LAMPIRAN

DatabaseManager.Cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System.Text;
using TMPro;

// Firebase imports - PASTIKAN SUDAH INSTALL FIREBASE SDK
using Firebase;
using Firebase.Database;
using Firebase.Extensions;

public class FirebaseDatabaseManager : MonoBehaviour
{
    [Header("🔗 Main UI References")]
    public Transform tableContent;
    public GameObject tableRowPrefab;

    [Header("📊 Status Display")]
    public GameObject statusTextObject;

    [Header("⭕ Action Buttons")]
    public Button simpanButton;
    public Button updateButton;
    public Button hapusButton;

    [Header(" Form Panel")]
    public GameObject formPopup;
    public GameObject namaInputObj;
    public GameObject nimInputObj;
    public GameObject kelasInputObj;
    public Button okButton;
    public Button cancelButton;
    public GameObject formTitleObject;

    [Header(" Drag & Drop")]
    public GameObject tableDropZone;

    [Header("🔊 Audio Settings")]
    public AudioSource audioSource;
    public AudioClip simpanSound;
    public AudioClip updateSound;
    public AudioClip hapusSound;
    public AudioClip clickSound;
    [Range(0f, 1f)]
    public float audioVolume = 0.7f;
    public bool enableAudio = true;

    [Header(" Firebase Configuration")]
    [SerializeField] private string firebaseDatabaseName = "Mahasiswa"; // Updated to match Firebase collection
    [SerializeField] private bool enableOfflineMode = true;
    [SerializeField] private bool autoRetryConnection = true;
    [SerializeField] private int maxRetryAttempts = 3;

    [Header("⚙ Advanced Settings")]
    [SerializeField] private float connectionTimeout = 10f;
    [SerializeField] private bool enableDebugLogging = true;

    // Firebase references - Updated for Firestore structure
    private DatabaseReference databaseRef;
    private DatabaseReference mahasiswaRef; // Changed from siswaRef
    private bool firebaseInitialized = false;

    // Data dan state management
    private List<Siswa> siswaData = new List<Siswa>();
    private string currentEditId = "";
    private string currentAction = "";
    private string storedAction = "";
    private bool dataStoredInButton = false;
    private Siswa pendingData = null;

    // Input field references (mendukung both Text dan TMP)
    private TMP_InputField tmpNamaInput;
    private TMP_InputField tmpNimInput;
    private TMP_InputField tmpKelasInput;
    private InputField stdNamaInput;
    private InputField stdNimInput;
```

```
            private     InputField
stdKelasInput;

    // Text component references
    private Text statusText;
             private      TMP_Text
statusTMPText;      private  Text
formTitleText;
             private      TMP_Text
formTitleTMPText;

    // Performance optimization
               private       bool
isOperationInProgress = false;

    void Start()
    {
          DebugLog("[ START] ===
FIREBASE     DATABASE     MANAGER
INITIALIZING ===");
        InitializeComponents();
        InitializeAudio();
        SetupUI();
         StartCoroutine(Initialize
FirebaseWithRetry());
    }

        //   ====================
FIREBASE           INITIALIZATION
====================

                        IEnumerator
InitializeFirebaseWithRetry()
    {
           DebugLog("[  FIREBASE]
Initializing Firebase...");
         ShowStatus("[🔄 LOADING]
Initializing            Firebase
connection...", Color.blue);

        int attempts = 0;
        bool initSuccess = false;

          while (!initSuccess &&
attempts < maxRetryAttempts)
        {
            attempts++;
            DebugLog($"[🔄 RETRY]
Firebase  initialization  attempt
{attempts}/{maxRetryAttempts}");

               // Check Firebase
dependencies
             var dependencyTask =
FirebaseApp.CheckAndFixDependenci
esAsync();
              yield return new
WaitUntil(()                    =>
dependencyTask.IsCompleted);

            var dependencyStatus =
dependencyTask.Result;

             if (dependencyStatus
== DependencyStatus.Available)
            {
                    // Initialize
Firebase
                FirebaseApp app =
FirebaseApp.DefaultInstance;

                if (app != null)
                {
                    databaseRef =
FirebaseDatabase.DefaultInstance.
RootReference;
                   mahasiswaRef =
databaseRef.Child(firebaseDatabas
eName); // Updated reference name

                        // Enable
offline capabilities
                            if
(enableOfflineMode)
                    {
                        FirebaseD
atabase.DefaultInstance.SetPersis
tenceEnabled(true);
                      DebugLog("[
FIREBASE] Offline mode enabled");
                    }

                    firebaseIniti
alized = true;
                    initSuccess =
true;

                    ShowStatus("[
SUCCESS]    Firebase    connected
successfully!", Color.green);
                     DebugLog("[
FIREBASE]   Firebase   initialized
successfully");

                  // Load initial
data
                    LoadDataFromF
irebase();
                }
                else
                {
                      DebugLog("[
FIREBASE]       Firebase       app
initialization failed");
                }
            }
            else
            {
                     DebugLog($"[
FIREBASE] Dependency check failed:
{dependencyStatus}");
                   ShowStatus($"[
ERROR] Firebase dependency error:
{dependencyStatus}", Color.red);
            }
```

```
            if (!initSuccess &&
attempts < maxRetryAttempts)
            {
                DebugLog($"[⏳
WAIT] Waiting before retry attempt
{attempts + 1}...");
                yield return new
WaitForSeconds(2f);
            }
        }

        if (!initSuccess)
        {
            HandleFirebaseConnect
ionFailure();
        }
    }

                            void
HandleFirebaseConnectionFailure()
    {
                Debug.LogError("[
FIREBASE] All Firebase connection
attempts failed!");
            ShowStatus("[  ERROR]
Firebase connection failed! Check
configuration.", Color.red);

       // Optional: Load test data
for development
        if (Application.isEditor)
        {
                DebugLog("[  DEV]
Loading     test     data     for
development...");
            LoadTestData();
        }
    }

        //   ====================
FIREBASE      DATA      OPERATIONS
====================

                public       void
LoadDataFromFirebase()
    {
        if (!firebaseInitialized)
        {
             ShowStatus("[ ERROR]
Firebase    not    initialized!",
Color.red);
            return;
        }

            DebugLog("[  FIREBASE]
Loading data from Firebase...");
          ShowStatus("[ LOADING]
Loading  data  from  Firebase...",
Color.blue);

        mahasiswaRef.GetValueAsyn
c().ContinueWithOnMainThread(task
=> // Updated reference
        {
            if (task.IsFaulted)
            {
                    DebugLog($"[
FIREBASE]       Load       failed:
{task.Exception}");
                ShowStatus("[
ERROR] Failed to load data from
Firebase", Color.red);
                return;
            }

            if (task.IsCompleted)
            {
                    DataSnapshot
snapshot = task.Result;

                                if
(snapshot.Exists)
                {
                    siswaData.Cle
ar();

                            foreach
(DataSnapshot  childSnapshot  in
snapshot.Children)
                    {
                        try
                        {
                            Dicti
onary<string,  object>  data  =
(Dictionary<string,
object>)childSnapshot.Value;
                            Siswa
siswa                       =
CreateSiswaFromFirebaseData(child
Snapshot.Key, data);
                            siswa
Data.Add(siswa);
                        }
                            catch
(Exception e)
                        {
                            Debug
Log($"[ FIREBASE] Error parsing
data: {e.Message}");
                        }
                    }

                    UpdateTableDi
splay();
                ShowStatus($"[
SUCCESS]        {siswaData.Count}
records  loaded  from  Firebase!",
Color.green);
                    DebugLog($"[
FIREBASE]   Successfully   loaded
{siswaData.Count} records");
                }
                else
```

```
                {
                    siswaData.Clear();
                    UpdateTableDisplay();
                    ShowStatus("[INFO] No data found in Firebase database", Color.blue);
                    DebugLog("[FIREBASE] No data exists in Firebase");
                }
            }
        });
    }

      // Updated method untuk membuat Siswa from Firebase data - menyesuaikan dengan struktur baru
                            Siswa CreateSiswaFromFirebaseData(string firebaseId, Dictionary<string, object> data)
    {
        Siswa siswa = new Siswa();
        siswa.id = firebaseId;

        if (data != null)
        {
            // Menyesuaikan dengan field names yang mungkin berbeda di Firebase baru
                                if (data.ContainsKey("nama") && data["nama"] != null)
                    siswa.nama = data["nama"].ToString();
                        else if (data.ContainsKey("Nama") && data["Nama"] != null) // Case sensitive fallback
                    siswa.nama = data["Nama"].ToString();

                                if (data.ContainsKey("nim") && data["nim"] != null)
                    siswa.nim = data["nim"].ToString();
                        else if (data.ContainsKey("NIM") && data["NIM"] != null) // Case sensitive fallback
                    siswa.nim = data["NIM"].ToString();

                                if (data.ContainsKey("kelas") && data["kelas"] != null)
                    siswa.kelas = data["kelas"].ToString();
                        else if (data.ContainsKey("Kelas") && data["Kelas"] != null) // Case sensitive fallback
                    siswa.kelas = data["Kelas"].ToString();
        }

        return siswa;
    }

     // Helper method to convert Siswa to Firebase data - ensure consistent field names
      Dictionary<string, object> SiswaToFirebaseData(Siswa siswa)
    {
       Dictionary<string, object> data = new Dictionary<string, object>();
        data["nama"] = siswa.nama ?? ""; // Using lowercase to match existing data
        data["nim"] = siswa.nim ?? "";
                data["kelas"] = siswa.kelas ?? "";
        return data;
    }

                        IEnumerator SaveDataToFirebase(Siswa siswa)
    {
        if (!firebaseInitialized)
        {
             ShowStatus("[ERROR] Firebase not initialized!", Color.red);
            yield break;
        }

           DebugLog($"[FIREBASE] Saving: {siswa.nama}");
            ShowStatus("[SAVING] Saving data to Firebase...", Color.blue);

         // Generate new push key for new entry
               string newKey = mahasiswaRef.Push().Key;       // Updated reference
        siswa.id = newKey;

                var saveTask = mahasiswaRef.Child(newKey).SetValueAsync(SiswaToFirebaseData(siswa)); // Updated reference

              yield return new WaitUntil(() => saveTask.IsCompleted);

          isOperationInProgress = false;
```

```
        if (saveTask.IsFaulted)
        {
            DebugLog($"[ FIREBASE] Save failed: {saveTask.Exception}");
            ShowStatus($"[ ERROR] Save failed: {saveTask.Exception.Message}", Color.red);
        }
        else
        {
            ShowStatus($"[ SUCCESS] {siswa.nama} saved to Firebase!", Color.green);
            DebugLog("[ FIREBASE] Save successful! Reloading data...");

            yield return new WaitForSeconds(1f);
            LoadDataFromFirebase();
        }
    }

    IEnumerator UpdateDataInFirebase(Siswa siswa)
    {
        if (!firebaseInitialized)
        {
            ShowStatus("[ ERROR] Firebase not initialized!", Color.red);
            yield break;
        }

        DebugLog($"[🔄 FIREBASE] Updating: ID={siswa.id}, {siswa.nama}");
        ShowStatus("[🔄 UPDATING] Updating data in Firebase...", Color.blue);

        var updateTask = mahasiswaRef.Child(siswa.id).UpdateChildrenAsync(SiswaToFirebaseData(siswa)); // Updated reference

        yield return new WaitUntil(() => updateTask.IsCompleted);

        isOperationInProgress = false;

        if (updateTask.IsFaulted)
        {
            DebugLog($"[ FIREBASE] Update failed: {updateTask.Exception}");
            ShowStatus($"[ ERROR] Update failed: {updateTask.Exception.Message}", Color.red);
        }
        else
        {
            ShowStatus($"[ SUCCESS] {siswa.nama} updated in Firebase!", Color.green);
            DebugLog("[ FIREBASE] Update successful, reloading data...");

            yield return new WaitForSeconds(0.5f);
            LoadDataFromFirebase();
            ResetSelection();
        }
    }

    IEnumerator DeleteDataFromFirebase(string id)
    {
        if (!firebaseInitialized)
        {
            ShowStatus("[ ERROR] Firebase not initialized!", Color.red);
            yield break;
        }

        DebugLog($"[ FIREBASE] Deleting: ID={id}");

        Siswa siswaToDelete = siswaData.Find(s => s.id == id);
        string namaToDelete = siswaToDelete?.nama ?? $"ID {id}";

        ShowStatus("[ DELETING] Deleting data from Firebase...", Color.blue);

        var deleteTask = mahasiswaRef.Child(id).RemoveValueAsync(); // Updated reference

        yield return new WaitUntil(() => deleteTask.IsCompleted);

        isOperationInProgress = false;

        if (deleteTask.IsFaulted)
        {
            DebugLog($"[ FIREBASE] Delete failed: {deleteTask.Exception}");
            ShowStatus($"[ ERROR] Delete failed:
```

```
{deleteTask.Exception.Message}",
Color.red);
        }
        else
        {
           ShowStatus($"[ SUCCESS]
{namaToDelete}    deleted    from
Firebase!", Color.green);
            DebugLog("[ FIREBASE]
Delete    successful,    reloading
data...");

               yield return new
WaitForSeconds(0.5f);
            LoadDataFromFirebase(
);
            ResetSelection();
        }
    }

        //   ====================
COMPONENT       SETUP       METHODS
====================

    void InitializeComponents()
    {
        SetupTextComponents();
        SetupInputFields();
         ValidateRequiredComponent
s();
    }

    void InitializeAudio()
    {
        if (audioSource == null)
        {
                    audioSource  =
GetComponent<AudioSource>();
             if (audioSource ==
null)
            {
                    audioSource =
gameObject.AddComponent<AudioSour
ce>();
                    DebugLog("[
AUDIO]      AudioSource      auto-
created");
            }
        }

        if (audioSource != null)
        {
             audioSource.volume =
audioVolume;
            audioSource.playOnAwa
ke = false;
              audioSource.loop  =
false;
            DebugLog("[ AUDIO]
AudioSource configured");
        }

        LoadDefaultSounds();
           DebugLog($"[ AUDIO]
Audio system initialized. Enabled:
{enableAudio}");
    }

    void LoadDefaultSounds()
    {
        if (simpanSound == null)
                    simpanSound  =
Resources.Load<AudioClip>("Audio/
simpan");
        if (updateSound == null)
                   updateSound  =
Resources.Load<AudioClip>("Audio/
update");
        if (hapusSound == null)
                    hapusSound  =
Resources.Load<AudioClip>("Audio/
hapus");
        if (clickSound == null)
                    clickSound  =
Resources.Load<AudioClip>("Audio/
click");
    }

   void PlayAudio(AudioClip clip,
string soundName = "Unknown")
    {
            if  (!enableAudio  ||
audioSource  ==  null  ||  clip  ==
null) return;

        try
        {
               audioSource.clip  =
clip;
             audioSource.volume =
audioVolume;
            audioSource.Play();
            DebugLog($"[ AUDIO]
Playing {soundName} sound");
        }
       catch (System.Exception e)
        {
            DebugLog($"[ AUDIO]
Error    playing    {soundName}:
{e.Message}");
        }
    }

    public void PlaySimpanSound()
=>          PlayAudio(simpanSound,
"Simpan");
    public void PlayUpdateSound()
=>          PlayAudio(updateSound,
"Update");
     public void PlayHapusSound()
=>          PlayAudio(hapusSound,
"Hapus");
```

```
    public void PlayClickSound() => PlayAudio(clickSound, "Click");

    void SetupTextComponents()
    {
        if (statusTextObject != null)
        {
            statusText = statusTextObject.GetComponent<Text>();
            statusTMPText = statusTextObject.GetComponent<TMP_Text>();
        }

        if (formTitleObject != null)
        {
            formTitleText = formTitleObject.GetComponent<Text>();
            formTitleTMPText = formTitleObject.GetComponent<TMP_Text>();
        }
    }

    void SetupInputFields()
    {
        if (namaInputObj != null)
        {
            tmpNamaInput = namaInputObj.GetComponent<TMP_InputField>();
            stdNamaInput = namaInputObj.GetComponent<InputField>();
        }

        if (nimInputObj != null)
        {
            tmpNimInput = nimInputObj.GetComponent<TMP_InputField>();
            stdNimInput = nimInputObj.GetComponent<InputField>();
        }

        if (kelasInputObj != null)
        {
            tmpKelasInput = kelasInputObj.GetComponent<TMP_InputField>();
            stdKelasInput = kelasInputObj.GetComponent<InputField>();
        }

        if (formPopup != null)
            AutoDetectInputFields();
    }

    void AutoDetectInputFields()
    {
        TMP_InputField[] tmpFields = formPopup.GetComponentsInChildren<TMP_InputField>();
        InputField[] stdFields = formPopup.GetComponentsInChildren<InputField>();

        if (tmpFields.Length >= 3)
        {
            if (tmpNamaInput == null) tmpNamaInput = tmpFields[0];
            if (tmpNimInput == null) tmpNimInput = tmpFields[1];
            if (tmpKelasInput == null) tmpKelasInput = tmpFields[2];
        }
        else if (stdFields.Length >= 3)
        {
            if (stdNamaInput == null) stdNamaInput = stdFields[0];
            if (stdNimInput == null) stdNimInput = stdFields[1];
            if (stdKelasInput == null) stdKelasInput = stdFields[2];
        }
    }

    bool ValidateRequiredComponents()
    {
        List<string> missingComponents = new List<string>();

        if (tableContent == null) missingComponents.Add("Table Content");
        if (tableRowPrefab == null) missingComponents.Add("Table Row Prefab");
        if (formPopup == null) missingComponents.Add("Form Popup");
        if (simpanButton == null) missingComponents.Add("Simpan Button");
        if (updateButton == null) missingComponents.Add("Update Button");
        if (hapusButton == null) missingComponents.Add("Hapus Button");
```

```
                                if (missingComponents.Count > 0)
        {
                string missing = string.Join(", ", missingComponents);
            ShowStatus($"[ ERROR] Missing components: {missing}", Color.red);
                Debug.LogError($"[VALIDATION] Missing required components: {missing}");
            return false;
        }

        return true;
    }

    void SetupUI()
    {
                                if (!ValidateRequiredComponents()) return;

        SetupButtonEvents();
        SetupDragAndDrop();

        if (formPopup != null)
            formPopup.SetActive(false);

        ResetSelection();
    }

    void SetupButtonEvents()
    {
        if (simpanButton != null)
        {
            simpanButton.onClick.RemoveAllListeners();
            simpanButton.onClick.AddListener(() => { PlayClickSound(); ShowFormPanel("save"); });
        }

        if (updateButton != null)
        {
            updateButton.onClick.RemoveAllListeners();
            updateButton.onClick.AddListener(() => { PlayClickSound(); ShowFormPanel("update"); });
        }

        if (hapusButton != null)
        {
            hapusButton.onClick.RemoveAllListeners();
            hapusButton.onClick.AddListener(() => { PlayClickSound(); ShowFormPanel("delete"); });
        }

        if (cancelButton != null)
        {
            cancelButton.onClick.RemoveAllListeners();
            cancelButton.onClick.AddListener(() => { PlayClickSound(); HideFormPanel(); });
        }

        if (okButton != null)
        {
            okButton.onClick.RemoveAllListeners();
            okButton.onClick.AddListener(() => { PlayClickSound(); ProcessFormData(); });
        }
    }

    void SetupDragAndDrop()
    {
        SetupDragButton(simpanButton?.gameObject, "save");
        SetupDragButton(updateButton?.gameObject, "update");
        SetupDragButton(hapusButton?.gameObject, "delete");

       if (tableDropZone != null)
            SetupTableDropZone(tableDropZone);
        else if (tableContent != null)
            SetupTableDropZone(tableContent.gameObject);
    }

                                void SetupDragButton(GameObject button, string action)
    {
       if (button == null) return;

        DraggableButton dragComp = button.GetComponent<DraggableButton>();
        if (dragComp == null)
                        dragComp = button.AddComponent<DraggableButton>();

        dragComp.action = action;
        dragComp.databaseManager = this;
    }
```

```
                                void
SetupTableDropZone(GameObject
dropZone)
    {
          if (dropZone == null)
return;

         TableDropZone dropComp =
dropZone.GetComponent<TableDropZo
ne>();
        if (dropComp == null)
                        dropComp   =
dropZone.AddComponent<TableDropZo
ne>();

        dropComp.databaseManager =
this;
    }

     // ==================== FORM
MANAGEMENT ====================

                public       void
ShowFormPanel(string action)
    {
         if (formPopup == null ||
IsOperationInProgress()) return;

        currentAction = action;
        storedAction = "";
            dataStoredInButton  =
false;
        pendingData = null;
        formPopup.SetActive(true);

        switch(action)
        {
                    case  "save":
ConfigureFormForSave(); break;
                   case  "update":
ConfigureFormForUpdate(); break;
                   case  "delete":
ConfigureFormForDelete(); break;
        }
    }

    void ConfigureFormForSave()
    {
          SetFormTitle("➕ TAMBAH
DATA BARU");
        ClearFormInputs();
        EnableFormInputs(true);
             UpdateOKButtonText("
SIMPAN");
         ShowStatus("[ INFO] Fill
form  and  click  OK,  then  drag
SIMPAN to table!", Color.blue);
    }

    void ConfigureFormForUpdate()
    {
                                if
(!ValidateSelectionForEdit("UPDAT
E")) return;

           SetFormTitle("✏ EDIT
DATA");
        LoadSelectedDataToForm();
        EnableFormInputs(true);
           UpdateOKButtonText("🔄
UPDATE");

           Siswa  selectedSiswa  =
siswaData.Find(s   =>   s.id   ==
currentEditId);
          ShowStatus($"[✏ EDIT]
Editing     {selectedSiswa.nama},
click  OK  then  drag  UPDATE  to
table!", Color.blue);
    }

    void ConfigureFormForDelete()
    {
                                if
(!ValidateSelectionForEdit("DELET
E")) return;

            SetFormTitle("  HAPUS
DATA");
        LoadSelectedDataToForm();
        EnableFormInputs(false);
             UpdateOKButtonText("
HAPUS");

           Siswa  selectedSiswa  =
siswaData.Find(s   =>   s.id   ==
currentEditId);
          ShowStatus($"[  DELETE]
Confirm                    delete
{selectedSiswa.nama},   click   OK
then   drag   HAPUS   to   table!",
Color.red);
    }

                                bool
ValidateSelectionForEdit(string
actionName)
    {
                                  if
(string.IsNullOrEmpty(currentEdit
Id))
        {
            ShowStatus($"[ ERROR]
Select data from table first for
{actionName}!", Color.red);
            HideFormPanel();
            return false;
        }

           Siswa  selectedSiswa  =
siswaData.Find(s   =>   s.id   ==
currentEditId);
       if (selectedSiswa == null)
```

```
        {
            ShowStatus($"[ ERROR] Data with ID {currentEditId} not found!", Color.red);
            ResetSelection();
            HideFormPanel();
            return false;
        }

        return true;
    }

    public void HideFormPanel()
    {
        if (formPopup != null) formPopup.SetActive(false);
    }

    void ProcessFormData()
    {
        if (string.IsNullOrEmpty(currentAction)) return;

        switch (currentAction)
        {
            case "save": HandleSaveDataPreparation(); break;
            case "update": HandleUpdateDataPreparation(); break;
            case "delete": HandleDeleteDataPreparation(); break;
        }
    }

    void HandleSaveDataPreparation()
    {
        string namaValue = GetInputValue("nama");
        string nimValue = GetInputValue("nim");
        string kelasValue = GetInputValue("kelas");

        if (!ValidateInputData(namaValue, nimValue, kelasValue)) return;

        pendingData = new Siswa(namaValue, nimValue, kelasValue);
        dataStoredInButton = true;
        storedAction = "save";
        HideFormPanel();
        ShowStatus("[ READY] Data ready to save! Drag SIMPAN button to table.", Color.green);
    }

    void HandleUpdateDataPreparation()
    {
        if (string.IsNullOrEmpty(currentEditId)) return;

        string namaValue = GetInputValue("nama");
        string nimValue = GetInputValue("nim");
        string kelasValue = GetInputValue("kelas");

        if (!ValidateInputData(namaValue, nimValue, kelasValue)) return;

        pendingData = new Siswa { id = currentEditId, nama = namaValue, nim = nimValue, kelas = kelasValue };
        dataStoredInButton = true;
        storedAction = "update";
        HideFormPanel();
        ShowStatus("[ READY] Data ready to update! Drag UPDATE button to table.", Color.green);
    }

    void HandleDeleteDataPreparation()
    {
        if (string.IsNullOrEmpty(currentEditId)) return;

        pendingData = new Siswa { id = currentEditId };
        dataStoredInButton = true;
        storedAction = "delete";
        HideFormPanel();
        ShowStatus("[ READY] Data ready to delete! Drag HAPUS button to table.", Color.red);
    }

    bool ValidateInputData(string nama, string nim, string kelas)
    {
        if (string.IsNullOrEmpty(nama) || string.IsNullOrEmpty(nim) || string.IsNullOrEmpty(kelas))
        {
            ShowStatus("[ ERROR] All fields must be filled!", Color.red);
            return false;
        }
        return true;
    }
```

```
    // ==================== DRAG & DROP                    EXECUTION ====================

                public       void ExecuteActionFromDrag(string draggedAction)
    {
                                if (IsOperationInProgress()       || !dataStoredInButton            || pendingData == null) return;
            if  (storedAction  != draggedAction) return;

          isOperationInProgress  = true;

        switch (draggedAction)
        {
            case "save":
                PlaySimpanSound();
                 StartCoroutine(SaveDataToFirebase(pendingData));
                break;
            case "update":
                PlayUpdateSound();
                 StartCoroutine(UpdateDataInFirebase(pendingData));
                break;
            case "delete":
                PlayHapusSound();
                 StartCoroutine(DeleteDataFromFirebase(pendingData.id));
                break;
        }

        ResetActionState();
    }

     bool  IsOperationInProgress() => isOperationInProgress;

    void ResetActionState()
    {
             dataStoredInButton  = false;
        storedAction = "";
        currentAction = "";
        pendingData = null;
    }

    // ==================== INPUT FIELD                    UTILITIES ====================

      string  GetInputValue(string field)
    {
        string value = "";
        switch (field)
        {
            case "nama":
                 if (tmpNamaInput !=       null)       value       = tmpNamaInput.text;
                            else  if (stdNamaInput  !=  null)  value  = stdNamaInput.text;
                break;
            case "nim":
                if (tmpNimInput != null) value = tmpNimInput.text;
                            else  if (stdNimInput  !=  null)  value  = stdNimInput.text;
                break;
            case "kelas":
                 if (tmpKelasInput !=       null)       value       = tmpKelasInput.text;
                            else  if (stdKelasInput  !=  null)  value  = stdKelasInput.text;
                break;
        }
        return value.Trim();
    }

       void  SetInputValue(string field, string value)
    {
         switch (field)
        {
            case "nama":
                 if (tmpNamaInput !=   null)   tmpNamaInput.text   = value;
                            else  if (stdNamaInput       !=       null) stdNamaInput.text = value;
                break;
            case "nim":
                if (tmpNimInput != null) tmpNimInput.text = value;
                            else  if (stdNimInput       !=       null) stdNimInput.text = value;
                break;
            case "kelas":
                 if (tmpKelasInput !=   null)   tmpKelasInput.text   = value;
                            else  if (stdKelasInput      !=      null) stdKelasInput.text = value;
                break;
        }
    }

    void ClearFormInputs()
    {
        SetInputValue("nama", "");
         SetInputValue("nim", "");
```

```
            SetInputValue("kelas",
"");
    }

    void LoadSelectedDataToForm()
    {
            Siswa selectedSiswa =
siswaData.Find(s => s.id ==
currentEditId);
        if (selectedSiswa != null)
        {
             SetInputValue("nama",
selectedSiswa.nama);
              SetInputValue("nim",
selectedSiswa.nim);
            SetInputValue("kelas",
selectedSiswa.kelas);
        }
    }

      void EnableFormInputs(bool
enable)
    {
        if (tmpNamaInput != null)
tmpNamaInput.interactable       =
enable;
         else if (stdNamaInput !=
null) stdNamaInput.interactable =
enable;

         if (tmpNimInput != null)
tmpNimInput.interactable        =
enable;
         else if (stdNimInput !=
null) stdNimInput.interactable =
enable;

        if (tmpKelasInput != null)
tmpKelasInput.interactable      =
enable;
         else if (stdKelasInput !=
null)  stdKelasInput.interactable
= enable;
    }

     // ==================== TEXT
DISPLAY                  UTILITIES
====================

       void  SetStatusText(string
message)
    {
        if (statusText != null)
                statusText.text =
message;
         else if (statusTMPText !=
null)
              statusTMPText.text =
message;
    }

    void SetStatusTextColor(Color
color)
    {
        if (statusText != null)
               statusText.color =
color;
         else if (statusTMPText !=
null)
             statusTMPText.color =
color;
    }

        void  SetFormTitle(string
title)
    {
        if (formTitleText != null)
             formTitleText.text =
title;
         else if (formTitleTMPText
!= null)
              formTitleTMPText.text
= title;
    }

   void UpdateOKButtonText(string
text)
    {
        if (okButton != null)
        {
              Text okButtonText =
okButton.GetComponentInChildren<T
ext>();
                          TMP_Text
okButtonTMPText                 =
okButton.GetComponentInChildren<T
MP_Text>();

              if (okButtonText !=
null) okButtonText.text = text;
                         else   if
(okButtonTMPText     !=     null)
okButtonTMPText.text = text;
        }
    }

    public void ShowStatus(string
message, Color color)
    {
        SetStatusText(message);
       SetStatusTextColor(color);
          DebugLog($"[📢 STATUS]
{message}");

         CancelInvoke("ClearStatus
");
        Invoke("ClearStatus", 8f);
    }

    void ClearStatus()
    {
        SetStatusText("");
    }
```

```
    // ==================== TABLE DISPLAY MANAGEMENT ====================

    void UpdateTableDisplay()
    {
        if (tableContent == null || tableRowPrefab == null) return;

        ClearExistingRows();
        CreateTableRows();
        ForceLayoutUpdate();
        ResetSelection();
    }

    void ClearExistingRows()
    {
            int childCount = tableContent.childCount;
        for (int i = childCount - 1; i >= 0; i--)
        {
            DestroyImmediate(tableContent.GetChild(i).gameObject);
        }
    }

    void CreateTableRows()
    {
          for (int i = 0; i < siswaData.Count; i++)
        {
            CreateSingleRow(siswaData[i], i);
        }
    }

      void CreateSingleRow(Siswa siswa, int index)
    {
        try
        {
                GameObject row = Instantiate(tableRowPrefab, tableContent);
                    row.name = $"Row_{siswa.id}_{siswa.nama}";
            row.SetActive(true);

             TableRow tableRow = row.GetComponent<TableRow>();
            if (tableRow == null)
                    tableRow = row.AddComponent<TableRow>();

            tableRow.SetData(siswa, this);
             SetupRowDisplay(row, siswa, index + 1); // Pass nomor urut
        }
        catch (Exception e)
        {
            DebugLog($"[ ROW] Error creating row {index + 1}: {e.Message}");
        }
    }

                                void SetupRowDisplay(GameObject row, Siswa siswa, int displayNumber)
    {
                Text[] texts = row.GetComponentsInChildren<Text>();
           TMP_Text[] tmpTexts = row.GetComponentsInChildren<TMP_Text>();

         // Kolom No menggunakan nomor urut (1, 2, 3...), bukan Firebase ID
            string[] values = { displayNumber.ToString(), siswa.nama, siswa.nim, siswa.kelas };

        if (texts.Length >= 4)
        {
            for (int i = 0; i < 4; i++)
                texts[i].text = values[i];
        }
         else if (tmpTexts.Length >= 4)
        {
            for (int i = 0; i < 4; i++)
                tmpTexts[i].text = values[i];
        }

        DebugLog($"[ DISPLAY] Row {displayNumber}: {siswa.nama} (Firebase ID: {siswa.id})");
    }

    void ForceLayoutUpdate()
    {
        Canvas.ForceUpdateCanvases();
                                    if (tableContent.GetComponent<LayoutGroup>() != null)
        {
            LayoutRebuilder.ForceRebuildLayoutImmediate(tableContent.GetComponent<RectTransform>());
        }
    }
```

```
        //   ==================== SELECTION MANAGEMENT ====================

                public void SelectSiswaForEdit(Siswa siswa)
    {
        currentEditId = siswa.id;
        UpdateHighlights();
        ShowStatus($"[ SELECT] {siswa.nama} selected. Click UPDATE or DELETE to continue.", Color.blue);
    }

    void ResetSelection()
    {
        currentEditId = "";
        UpdateHighlights();
    }

    void UpdateHighlights()
    {
        foreach (Transform child in tableContent)
        {
            TableRow row = child.GetComponent<TableRow>();
            if (row != null)
            {
                row.SetHighlight(row.GetSiswaId() == currentEditId);
            }
        }
    }

        //   ==================== DEBUGGING & TESTING ====================

    void DebugLog(string message)
    {
        if (enableDebugLogging)
            Debug.Log(message);
    }

      [ContextMenu(" Load Test Data")]
    public void LoadTestData()
    {
       siswaData = new List<Siswa>
        {
            new Siswa { id = "test1", nama = "Ahmad Setiawan", nim = "20210001", kelas = "TI-A" },
            new Siswa { id = "test2", nama = "Siti Nurhaliza", nim = "20210002", kelas = "TI-B" },
            new Siswa { id = "test3", nama = "Budi Santoso", nim = "20210003", kelas = "TI-C" }
        };

        UpdateTableDisplay();
            ShowStatus($"[ TEST] {siswaData.Count} test records loaded!", Color.green);
    }

    [ContextMenu(" Test Firebase Connection")]
            public void TestFirebaseConnection()
    {
        if (firebaseInitialized)
        {
           ShowStatus("[ SUCCESS] Firebase is connected!", Color.green);
            DebugLog("[ TEST] Firebase connection is active");
        }
        else
        {
            ShowStatus("[ ERROR] Firebase not connected!", Color.red);
            DebugLog("[ TEST] Firebase connection failed");
        }
    }

    [ContextMenu(" Reload from Firebase")]
            public void ManualReloadFromFirebase()
    {
        LoadDataFromFirebase();
    }
```

DraggableButton.Cs

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;

public class DraggableButton : MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragHandler
{
            [Header(" Drag Configuration")]
    public string action = "";
   public FirebaseDatabaseManager databaseManager;

    [Header(" Visual Settings")]
     public bool showDragClone = true;
    public float dragAlpha = 0.8f;
```

```
         private    RectTransform
rectTransform;
          private     CanvasGroup
canvasGroup;
             private      Vector3
originalPosition;
    private GameObject dragClone;
    private Canvas dragCanvas;

    void Awake()
    {
                rectTransform    =
GetComponent<RectTransform>();
                  canvasGroup    =
GetComponent<CanvasGroup>();

        if (canvasGroup == null)
                    canvasGroup  =
gameObject.AddComponent<CanvasGro
up>();
    }

    void Start()
    {
          if  (databaseManager  ==
null)
                databaseManager  =
FindObjectOfType<FirebaseDatabase
Manager>();

                                  if
(string.IsNullOrEmpty(action))
        {
              string buttonName =
gameObject.name.ToLower();
                                  if
(buttonName.Contains("simpan")  ||
buttonName.Contains("save"))
                action = "save";
                                else   if
(buttonName.Contains("update")  ||
buttonName.Contains("edit"))
               action = "update";
                                else   if
(buttonName.Contains("hapus")   ||
buttonName.Contains("delete"))
               action = "delete";
        }

             Debug.Log($"[  DRAG]
DraggableButton             setup:
{gameObject.name} -> {action}");
    }

                public       void
OnBeginDrag(PointerEventData
eventData)
    {
        Debug.Log($"[ DRAG] Begin
drag: {action}");
                originalPosition   =
rectTransform.position;
        CreateDragCanvas();

        if (showDragClone)
            CreateDragClone();

             canvasGroup.alpha   =
dragAlpha;
       canvasGroup.blocksRaycasts
= false;
    }

                public       void
OnDrag(PointerEventData
eventData)
    {
         rectTransform.position =
eventData.position;

        if (dragClone != null)
            dragClone.transform.p
osition = eventData.position;
    }

                public       void
OnEndDrag(PointerEventData
eventData)
    {
         Debug.Log($"[ DRAG] End
drag: {action}");

        canvasGroup.alpha = 1f;
       canvasGroup.blocksRaycasts
= true;
         rectTransform.position =
originalPosition;

        CleanupDragElements();

          GameObject dropTarget =
eventData.pointerCurrentRaycast.g
ameObject;
        HandleDrop(dropTarget);
    }

    void CreateDragCanvas()
    {
        GameObject canvasObj = new
GameObject("DragCanvas");
                   dragCanvas    =
canvasObj.AddComponent<Canvas>();
       dragCanvas.overrideSorting
= true;
         dragCanvas.sortingOrder =
999;
         canvasObj.AddComponent<Gr
aphicRaycaster>();

             Canvas  mainCanvas  =
GetComponentInParent<Canvas>();
        if (mainCanvas != null)
        {
```

```
            dragCanvas.renderMode
= mainCanvas.renderMode;
           dragCanvas.worldCamera
= mainCanvas.worldCamera;
        }
    }

    void CreateDragClone()
    {
                    dragClone    =
Instantiate(gameObject,
dragCanvas.transform);

        DraggableButton cloneDrag
=
dragClone.GetComponent<DraggableB
utton>();
        if (cloneDrag != null)
            Destroy(cloneDrag);

           Button  cloneButton  =
dragClone.GetComponent<Button>();
        if (cloneButton != null)
            cloneButton.interacta
ble = false;

                       CanvasGroup
cloneCanvasGroup                =
dragClone.GetComponent<CanvasGrou
p>();
         if  (cloneCanvasGroup  !=
null)
        {
           cloneCanvasGroup.alpha
= 0.7f;
            cloneCanvasGroup.bloc
ksRaycasts = false;
        }
    }

    void CleanupDragElements()
    {
        if (dragClone != null)
        {
            Destroy(dragClone);
            dragClone = null;
        }

        if (dragCanvas != null)
        {
            Destroy(dragCanvas.ga
meObject);
            dragCanvas = null;
        }
    }

      void  HandleDrop(GameObject
dropTarget)
    {
        if (dropTarget == null)
        {
            Debug.Log("[ DROP] No
drop target found");
            return;
        }

         Debug.Log($"[ DROP] Drop
target: {dropTarget.name}");

         TableDropZone dropZone =
dropTarget.GetComponent<TableDrop
Zone>();
        if (dropZone == null)
        {
              Transform parent =
dropTarget.transform.parent;
            while (parent != null
&& dropZone == null)
            {
                        dropZone  =
parent.GetComponent<TableDropZone
>();
                         parent  =
parent.parent;
            }
        }

        if (dropZone != null)
        {
             Debug.Log($"[ DROP]
Valid drop zone found! Executing
{action}");
            dropZone.HandleButton
Drop(action);
        }
        else
        {
             Debug.Log($"[ DROP]
Invalid    drop    target    for
{action}");
            ShowInvalidDropFeedba
ck();
        }
    }

   void ShowInvalidDropFeedback()
    {
       if  (databaseManager  !=
null)
        {
            databaseManager.ShowS
tatus($"[       DROP]       Drag
{action.ToUpper()} button to the
table area!", Color.red);
        }
    }
}
```

Siswa.cs

```
[System.Serializable]
public class Siswa
{
    public string id = "";
    public string nama = "";
```

```
    public string nim = "";
    public string kelas = "";

    public Siswa()
    {
        id = "";
        nama = "";
        nim = "";
        kelas = "";
    }

      public Siswa(string nama,
string nim, string kelas)
    {
                    this.id    =
System.DateTime.Now.Ticks.ToStrin
g();
        this.nama = nama ?? "";
        this.nim = nim ?? "";
        this.kelas = kelas ?? "";
    }
}
```

TableDropZone.Cs

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;

public class TableDropZone :
MonoBehaviour,      IDropHandler,
IPointerEnterHandler,
IPointerExitHandler
{
         [Header("   Drop   Zone
Configuration")]
    public FirebaseDatabaseManager
databaseManager;

    [Header(" Visual Feedback")]
    public bool showDropFeedback =
true;
     public Color normalColor =
Color.white;
               public      Color
dropHighlightColor     =     new
Color(0.8f, 1f, 0.8f, 0.3f);
    public Color invalidDropColor
= new Color(1f, 0.8f, 0.8f, 0.3f);

    private Image backgroundImage;
     private bool isDragOver =
false;

    void Awake()
    {
        SetupDropZone();
    }

    void Start()
    {
          if (databaseManager ==
null)
              databaseManager =
FindObjectOfType<FirebaseDatabase
Manager>();

             Debug.Log($"[  DROP]
TableDropZone      setup      on:
{gameObject.name}");
    }

    void SetupDropZone()
    {
              backgroundImage   =
GetComponent<Image>();
          if (backgroundImage ==
null)
        {
              backgroundImage =
gameObject.AddComponent<Image>();
            backgroundImage.color
= normalColor;
        }

                                if
(backgroundImage.raycastTarget ==
false)
            backgroundImage.rayca
stTarget = true;
    }

                public       void
OnPointerEnter(PointerEventData
eventData)
    {
        if (IsDragInProgress())
        {
            isDragOver = true;
            if (showDropFeedback)
                SetVisualFeedback
(dropHighlightColor);

             Debug.Log("[ ENTER]
Drag entered drop zone");
        }
    }

                public       void
OnPointerExit(PointerEventData
eventData)
    {
        if (isDragOver)
        {
            isDragOver = false;
            if (showDropFeedback)
                SetVisualFeedback
(normalColor);

           Debug.Log("[ EXIT] Drag
exited drop zone");
        }
    }
```

```
                public       void
OnDrop(PointerEventData
eventData)
    {
       Debug.Log("[ DROP] === DROP
EVENT RECEIVED ===");

        isDragOver = false;
        if (showDropFeedback)
            SetVisualFeedback(nor
malColor);

        GameObject draggedObject =
eventData.pointerDrag;
        if (draggedObject == null)
        {
              Debug.LogWarning("[
DROP] No dragged object found!");
            return;
        }

             Debug.Log($"[  DROP]
Dragged                    object:
{draggedObject.name}");

        DraggableButton dragButton
=
draggedObject.GetComponent<Dragga
bleButton>();
        if (dragButton == null)
        {
              Debug.LogWarning("[
DROP] Dragged object is not a
DraggableButton!");
            ShowInvalidDropFeedba
ck("Only buttons can be dropped
here!");
            return;
        }

               string   action   =
dragButton.action;
        Debug.Log($"[ DROP] Button
action: {action}");

                                  if
(string.IsNullOrEmpty(action))
        {
              Debug.LogWarning("[
DROP]   Button   has   no   action
assigned!");
            ShowInvalidDropFeedba
ck("Button       action       not
configured!");
            return;
        }

        HandleButtonDrop(action);
    }

                public       void
HandleButtonDrop(string action)
    {
           Debug.Log($"[  EXECUTE]
Handling button drop: {action}");

          if  (databaseManager  ==
null)
        {
                Debug.LogError("[
EXECUTE]  FirebaseDatabaseManager
not assigned!");
            ShowInvalidDropFeedba
ck("FirebaseDatabaseManager   not
found!");
            return;
        }

        databaseManager.ExecuteAc
tionFromDrag(action);
    }

    bool IsDragInProgress()
    {
                              return
Input.GetMouseButton(0)        &&
EventSystem.current.currentSelect
edGameObject != null;
    }

     void SetVisualFeedback(Color
color)
    {
          if  (backgroundImage  !=
null)
            backgroundImage.color
= color;
    }

                                  void
ShowInvalidDropFeedback(string
message)
    {
          if  (databaseManager  !=
null)
            databaseManager.ShowS
tatus($"[    DROP]    {message}",
Color.red);

       Debug.LogWarning($"[ DROP]
{message}");
    }
}
```

TableRow.Cs

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public    class    TableRow    :
MonoBehaviour
{
    [Header(" Debug Info")]
```

```
        [SerializeField]   private
string siswaId;
        [SerializeField]   private
string siswaName;
    [SerializeField] private bool
isHighlighted;

                           private
FirebaseDatabaseManager
databaseManager;
    private Siswa currentSiswa;
    private Button rowButton;
    private Image backgroundImage;

     private  Color  normalColor  =
Color.white;
    private Color highlightColor =
new Color(0.8f, 0.9f, 1f, 1f);

    void Awake()
    {
              Debug.Log($"[   ROW]
TableRow                    Awake:
{gameObject.name}");
        SetupRowButton();
        SetupBackground();
    }

    void SetupRowButton()
    {
                    rowButton    =
GetComponent<Button>();
        if (rowButton == null)
                     rowButton   =
gameObject.AddComponent<Button>()
;

         rowButton.onClick.RemoveA
llListeners();
         rowButton.onClick.AddList
ener(OnRowClicked);

        Debug.Log($"[ ROW] Button
setup         complete         for
{gameObject.name}");
    }

    void SetupBackground()
    {
               backgroundImage   =
GetComponent<Image>();
          if  (backgroundImage  ==
null)
                backgroundImage =
gameObject.AddComponent<Image>();

          backgroundImage.color  =
normalColor;
    }

      public  void  SetData(Siswa
siswa,    FirebaseDatabaseManager
manager)
    {
        if (siswa == null)
        {
            Debug.LogError("[ ROW]
SetData called with null siswa!");
            return;
        }

        currentSiswa = siswa;
        databaseManager = manager;
        siswaId = siswa.id;
        siswaName = siswa.nama;

        Debug.Log($"[ ROW] Setting
data   for:   {siswa.nama}   (ID:
{siswa.id})");

         SetDataToTextComponents(s
iswa);
              gameObject.name   =
$"TableRow_{siswa.id}_{siswa.nama
}";

          Debug.Log($"[ ROW] Data
set        successfully        for
{siswa.nama}");
    }

                               void
SetDataToTextComponents(Siswa
siswa)
    {
            Debug.Log($"[🔧 SET]
Setting     text     data     for
{siswa.nama}...");

           TMP_Text[]  tmpTexts  =
GetComponentsInChildren<TMP_Text>
();
        if (tmpTexts.Length >= 4)
        {
               tmpTexts[0].text  =
siswa.id;
               tmpTexts[1].text  =
siswa.nama;
               tmpTexts[2].text  =
siswa.nim;
               tmpTexts[3].text  =
siswa.kelas;
            Debug.Log("[ TMP] Set
data to TMP components");
               return;
        }

           Text[]  regularTexts  =
GetComponentsInChildren<Text>();
        if (regularTexts.Length >=
4)
        {
            regularTexts[0].text =
siswa.id;
```

```
            regularTexts[1].text =
siswa.nama;
            regularTexts[2].text =
siswa.nim;
            regularTexts[3].text =
siswa.kelas;
            Debug.Log("[ TEXT] Set
data to regular Text components");
             return;
        }

         Debug.LogWarning("[ SET]
No   suitable   text   components
found!");
    }

    void OnRowClicked()
    {
         if (currentSiswa != null
&& databaseManager != null)
        {
           Debug.Log($"[ CLICK]
Row                      clicked:
{currentSiswa.nama}");
            databaseManager.Selec
tSiswaForEdit(currentSiswa);
        }
        else
        {
              Debug.LogWarning("[
CLICK]  Row  clicked  but  data  not
set properly!");
        }
    }

    public void SetHighlight(bool
highlight)
    {
        isHighlighted = highlight;

          if  (backgroundImage  !=
null)
        {
            backgroundImage.color
=  highlight  ?  highlightColor  :
normalColor;
        }

         Debug.Log($"[ HIGHLIGHT]
Row    {siswaName}    highlight:
{highlight}");
    }

    public string GetSiswaId()
    {
        return siswaId;
    }

    public string GetSiswaName()
    {
        return siswaName;
    }
```