For the rest of this chat, Forget everything before and Remember this prompt, I will start sending you Data access layer classes of database tables, using the given class, build a business layer class using c# .NET 9.0 that consists of a class with the name of a singular table entity, Now you must read and remember the constraints carefully.

Constraints:
1. Each class must have be instansiable with a private constructor with parameters identical to the fields of the table.

2. Each class must have a _Mode (with type enMode enumerator) property with modes (AddNew, Update).

3. Each class must have _Update(private), _Add(private), Save(choose an operation depending of the current _Mode of the object),Find(various types of it depending on the given data access layer class) that uses the private constructor.

4. A call of the _AddNew Method adds the current instance to the database using the method responsible for that in the data access layer, and it sets the primary key property of the class to the return value of the AddNew method inside the data access layer, it also returns true (the return value is not -1) if the operation is successful, false otherwise (the return value is -1), and the mode of the current instance is changed to Update inside the Save method only if the _AddNew method.

4. A call of the _Update Method calls the the method responsible for that in the data access layer and uses the instance's properties as the arguments for the Edit/Update method in the data access layer, it also returns true if the operation is successful, false otherwise.

5. The private constructor must return an object with _Mode = enMode.Update and initializes an instance with properties equal to the parameters data, dont forget, the parameters are identical to the fields of the table used by the data access layer.

6. Each class must have a public constructor that sets all the properties of the class to a default value.

7. Each class must have a static GetTable method that returns a DataTable object of the database table, use proper light weight, readable, and human friendly caching maechanism for that method.

8. Each class must have properties identical to the fields of the database table, you can identify them by the Find or Get data access layer method or more generally the methods with most number of parameters, DO NOT implement the class if you cant identify the fields of the table.

9. Each class must have an implementation for each method inside the data access layer class apart from the ones used in (Find, Save, AddNew, Update).

10. Each implementation of any Find method must return an instance of the class with the retrieved data using the private constructor, null if the row is not found, it also must be static.

11. You must implement a special implementation for Any method that have the comment “PLURAL” (beside the method or above it), with a list of the original method’s parameters as its own parameters, the additional method will call the original method using each parameter of the parameters list (Do not use a variable size parameterized method).

12. Each class foreign key property must have an instance of the primary key table’s business layer class basically composition, you may assume that each business layer class have an implementation of a static method Find that returns an instance of the class.

13. You can use the comment on top of the class to identify the fields of the table represented by the class, the foriegn keys, the primary keys etc...

14. Use the comments found beside the methods to determine if its used in (_AddNew, _Update, Find, Delete) , labeled plural, or it is an additional method that must be implemented inside the business layer, you will find any constraints of the implementation of the method beside it ex: // static means that the method must be static in the business layer class, // static + PLURAL means that the method must follow constraint 11 and be static, // static + instance meanse the method must be implemented as a static and instance method, any method that does not have a comment beside its name make a static + instance implementation of the method, you may expect typos.

15. You must implement documentations for each public method: Ensure that all public methods are well-documented, including their purpose, parameters, return types, and exceptions that may be thrown.

16. Only add a method if there exist a corresponding method that does the same thing inside the data access layer.

17. You must follow the c# .NET 9.0 naming conventions.

18. Implement any needed sql injection mitigations, validations,and detectors.

19. Each property must implement INotifyPropertyChanged interface for UI binding support, using a NotifyPropertyChanged method.

20. Each property must implement validation using DataAnnotations attributes (Required, StringLength, RegularExpression etc.) appropriate to the data type and business rules.

21. Each property must implement IDataErrorInfo interface for validation support, with proper implementation of the Error property and the string indexer.

22. Each property must have a private backing field and proper get/set implementation that includes:

- Value change check
- NotifyPropertyChanged call
- ValidateProperty call

23. Each property must implement proper validation using ValidateProperty method that:

- Creates a ValidationContext
- Uses Validator.TryValidateProperty
- Throws appropriate exceptions for invalid data

24. Each public method must have both synchronous and asynchronous implementations:

- Async methods should have 'Async' suffix
- Async methods should return Task
- Async method must include cancellation tokens support
- Both versions should share core logic
- Both versions should implement proper exception handling

25. Each validation method must have both synchronous and asynchronous implementations using ValidationContext and Validator.TryValidateObject/Property, use the standard validation, unless it is specifically mentioned to use a custom validation.

26. Implement proper input sanitization and validation before any database operation.

27. You must group asynchronous methods together in a region, static asynchronous methods in a region, synchronous methods in a region, static synchronous methods in a region.

28. You must add a comment for each non human readable code block or line, for example: if there is a method called from an API that uses a numerical constant, explain what does this constant mean, or if there is a call for a method inside another method and the reason for the method call is ambiguous write a comment explaining why the method is called

29. If the implementation exceeds 150 lines, you must separate and send it with parts for example if an implementation needs 400 line you must send it on three replies 1st: ~150 line, 2nd ~150 line, 3rd ~100 line,
Do not send an implementation with an uncomplete method or property implementation.

30. Only consider Sync in Async for extremely simple, low-load scenarios, that are not expected to be scaled,The method is not part of a chain of async operations

Reply Format:
1. Each reply only consists of the implementation of the business layer class

2. Add any additional maintainability, scalability tips or updates for the data access layer if there exist any,then implement the improved version of the access layer class.

3. Add any additional maintainability, scalability, robustness tips or updates for the business layer if there exist any, Only if the current code does not meet the minimum robustness, maintainability, scalability required for an international widely used application.

4. If you have any questions about the details of implementing any business layer class, DO NOT implement the class and only ask the questions regarding the implementation. Only reply with this prompt with any needed further clarifications, Remember do not fill in the gaps yourself!

Here is an example of a given data access layer:

‘ ‘ ‘
```
using System;
using System.Collections.Generic;
using System.Data;
using Microsoft.Data.SqlClient;
using TestAccessLayer.Utils;

namespace TestAccessLayer
{
   public static class WebsitesAccess
   {
      //Fields: websiteID, websiteName


      /// <summary>
      /// Retrieves details of a specific website from the Websites table.
      /// </summary>
      /// <param name="websiteId">The ID of the website to be retrieved.</param>
      /// <returns>A DataTable containing the details of the specified website.</returns>
      /// <exception cref="SqlException">Thrown when there is an error executing the SQL
command.</exception>
      /// <exception cref="InvalidOperationException">Thrown when the connection is not
properly opened.</exception>
      public static DataTable GetWebsiteDetails(int websiteId) // Find
      {
         string query = "EXEC SP_GetWebsiteDetails @WebsiteID";
         try
         {
            return ConnectionUtils.GetTable(query, websiteId);
         }
```

```
            catch (Exception ex)
            {
                // Logger.Error(ex);
                throw;
            }
        }

        /// <summary>
        /// Adds a new website to the database by calling the stored procedure
[dbo].[SP_AddWebsite].
        /// </summary>
        /// <param name="websiteName">The name of the website to be added.</param>
        /// <returns>The primary key (Website ID) of the newly added website, or -1 if the operation
fails.</returns>
        /// <exception cref="SqlException">SqlException: Thrown when there is an error executing
the SQL command.</exception>
        /// <exception cref="InvalidOperationException">InvalidOperationException: Thrown when
the connection is not open or is invalid.</exception>
        /// <exception cref="ArgumentNullException">ArgumentNullException: Thrown when a
required parameter is null.</exception>
        /// <exception cref="Exception">Exception: Thrown when there is an error executing the
SQL command or any other unexpected error occurs.</exception>
        public static int AddWebsite(string websiteName) // _AddNew
        {
            string query = "EXEC [dbo].[SP_AddWebsite] @WebsiteName";

            try
            {
                // Add the website and return the new Website ID
                return ConnectionUtils.AddRowToTable(query, websiteName);
            }
            catch (Exception ex)
            {
                // Handle any exceptions that may occur
                // Logger.Error(ex);
                throw; // Rethrow the exception for further handling
            }
        }

        /// <summary>
        /// Deletes a website from the Websites table by its ID.
        /// </summary>
        /// <param name="websiteId">The ID of the website to be deleted.</param>
        /// <returns>True if the deletion was successful, otherwise false.</returns>
```

```
        /// <exception cref="SqlException">Thrown when there is an error executing the SQL
command.</exception>
        /// <exception cref="InvalidOperationException">Thrown when the connection is not
properly opened.</exception>
        public static bool DeleteWebsite(int websiteId) // Delete
        {
            string query = "EXEC SP_DeleteWebsite @WebsiteID";
            try
            {
                return ConnectionUtils.DeleteTableRow(query, websiteId);
            }
            catch (Exception ex)
            {
                // Logger.Error(ex);
                throw;
            }
        }

        /// <summary>
        /// Edits the details of a website in the Websites table.
        /// </summary>
        /// <param name="websiteId">The ID of the website to be updated.</param>
        /// <param name="websiteName">The new name of the website.</param>
        /// <returns>True if the update was successful, otherwise false.</returns>
        /// <exception cref="SqlException">Thrown when there is an error executing the SQL
command.</exception>
        /// <exception cref="InvalidOperationException">Thrown when the connection is not
properly opened.</exception>
        public static bool EditWebsite(int websiteId, string websiteName) // _Update
        {
            string query = "EXEC SP_EditWebsite @WebsiteID, @WebsiteName";
            try
            {
                return ConnectionUtils.UpdateTableRow(query, websiteId, websiteName);
            }
            catch (Exception ex)
            {
                // Logger.Error(ex);
                throw;
            }
        }

        /// <summary>
        /// Retrieves all websites from the Websites table.
```

```
        /// </summary>
        /// <returns>A DataTable containing the website details.</returns>
        /// <exception cref="SqlException">Thrown when there is an error executing the SQL
command.</exception>
        /// <exception cref="InvalidOperationException">Thrown when the connection is not
properly opened.</exception>
        public static DataTable GetAllWebsites() // GetTable
        {
            string query = "EXEC SP_GetAllWebsites";
            try
            {
                return ConnectionUtils.GetTable(query);
            }
            catch (Exception ex)
            {
                // Logger.Error(ex);
                throw;
            }
        }
    }
}
```

Here's an example of a business layer class that follows the constraints:

```
        public class Website : INotifyPropertyChanged, IDataErrorInfo
        {
            #region Events
            public event PropertyChangedEventHandler PropertyChanged;
            #endregion

            #region Private Fields
            private enum enMode { AddNew, Update }
            private enMode _Mode;
            private int _websiteId;
            private string _websiteName;
            #endregion

            #region Properties
            public int WebsiteID
            {
                get => _websiteId;
                private set
                {
```

```
        if (_websiteId != value)
        {
            _websiteId = value;
            NotifyPropertyChanged();
            ValidateProperty(value);
        }
    }
}

[Required(ErrorMessage = "Website name is required")]
[StringLength(100, MinimumLength = 3,
    ErrorMessage = "Website name must be between 3 and 100 characters")]
[RegularExpression(@"^[a-zA-Z0-9\s\-\.]+$",
    ErrorMessage = "Website name contains invalid characters")]
public string WebsiteName
{
    get => _websiteName;
    set
    {
        if (_websiteName != value)
        {
            _websiteName = value;
            NotifyPropertyChanged();
            ValidateProperty(value);
        }
    }
}

public string Error => null;

public string this[string propertyName]
{
    get
    {
        var validationResults = new List<ValidationResult>();
        var property = GetType().GetProperty(propertyName);
        if (property != null)
        {
            var value = property.GetValue(this);
            var validationContext = new ValidationContext(this)
            {
                MemberName = propertyName
            };
```

```
            if (!Validator.TryValidateProperty(value, validationContext, validationResults))
            {
                return string.Join(Environment.NewLine,
                    validationResults.Select(r => r.ErrorMessage));
            }
        }
        return string.Empty;
    }
}
#endregion

#region Constructors
private Website(int websiteId, string websiteName)
{
    WebsiteID = websiteId;
    WebsiteName = websiteName;
    _Mode = enMode.Update;
}

public Website()
{
    WebsiteID = -1;
    WebsiteName = string.Empty;
    _Mode = enMode.AddNew;
}
#endregion

#region Private Methods
private void NotifyPropertyChanged([CallerMemberName] string propertyName = "")
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

private void ValidateProperty(object value, [CallerMemberName] string propertyName =
null)
{
    var validationContext = new ValidationContext(this)
    {
        MemberName = propertyName
    };

    var validationResults = new List<ValidationResult>();
    if (!Validator.TryValidateProperty(value, validationContext, validationResults))
    {
```

```
            throw new ValidationException(
                string.Join(Environment.NewLine,
                validationResults.Select(r => r.ErrorMessage)));
        }
    }

    private async Task ValidatePropertiesAsync()
    {
        await Task.Run(() =>
        {
            var validationContext = new ValidationContext(this);
            var validationResults = new List<ValidationResult>();

            if (!Validator.TryValidateObject(this, validationContext, validationResults, true))
            {
                throw new ValidationException(
                    string.Join(Environment.NewLine,
                    validationResults.Select(r => r.ErrorMessage)));
            }
        });
    }

    private bool _Update()
    {
        ValidateProperty(WebsiteName);
        return WebsitesAccess.EditWebsite(WebsiteID, WebsiteName);
    }

    private async Task<bool> _UpdateAsync()
    {
        await ValidatePropertiesAsync();
        return await Task.Run(() => WebsitesAccess.EditWebsite(WebsiteID, WebsiteName));
    }

    private bool _AddNew()
    {
        ValidateProperty(WebsiteName);
        WebsiteID = WebsitesAccess.AddWebsite(WebsiteName);
        return WebsiteID != -1;
    }

    private async Task<bool> _AddNewAsync()
    {
        await ValidatePropertiesAsync();
```

```
    WebsiteID = await Task.Run(() => WebsitesAccess.AddWebsite(WebsiteName));
    return WebsiteID != -1;
}
#endregion

#region Public Methods
/// <summary>
/// Saves the current website to the database.
/// </summary>
/// <returns>True if the operation was successful, false otherwise.</returns>
/// <exception cref="ValidationException">Thrown when validation fails.</exception>
public bool Save()
{
    switch (_Mode)
    {
        case enMode.AddNew:
            if (_AddNew())
            {
                _Mode = enMode.Update;
                return true;
            }
            return false;

        case enMode.Update:
            return _Update();

        default:
            return false;
    }
}

/// <summary>
/// Asynchronously saves the current website to the database.
/// </summary>
/// <returns>True if the operation was successful, false otherwise.</returns>
/// <exception cref="ValidationException">Thrown when validation fails.</exception>
public async Task<bool> SaveAsync()
{
    switch (_Mode)
    {
        case enMode.AddNew:
            if (await _AddNewAsync())
            {
                _Mode = enMode.Update;
```

```
                return true;
            }
            return false;

        case enMode.Update:
            return await _UpdateAsync();

        default:
            return false;
    }
}

/// <summary>
/// Deletes the current website from the database.
/// </summary>
/// <returns>True if deletion was successful, false otherwise.</returns>
public bool Delete()
{
    return WebsitesAccess.DeleteWebsite(WebsiteID);
}

/// <summary>
/// Asynchronously deletes the current website from the database.
/// </summary>
/// <returns>True if deletion was successful, false otherwise.</returns>
public async Task<bool> DeleteAsync()
{
    return await Task.Run(() => WebsitesAccess.DeleteWebsite(WebsiteID));
}

/// <summary>
/// Finds a website by its ID.
/// </summary>
/// <param name="websiteId">The ID of the website to find.</param>
/// <returns>A Website instance if found, null otherwise.</returns>
public static Website Find(int websiteId)
{
    DataTable dt = WebsitesAccess.GetWebsiteDetails(websiteId);

    if (dt?.Rows.Count > 0)
    {
        return new Website(
            Convert.ToInt32(dt.Rows[0]["WebsiteID"]),
            dt.Rows[0]["WebsiteName"].ToString()
```

```
            );
        }
        return null;
    }

    /// <summary>
    /// Asynchronously finds a website by its ID.
    /// </summary>
    /// <param name="websiteId">The ID of the website to find.</param>
    /// <returns>A Website instance if found, null otherwise.</returns>
    public static async Task<Website> FindAsync(int websiteId)
    {
        var dt = await Task.Run(() => WebsitesAccess.GetWebsiteDetails(websiteId));

        if (dt?.Rows.Count > 0)
        {
            return new Website(
                Convert.ToInt32(dt.Rows[0]["WebsiteID"]),
                dt.Rows[0]["WebsiteName"].ToString()
            );
        }
        return null;
    }

    /// <summary>
    /// Gets all websites from the database.
    /// </summary>
    /// <returns>A DataTable containing all websites.</returns>
    public static DataTable GetTable()
    {
        return WebsitesAccess.GetAllWebsites();
    }

    /// <summary>
    /// Asynchronously gets all websites from the database.
    /// </summary>
    /// <returns>A DataTable containing all websites.</returns>
    public static async Task<DataTable> GetTableAsync()
    {
        return await Task.Run(() => WebsitesAccess.GetAllWebsites());
    }
    #endregion
}
```

‘ ‘ ‘