("Unable to begin:");
    Serial.println("1. Please recheck the connection!");
    Serial.println("2. Please insert the SD card!");
```

```
  while (true);
}

Serial.println(F("DFPlayer Mini online."));
myDFPlayer.setTimeOut(500);
myDFPlayer.volume(30);

startMillis = millis();

// Connect to Wi-Fi
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Connecting to WiFi...");
}
Serial.println("Connected to WiFi");

// Initialize NTP client
timeClient.begin();
timeClient.update();

// Initialize Firebase
firebaseConfig.host = FIREBASE_HOST;
firebaseConfig.api_key = FIREBASE_API_KEY;
firebaseAuth.user.email = "PLACEHOLDER_your@example.com";
firebaseAuth.user.password = "PLACEHOLDER_your_password";

Firebase.begin(&firebaseConfig, &firebaseAuth);
Firebase.reconnectWiFi(true);
```

```
  // Initialize the record counter if it doesn't exist
  if (!Firebase.getInt(firebaseData, "/runner/recordCounter")) {
    Firebase.setInt(firebaseData, "/runner/recordCounter", 0);
  } else {
    recordCounter = firebaseData.intData();
  }
}

void loop() {
  // Check laser state from Firebase
  if (!Firebase.getInt(firebaseData, "/laserstate")) {
    Serial.print("Error getting laserstate: ");
    Serial.println(firebaseData.errorReason());
  } else {
    int laserState = firebaseData.intData();
    if (laserState != previousLaserState) {
      previousLaserState = laserState;
      if (laserState == 1) {
        clearRow(2);
        lcd.print("Are You Ready?");
        lcd.setCursor(0, 3);
      } else {
        clearRow(2);
        lcd.print("Laser Not Detected");
        lcd.setCursor(0, 3);
      }
    }
  }
```

```
if (previousLaserState == 1 && !measureDistance) {
  if (digitalRead(BUTTON_HASH) == LOW) {
    measureDistance = true; // Start measuring distance
    lcd.clear();
    lcd.print("Place on Position,");
    lcd.setCursor(0, 1);
    lcd.print("Command will Play");
    lcd.setCursor(0, 2);
    lcd.print("Soon");
    lcd.setCursor(0, 3);
    delay(10000);
    lcd.clear();
  } else if (digitalRead(BUTTON_STAR) == LOW) {
    measureDistance = false; // Stop measuring distance
    Serial.print("Answer is NO");
    lcd.clear();
    lcd.print("TAKE TIME & BE READY");
    lcd.setCursor(0, 1);
    delay(10000);
    lcd.print("");
    lcd.setCursor(0, 2);
    lcd.print("Are You Ready?");
    lcd.setCursor(0, 3);
  }
}

if (measureDistance && !isPlaying) {
  long duration, distance;
```

```
unsigned long currentMillis = millis();

// Send a pulse to trigger the ultrasonic sensor
digitalWrite(TRIG_PIN, LOW);
delayMicroseconds(2);
digitalWrite(TRIG_PIN, HIGH);
delayMicroseconds(10);
digitalWrite(TRIG_PIN, LOW);

// Read the duration of the echo
duration = pulseIn(ECHO_PIN, HIGH);
distance = duration * 0.034 / 2; // Calculate the distance in cm

Serial.print("Distance: ");
Serial.print(distance);
Serial.println(" cm");

if (distance <= 200) {
  if (!objectDetected) {
    objectDetected = true;
    startMillis = currentMillis;
  }

  if (currentMillis - startMillis >= interval) {
    if (!isPlaying) {
      myDFPlayer.play(1);
      isPlaying = true;

      // Get current date and time
```

```
time_t rawTime = timeClient.getEpochTime();
struct tm* timeInfo = localtime(&rawTime);

char currentDate[11];
char currentTime[30];
sprintf(currentDate, "%04d-%02d-%02d",
      timeInfo->tm_year + 1900, timeInfo->tm_mon + 1, timeInfo->tm_mday);
sprintf(currentTime, "%04d-%02d-%02d %02d:%02d:%02d:%03d",
      timeInfo->tm_year + 1900, timeInfo->tm_mon + 1, timeInfo->tm_mday,
      timeInfo->tm_hour, timeInfo->tm_min, timeInfo->tm_sec,
      millis() % 1000);

// Add small offset then store to Firebase
time_t newRawTime = rawTime + 9;
int newMillis = (millis() % 1000) + 30;
if (newMillis >= 1000) {
 newRawTime += 1;
 newMillis -= 1000;
}
struct tm* newTimeInfo = localtime(&newRawTime);
char newTime[30];
sprintf(newTime, "%02d:%02d:%02d:%03d",
      newTimeInfo->tm_hour, newTimeInfo->tm_min, newTimeInfo->tm_sec,
      newMillis);

Serial.print("Current date: ");
Serial.println(currentDate);
Serial.print("Current time: ");
Serial.println(currentTime);
```

```
    Serial.print("New time: ");
    Serial.println(newTime);

    // Increment and store record counter
    recordCounter++;
    if (!Firebase.setInt(firebaseData, "/runner/recordCounter", recordCounter)) {
      Serial.print("Error setting recordCounter: ");
      Serial.println(firebaseData.errorReason());
    }

    String recordPath = "/runner/" + String(recordCounter);
    if (Firebase.setString(firebaseData, recordPath + "/Date", currentDate) &&
        Firebase.setString(firebaseData, recordPath + "/Starttime", newTime)) {
      Serial.println("Date and time recorded successfully.");
    } else {
      Serial.print("Error recording date and time: ");
      Serial.println(firebaseData.errorReason());
    }

    delay(30000);
    myDFPlayer.stop();

    // Optionally restart
    ESP.restart();
   }
  }
 } else {
  objectDetected = false;
  isPlaying = false;
```

```
   }
  }
}

void clearRow(int row) {
  lcd.setCursor(0, row);
  for (int i = 0; i < 20; i++) {
    lcd.print(" ");
  }
  lcd.setCursor(0, row);
}
```

2) Button-timing Firebase sketch

```
/* button_timing_firebase.ino
   Button press timing + Firebase write. Sanitized credentials.
*/

#include <WiFi.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <FirebaseESP32.h>

const char* ssid = "PLACEHOLDER_YOUR_SSID";
const char* password = "PLACEHOLDER_YOUR_WIFI_PASSWORD";
```

```
#define FIREBASE_HOST "PLACEHOLDER_YOUR_FIREBASE_HOST"
#define FIREBASE_API_KEY "PLACEHOLDER_YOUR_FIREBASE_API_KEY"
#define USER_EMAIL "PLACEHOLDER_your@example.com"
#define USER_PASSWORD "PLACEHOLDER_your_password"

FirebaseData firebaseData;
FirebaseAuth firebaseAuth;
FirebaseConfig firebaseConfig;

const int buttonPin = 34;
const unsigned long interval = 7000; // 7 seconds
unsigned long starttime = 0;
bool timing = false;

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org", 19800, 60000);

void setup() {
 Serial.begin(115200);
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Connecting to WiFi...");
 }
 Serial.println("Connected to WiFi");

 timeClient.begin();
```

```
  pinMode(buttonPin, INPUT);

  firebaseConfig.host = FIREBASE_HOST;
  firebaseConfig.api_key = FIREBASE_API_KEY;
  firebaseAuth.user.email = USER_EMAIL;
  firebaseAuth.user.password = USER_PASSWORD;

  Firebase.begin(&firebaseConfig, &firebaseAuth);
  Firebase.reconnectWiFi(true);
}

void loop() {
  int buttonState = digitalRead(buttonPin);
  if (buttonState == HIGH) {
    if (!timing) {
      starttime = millis();
      timing = true;
    }
  } else {
    if (timing) {
      if (millis() - starttime >= interval) {
        timeClient.update();
        String formattedTime = timeClient.getFormattedTime();
        unsigned long ms = millis() % 1000;
        char timeWithMillis[25];
        snprintf(timeWithMillis, sizeof(timeWithMillis), "%s:%03lu", formattedTime.c_str(), ms);

        Serial.println("Button released at: " + String(timeWithMillis));
```

```
    if (Firebase.getInt(firebaseData, "/runner/limitCounter")) {
      int limitCounter = firebaseData.intData();
      Serial.print("Fetched limitCounter: ");
      Serial.println(limitCounter);

      String termPath = "/runner";
      String pathWithCounter = termPath + "/" + String(limitCounter) + "/Rtime";

      if (Firebase.setString(firebaseData, pathWithCounter, timeWithMillis)) {
        Serial.println("Time stored successfully.");
        limitCounter++;
        if (Firebase.setInt(firebaseData, termPath + "/limitCounter", limitCounter)) {
          Serial.println("Counter updated successfully.");
        } else {
          Serial.print("Error updating counter: ");
          Serial.println(firebaseData.errorReason());
        }
      } else {
        Serial.print("Error storing time: ");
        Serial.println(firebaseData.errorReason());
      }
    } else {
      Serial.print("Error getting limitCounter: ");
      Serial.println(firebaseData.errorReason());
    }
  }
  timing = false;
 }
}
```

```
}
```

3) ESP32-CAM + Firebase upload sketch

```
/* esp32cam_upload.ino
  ESP32-CAM capture + save to SD + upload (placeholder settings)
*/

#include "esp_camera.h"
#include "FS.h"
#include "SD_MMC.h"
#include "WiFi.h"
#include "HTTPClient.h"
#include "base64.h"
#include "time.h"
#include <Preferences.h>
#include <esp_now.h>
#include <esp_wifi.h>

#define CAMERA_MODEL_AI_THINKER
#include "camera_pins.h"

const char* ssid = "PLACEHOLDER_YOUR_SSID";
const char* password = "PLACEHOLDER_YOUR_WIFI_PASSWORD";

const char* firebaseHost = "PLACEHOLDER_YOUR_FIREBASE_HOST";
const char* firebaseAuth = "PLACEHOLDER_YOUR_FIREBASE_AUTH_TOKEN";
```

```
const char* firebaseStorageBucket = "PLACEHOLDER_YOUR_FIREBASE_STORAGE_BUCKET";

const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = 5 * 3600;
const int daylightOffset_sec = 30 * 60;

const int numImagesPerAttempt = 10;
int folderNumber = 0;
Preferences preferences;
bool startCapture = false;

void startCamera() {
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
```

```
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;

  if (psramFound()) {
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }

  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x\n", err);
    return;
  }
}

void onDataRecv(const esp_now_recv_info *recvInfo, const uint8_t *incomingData, int len) {
  Serial.println("Signal received to start capturing images.");
  startCapture = true;
}

void setup() {
  Serial.begin(115200);
```

```
WiFi.mode(WIFI_STA);
WiFi.disconnect();
delay(100);

if (esp_now_init() != ESP_OK) {
  Serial.println("Error initializing ESP-NOW");
  return;
}
esp_now_register_recv_cb(onDataRecv);

if (!SD_MMC.begin()) {
  Serial.println("Card Mount Failed");
  return;
}
uint8_t cardType = SD_MMC.cardType();
if (cardType == CARD_NONE) {
  Serial.println("No SD card attached");
  return;
}

WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Connecting to WiFi...");
}
Serial.println("Connected to WiFi");

startCamera();
configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
```

```
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    Serial.println("Failed to obtain time");
  } else {
    Serial.println("Time obtained successfully");
  }

  preferences.begin("camera", false);
  folderNumber = preferences.getInt("folderNumber", 0);
  preferences.end();

  Serial.println("Setup complete. Waiting for signal to capture images...");
}

void loop() {
  if (startCapture) {
    startCapture = false;
    String folderName = getFolderName(folderNumber);
    fs::FS &fs = SD_MMC;
    fs.mkdir("/" + folderName);
    for (int i = 0; i < numImagesPerAttempt; i++) {
      captureAndSaveImage(folderName, i);
      delay(10);
    }
    // uploadFolderToFirebase(folderName); // Implement upload logic with placeholders
    folderNumber++;
    preferences.begin("camera", false);
    preferences.putInt("folderNumber", folderNumber);
```

```
    preferences.end();
    Serial.println("Image capture and upload complete. Waiting for next signal...");
  }
}

String getFolderName(int folderNum) {
  return String(folderNum);
}

void captureAndSaveImage(const String& folderName, int imageIndex) {
  camera_fb_t * fb = esp_camera_fb_get();
  if (!fb) {
    Serial.println("Camera capture failed");
    return;
  }
  String path = "/" + folderName + "/image" + String(imageIndex) + ".jpg";
  fs::FS &fs = SD_MMC;
  File file = fs.open(path.c_str(), FILE_WRITE);
  if (!file) {
    Serial.println("Failed to open file in writing mode");
  } else {
    file.write(fb->buf, fb->len);
    Serial.printf("Saved file to path: %s\n", path.c_str());
  }
  file.close();
  esp_camera_fb_return(fb);
}
```

// uploadImageToFirebaseStorage / uploadFolderToFirebase: implement using firebaseHost, firebaseAuth, firebaseStorageBucket placeholders

4) Additional LCD + Firebase + timing sketch

```
/* lcd_timing_firebase.ino
  Extra LCD + timer logic. Sanitized credentials.
*/

#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <WiFi.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <TimeLib.h>
#include <FirebaseESP32.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);

const char* ssid = "PLACEHOLDER_YOUR_SSID";
const char* password = "PLACEHOLDER_YOUR_WIFI_PASSWORD";

#define FIREBASE_HOST "PLACEHOLDER_YOUR_FIREBASE_HOST"
#define FIREBASE_API_KEY "PLACEHOLDER_YOUR_FIREBASE_API_KEY"

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org", 19800, 60000);
```

```
#define LDRPIN 34
const int ledPin = 26;
const int ledPin1 = 27;
const int b1 = 25;

unsigned long startTime = 0;
bool timerRunning = false;
bool b1State = false;
unsigned long displayTimeStart = 0;
bool displayingTime = false;

String currentDate;
String currentTime;
String lastTermStartTime;
String lastTermRtime;

FirebaseData firebaseData;
FirebaseConfig firebaseConfig;
FirebaseAuth firebaseAuth;

bool b1low = false;
bool b1high = false;

int recordCounter = 0;

void setup() {
  Serial.begin(115200);

  pinMode(ledPin, OUTPUT);
```

```
 pinMode(ledPin1, OUTPUT);
 pinMode(b1, OUTPUT);
 digitalWrite(b1, LOW);
 delay(500);

 Wire.begin(21, 22);
 lcd.init();
 lcd.backlight();

 lcd.setCursor(0, 0);
 lcd.print("Do Your Best..");

 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Connecting to WiFi...");
 }
 Serial.println("Connected to WiFi");

 timeClient.begin();

 firebaseConfig.host = FIREBASE_HOST;
 firebaseConfig.api_key = FIREBASE_API_KEY;
 firebaseAuth.user.email = "PLACEHOLDER_your@example.com";
 firebaseAuth.user.password = "PLACEHOLDER_your_password";

 Firebase.begin(&firebaseConfig, &firebaseAuth);
 Firebase.reconnectWiFi(true);
}
```

// parseTime, displayTimeDifference, updateLaserState, etc. Keep your original logic but ensure any credentials used reference placeholders above.

5) Web files (HTML + JS) — Firebase config sanitized

Dashboard / Progress HTML

```
<!-- progress.html (sanitized) -->
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8"/>
 <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
 <title>Progress</title>
 <style>
  /* (omitted long CSS — same as original; keep as needed) */
 </style>
 <script type="module" src="progress.js" defer></script>
</head>
<body>
 <section class="header">
  <div class="box">
   <a href="final.html"><img class="qe" src="run0001.png" width="300" height="60"></a>
  </div>
  <div class="box hideOnMobile">
   <ul id="navbar">
    <li><a class="n" href="final.html">Dashboard</a></li>
    <li><a class="n" href="prograss.html">Progress</a></li>
```

```
    </ul>
   </div>
  </section>

  <div class="secondtopic"><b>Your Progress Report as a Runner .....</b></div>
  <div class="mychart">
   <canvas id="myChart"></canvas>
  </div>

  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</body>
</html>
```

web firebase JS

```
// firebaseConfig.js (sanitized)
export const firebaseConfig = {
  apiKey: "PLACEHOLDER_YOUR_FIREBASE_API_KEY",
  authDomain: "PLACEHOLDER_YOUR_FIREBASE_AUTH_DOMAIN",
  databaseURL: "PLACEHOLDER_YOUR_FIREBASE_DATABASE_URL",
  projectId: "PLACEHOLDER_YOUR_FIREBASE_PROJECT_ID",
  storageBucket: "PLACEHOLDER_YOUR_FIREBASE_STORAGE_BUCKET",
  messagingSenderId: "PLACEHOLDER_YOUR_MESSAGING_SENDER_ID",
  appId: "PLACEHOLDER_YOUR_FIREBASE_APP_ID"
};

// Example usage in progress.js:
import { initializeApp } from "https://www.gstatic.com/firebasejs/10.12.3/firebase-app.js";
```

```
import { getDatabase, ref, onValue } from "https://www.gstatic.com/firebasejs/10.12.3/firebase-database.js";
import { firebaseConfig } from './firebaseConfig.js';

const app = initializeApp(firebaseConfig);
const db = getDatabase(app);

function GetAllDataRealtime() {
  const dbRef = ref(db, `runner`);
  onValue(dbRef, (snapshot) => {
    const data = [];
    snapshot.forEach(childSnapshot => {
      data.push(childSnapshot.val());
    });
    initChart(data);
  }, { onlyOnce: false });
}

// Helper: convert timing to seconds
function timingToSeconds(timing) {
  if (!timing) return null;
  const [hours, minutes, seconds, milliseconds] = timing.split(':').map(Number);
  return hours * 3600 + minutes * 60 + seconds + (milliseconds || 0) / 1000;
}

// Chart initialization logic (use Chart.js)
function initChart(data) {
  const labels = data.map(entry => entry.Date || " ");
  const timings = data.map(entry => entry.Timing ? timingToSeconds(entry.Timing) : null);
```

```
  // ... Chart.js setup (same as original) ...
}

// call on load
window.onload = async () => {
  try {
    await loadChartJs(); // or ensure Chart.js loaded
    GetAllDataRealtime();
  } catch (error) {
    console.error(error);
  }
};
```

Image display page

```
<!-- imageDisplay.html -->
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width,initial-scale=1">
  <title>Image Display</title>
  <script type="module" src="firebaseImageFetcher.js"></script>
  <style>
    /* (same CSS as original; omitted for brevity) */
  </style>
</head>
<body>
```

```
<section class="header">
  <div class="box">
    <a href="final.html"><img class="qe" src="run0001.png"></a>
  </div>
  <div class="box">
    <ul id="navbar">
      <li><a class="n" href="final.html">Dashboard</a></li>
      <li><a class="n" href="prograss.html">Progress</a></li>
    </ul>
  </div>
</section>

<h1 class="secondtopic"><b>Starting Block Position .....</b></h1>
<div id="image-container" class="imageContainer">
  <div class="imageSlideshow">
    <button id="prevBtn" class="slidebtn1"><img src="icons/left.png" class="icon"></button>
    <button id="zoomInBtn" class="zoomButton"><img src="icons/zoom in.png"></button>
    <button id="zoomOutBtn" class="zoomButton"><img src="icons/zoom out.png"></button>
    <button id="nextBtn" class="slidebtn2"><img src="icons/right.png" class="icon"></button>
  </div>
  <img src="" alt="image" id="imagel" class="slideshowimage">
</div>

<script type="module">
  // firebaseImageFetcher.js (sanitized sample)
  import { initializeApp } from "https://www.gstatic.com/firebasejs/10.12.3/firebase-app.js";
  import { getStorage, ref as storageRef, listAll, getDownloadURL } from
"https://www.gstatic.com/firebasejs/10.12.3/firebase-storage.js";
  import { firebaseConfig } from './firebaseConfig.js';
```

```
const app = initializeApp(firebaseConfig);
const storage = getStorage(app);

const imageUrls = [];
function fetchImages(folderName) {
  const imagesContainer = document.getElementById('image-container');
  if (!imagesContainer) { console.error('Image container not found.'); return; }
  const listRef = storageRef(storage, folderName);
  listAll(listRef).then((result) => {
    let skipFirst = true;
    const promises = result.items.map((imageRef) => {
      if (skipFirst) { skipFirst = false; return Promise.resolve(null); }
      return getDownloadURL(imageRef).then((url) => {
        imageUrls.push(url);
        startSlideshow();
        return url;
      }).catch((error) => { console.error('Error getting download URL:', error); return null; });
    });
    Promise.all(promises).then(() => { console.log('All image URLs:', imageUrls); }).catch(console.error);
  }).catch((error) => { console.error('Error listing images:', error); });
}

// on load
window.addEventListener('load', () => {
  const folderName = new URLSearchParams(window.location.search).get('id');
  if (folderName) fetchImages(folderName);
  else console.error('Folder name not provided.');
  document.getElementById('prevBtn').addEventListener('click', showPreviousImage);
```

```
    document.getElementById('nextBtn').addEventListener('click', showNextImage);
    document.getElementById('zoomInBtn').addEventListener('click', () => zoomImage(0.1));
    document.getElementById('zoomOutBtn').addEventListener('click', () => zoomImage(-0.1));
  });

  // implement startSlideshow, showPreviousImage, showNextImage, zoomImage as in original code
 </script>
</body>
</html>
```