Dynamics CRM provides flexibility of customisation & configuration both as Microsoft market CRM as XRM (Extended relationship Management) but it’s always best to use configuration over customisation.Dynamics 365 Customer inherits Dataverse extensibility model for extension which need development activities.

A diagram of a software development

Description automatically generated

Dataverse extensibility Model

As per above extensibility model core code component involves C#.Net and Javascript so developer need to follow C# best practices and Javascript best practices along with Entity and core CRM best practices.

**C# best practices:**

* Use consistent naming conventions and casing for classes, methods, variables, and parameters. Follow the Microsoft guidelines for C# naming conventions.
* Use meaningful and descriptive names that indicate the purpose and functionality of the code components.
* Use comments and XML documentation to explain the logic and intent of the code. Avoid commenting obvious or redundant code.
* Use regions to group related code blocks and collapse them when not needed. This improves the readability and maintainability of the code.
* Use the using statement to ensure that resources such as connections, files, or streams are properly disposed after use. Avoid using the finalize method for resource cleanup.
* Use exception handling to handle errors and unexpected situations gracefully. Use try-catch-finally blocks to catch and handle different types of exceptions. Use the throw keyword to rethrow an exception without losing the stack trace. Use the finally block to execute any code that must run regardless of whether an exception occurs or not.
* Use LINQ (Language Integrated Query) to query and manipulate data from various sources, such as entities, collections, XML, or databases. LINQ provides a concise and expressive way to perform common data operations, such as filtering, sorting, grouping, or aggregating.
* Use async and await keywords to write asynchronous code that does not block the main thread. Asynchronous code can improve the performance and responsiveness of the application, especially when dealing with long-running or IO-bound tasks. Use the Task and Task classes to represent asynchronous operations and their results.
* Use generics to write type-safe and reusable code that can work with different types of data. Generics can reduce code duplication and avoid casting or boxing operations that can affect performance. Use generic constraints to specify the types that can be used as type parameters for a generic class or method.
* Use dependency injection to decouple the code components and make them easier to test and maintain. Dependency injection is a design pattern that allows a class to receive its dependencies from an external source, rather than creating them internally. This reduces the coupling between the classes and improves the modularity and flexibility of the code.

**Javascript best practices:**

* Use descriptive and consistent naming for variables, functions, and classes. Avoid using abbreviations, acronyms, or generic names that could cause confusion or ambiguity. Follow the camelCase convention for variables and functions, and the PascalCase convention for classes.
* Use comments and documentation to explain the purpose and functionality of your code. Comments should be concise and clear, and follow the JSDoc format for generating documentation. Documentation should provide an overview of the structure, design, and usage of your code, as well as examples and references.
* Use modular and reusable code to avoid duplication and improve readability. Modular code is organized into smaller units that perform specific tasks and can be easily combined or reused. Reusable code is written in a way that allows it to be used in different contexts and scenarios without requiring changes.
* Use consistent indentation and spacing to format your code and improve readability. Indent your code using two spaces per level, and use spaces around operators, keywords, and commas. Avoid unnecessary whitespace or empty lines.
* Use semicolons to end statements and avoid errors related to automatic semicolon insertion. Although JavaScript allows omitting semicolons in some cases, it can also lead to unexpected results or bugs.
* Use const and let keywords to declare variables and avoid using var. Const and let provide block-level scope and prevent variable hoisting, which can cause confusion and errors. Const also prevents reassignment of variables, which can help maintain code integrity.
* Use arrow functions to write concise and expressive code, especially for callbacks and anonymous functions. Arrow functions have a shorter syntax than regular functions and can implicitly return values. They also inherit the this value from their enclosing context, which can avoid problems with binding or losing the this value.
* Use the Xrm.Page and Xrm.Utility namespaces to access the form context and the utility methods of the Dynamics 365 CRM JavaScript API. You can use these namespaces to perform actions such as getting or setting field values, showing or hiding fields, registering or unregistering event handlers, opening web resources or dialogs, and retrieving the current user information.
* Use the Web API to perform CRUD (create, read, update, delete) operations on the Dynamics 365 data and metadata. You can use the XMLHttpRequest object or the jQuery.ajax method to send HTTP requests and receive HTTP responses. You can also use the OData query parameters to filter, sort, expand, or select the data that you want to retrieve or update.
* Use the asynchronous programming pattern to handle the callbacks of your Web API requests. You can use the standard callback functions or the Promise objects to handle the success or failure of your requests. You can also use the async and await keywords to write asynchronous code in a synchronous-like manner.
* Use the JavaScript coding standards and best practices to write clean, readable, and maintainable code. You can use tools such as ESLint, JSHint, or JSLint to check your code quality and style. You can also use tools such as JSBeautifier, Prettier, or Visual Studio Code to format your code consistently and automatically.
* Use the strict mode ("use strict") to avoid common errors and enforce best practices in your JavaScript code. The strict mode prevents the use of undeclared variables, duplicate parameters, octal literals, and other potential bugs.
* Use the JavaScript naming conventions to make your code more readable and consistent. The naming conventions suggest using camelCase for variables and functions, PascalCase for constructors and classes, UPPER\_CASE for constants, and \_underscore for private properties and methods.
* Use the modular pattern to organize your code into reusable and maintainable modules. The modular pattern allows you to create private and public scopes for your variables and functions, and avoid polluting the global namespace. You can use the IIFE (immediately invoked function expression) syntax to create a module and expose its public members as an object.
* Use the addOnChange, addOnLoad, and addOnSave methods of the Xrm.Page.data.entity object to register your event handlers for the form events. These methods allow you to specify the execution context, the handler function, and the dependencies for your event handlers. You can also use the removeOnChange, removeOnLoad, and removeOnSave methods to unregister your event handlers when they are no longer needed.

**Plugin/Workflow Development Best Practices:**

* Use the IPluginExecutionContext interface to access the context information of the plugin execution, such as the input and output parameters, the target entity, the stage, the mode, and the user identity.
* Use the IOrganizationServiceFactory interface to create an instance of the IOrganizationService interface, which allows you to perform CRUD operations on the entities and metadata of your Dynamics 365 system.
* Use the tracing service to log any information or errors that occur during the plugin execution. You can use the ITracingService interface to write messages to the trace log, which can help you debug and troubleshoot your plugin code.
* Use the PluginRegistration tool to register your plugin assemblies and steps in your Dynamics 365 system. You can use this tool to specify the message, entity, stage, mode, filtering attributes, pre-image, post-image, and other properties of your plugin steps.
* Use exception handling to handle any errors or exceptions that may occur in your plugin code. You can use the InvalidPluginExecutionException class to throw an exception that will be displayed to the user and logged in the trace log. You can also use the try-catch-finally blocks to handle different types of exceptions and perform cleanup actions if needed.
* Use the sandbox mode to isolate your plugin code from the main Dynamics 365 process and prevent any potential security risks or performance issues. The sandbox mode limits the access to external resources and enforces a time limit for the plugin execution. You can use the IsolationMode property of the PluginStep class to specify the sandbox mode for your plugin steps.
* Implement the IPlugin interface to create your plugin class and define the Execute method, which contains the logic of your plugin code. You can use the ServiceProvider parameter of the Execute method to access the services that are available to your plugin, such as the context, the organization service, and the tracing service.
* Implement the constructor of your plugin class to pass the secure and unsecure configuration data that you specified when registering your plugin. You can use the SecureConfig and UnsecureConfig parameters of the constructor to access the configuration data, which can be used to store any custom settings or parameters for your plugin.
* Use the Entity and EntityReference classes to represent the records and references of the entities that are involved in your plugin execution. You can use the Attributes and FormattedValues properties of the Entity class to access the attribute values and formatted values of the entity record. You can also use the LogicalName and Id properties of the EntityReference class to access the logical name and identifier of the entity reference.
* Use the OptionSetValue and Money classes to represent the option set and money attribute values of the entity record. You can use the Value property of the OptionSetValue class to access the integer value of the option set. You can also use the Value property of the Money class to access the decimal value of the money.
* Use the QueryExpression, FetchExpression, and QueryByAttribute classes to query the data and metadata of your Dynamics 365 system. You can use these classes to specify the criteria, filters>, columns, orders, and other properties of your query. You can also use the Retrieve and RetrieveMultiple methods of the IOrganizationService interface to execute your query and return the results as an Entity or an EntityCollection object.

**Azure Function coding best practices**

* Use the DependencyInjection and ExecutionContext classes to register and resolve the dependencies of your function app. You can use the ConfigureServices method of the DependencyInjection class to configure the services and interfaces that your function app requires. You can also use the GetService or GetRequiredService methods of the ExecutionContext class to get an instance of the service or interface that you need.
* Use the async and await keywords to make your function code asynchronous and improve the performance and scalability of your function app. You can use the async modifier to declare an asynchronous function and the await operator to wait for the completion of an asynchronous task. You can also use the Task and Task classes to represent the asynchronous operations and their results.
* Use the HttpClientFactory and Polly libraries to create and manage HttpClient instances and implement resilient HTTP requests. You can use the AddHttpClient method of the DependencyInjection class to register and configure the HttpClient instances that your function app needs. You can also use the AddPolicyHandler method of the IHttpClientBuilder interface to apply policies such as retries, timeouts, circuit breakers, and fallbacks to your HTTP requests.
* Use the ConfigurationBuilder and IConfigurationRoot classes to build and access the configuration settings of your function app. You can use the AddEnvironmentVariables, AddJsonFile, AddAzureKeyVault, and other extension methods of the ConfigurationBuilder class to load the configuration sources from different locations. You can also use the GetSection, GetChildren, GetConnectionString, and GetValue methods of the IConfigurationRoot interface to read the configuration values from different sections and levels.

**Entity customization & general best practices**

* Follow below practices for better customisation –
  + Publisher-
    - Always create a proper publisher of the instance identifying the deployment organisation e.g. Organisation name is Cadbury then publisher should be in a such a way that it’s easily identifiable e.g*. cdbry\_* instead of default *new\_*
  + Fields -
    - Each field creating for the entity should be properly named so that it’s easy to identify including its schema name e.g. field name is *SSN No* schema name could be *cdbry\_ssnNo*
    - If field is local option set, then naming convention should contain keyword OptionSet e.g. for Channel option set Schema name could be *cdbry\_ChannelOptionSet*
    - If the key to the options is large number then change it to easily identifiable number e.g. Channel OptionSet Key system generated number is 3571234 pls change it to 00 and next number to 01, next number of the key to 02 sequentially.
    - If any option set value having high chance to use in number of entities e.g. Countries, then make it Global Option Set
    - If field is global option set, then naming convention should contain keyword Options e.g. for Countries option set Schema name could be *cdbry\_Countries*
    - If any field is of two options e.g. True/False, Yes/No then the name should contain Is e.g. cdbry\_IsTrue
    - If any field is of image type, then naming should contain img word e.g. for Person Image field schema name should be cdbry\_ImgPerson
    - If any look up field is there pls name, its property appending ID keyword e.g. Contact lookup is there then name it to cdbry\_ContactID
  + Use custom attribute not entities.
    - Always focus to save server space. Use existing entity and add custom attributes to achieve a specific task as much as possible.
    - Rename existing entity to make the entities more meaningful to fit the business requirement.
  + Use Meaningful attributes & entities.
    - Create custom attributes with meaningful Display Name and Schema Name.
    - Avoid changing the Form label of attributes very frequently.
    - Keep Display Name, Schema Name, Form Label Name & Logical Name same.
  + Use searchable & requirement level wisely.
    - While adding new custom attribute do not leave the searchable and requirement level property as default. Set these values as per proper business requirement.
    - Set the field’s Searchable file as “NO” if you don’t want to show this field in Advance Find Query.
    - While creating fields mention a meaningful description for all fields without being lazy.
  + Use Existing Entities to avail built-in functionality.
    - Customize a system entity, such as the opportunity entity, instead of replacing it with a new custom entity so that you can use the many built-in features in an existing entity.
    - For example, the opportunity and case entities have lookup fields to associate customers. Customers may be accounts or contacts. You cannot create a custom entity that has the same type of lookup. You can change the display name of a system entity to make it more meaningful to your business.
  + Don’t customize default solution.
    - Using “Solution” we can package all customizations which gives a flexibility in distributing the customization to other environment or to marketplace or to other organizations.
    - Always create custom solution and add all components to that solution.
    - Do not customize the default solution. Decide how to manage the custom solution and export it as Managed or Un-managed solution.
  + Don’t customize directly in Production.
    - As a software development best practice there should be different environment for solution deployment like: Development (DEV), System Testing (ST), System Integration Testing (SIT), User Acceptance Test (UAT), Training, Pre-Production (PRE-PROD) and Production (PROD).
    - Do all development activities, customizations in DEV environment. Never ever customize the CRM directly in production.
    - This is a quick intention to change directly in target environment for a quick fix but please avoid this activity.
  + Decide when to use Plugins vs Workflows.
    - There are various ways to customize a CRM system. Like Java script, Plugins, Custom Work-flow Activities, custom web pages etc.
    - When to use a Plugin or workflow depends on the business requirement and the customizer as well.
    - If you want to execute custom code immediately before or after the core platform operation executes and before the result of the operation is returned from the platform, then you must use a synchronous plug-in or real-time workflow. You cannot use an asynchronous workflow or asynchronous plug-in in this situation because they are queued to execute after the core operation finishes executing. So, you cannot predict when they will run.
    - Plugins are targeted to run within 2 minutes otherwise it will throw time out exception rolling back the functionality. So here for this case also we need to very sure to use correct way of process.
    - Analyse these techniques and choose the one that best suits your business objectives after you consider the deployment, performance, and maintenance concerns of your plug-in or workflow solution.
    - Check out the characteristics and differences between Plug-in and Workflow to decide which is the best option for your requirement.
  + Use single workflow instead of Multiple child workflow.
    - The child workflow approach achieves lower throughput, but it is more manageable if you frequently change your workflow definition. Compilation overhead is not a major concern because the workflow is compiled only during publishing.
    - However, Microsoft Dynamics CRM incurs overhead when it starts each workflow instance. The overhead occurs when all entities that are used in the workflow are retrieved, and the child workflow is started in a two-step process that includes a ‘Workflow Expansion Task’ and the actual workflow instance. Therefore, for maximum throughput, use a single long workflow.
  + Mark custom workflow activity as completed.
    - The return value from the Execute method is used by the workflow runtime to mark the activity as “completed.” You should use return base.Execute(executionContext) unless the activity bypasses base class functionality.
    - Avoid returning ActivityExecutionStatus.Closed.
  + Use exception handling in custom workflow activities.
    - You should throw an InvalidPlugInExecutionException in your code. This error will be shown in the workflow instance form.
  + Do not interact with DOM elements using Javascript
    - This very much allowed to use DOM (Document Object Model) elements in JavaScript and as Microsoft Dynamics CRM is a web application so these techniques work, but they are likely to break during an update because the names of the elements they reference are subject to change at any time.
    - So, every time we have to revisit our scripts to check if a specific DOM code that we have written is compatible with latest version or not which is really a boring re-work job to clean these code by alternative codes.
    - Do not use JQuery in Form customization as this is not recommended. Only use JQuery in HTML web resources.
  + Never ever change the files in the Dynamics CRM application folder
    - As Microsoft Dynamics CRM(On-Premise) is a web application so it is hosted in IIS and the files,folders are stored in selected drive on installation.
    - Do not change the default web pages or any files from this folder as this will cause unexpected errors in CRM.
  + Do not manipulate Data directly in CRM Org SQL DB
    - As Dynamics CRM uses SQL Server as its Database, so it is obvious that we can use asp.net application to retrieve data from CRM database without calling CRM SDK.
    - But this is strictly denied as direct database query by passes the security infrastructure of CRM which is the heart of CRM.
    - The recommended practice is to use special filtered views to retrieve the data. This will apply the calling user’s security so that they can only see data that they should see.
    - You can perform updates on the CRM data directly in the database tables. But the risk with this approach is that you can set invalid data that can break the application. Developers should always use the APIs (CRM SDK) provided with the application platform web services to update data.
  + Avoid Record Reference in workflow design.
    - While creating workflows try to avoid using any reference to any record of the system as the record depends on primary key and when the solution will be deployed in other environment the record ID will be changed which will break the reference in the workflow.
    - If there is a requirement to do this then always import the referenced record with Record ID in target environment first so that the reference will not break.
  + Always choose correct filter attributes in Plugin Registration for update messages
    - While we create a plugin to run on update message of an entity then by default the plugin runs on update of any field of the entity record which may cause some unexpected result if it is not handled properly in plugin code.
    - So, to avoid unwanted issues always select correct filter attributes in Plugin registration tool by selecting the sdk step.
  + Use Indexes when required.
    - When we use fetch XMLs or query expressions then ultimately it hits SQL Database as a SQL Query and if the CRM table you are querying involves large data or there is relationship with other tables then you will find performance impact in big projects.
    - So, Add required index on the table to optimize the CRM performance also minimizing database deadlocks.
  + Look up field naming convention.
    - When you add new look-up type field on an entity add a Suffix ID in the field name so that these fields are well identifiable.
  + Caution with Dataverse Plugin registered for Retrieve & Retrieve Multiple Messages.
    - Adding synchronous plug-in logic to the Retrieve and RetrieveMultiple message events can result:  
      – Unresponsive model-driven apps  
      – Slow client interactions  
      – The browser stops responding.
  + Remove deactivated or disabled customizations.
    - Deactivated or disabled customizations should be removed from a solution to improve solution management and to decrease the risk of utilizing or managing an outdated component.
  + Interact with HTTP and HTTPS resources asynchronously.
    - Synchronous requests block the execution of other scripts, which can cause the following:  
      - Unresponsive model-driven and canvas apps  
      - Slow client interactions.

**Summary**

* Follow the naming conventions for entities, attributes, relationships, forms, views, web resources, plugins, workflows, and other components.
* Use solution layers to manage dependencies and avoid conflicts between different customizations.
* Use managed solutions for deployment and unmanaged solutions for development and testing.
* Avoid hard-coding values, such as URLs, record IDs, or option set values, in your code. Use configuration entities or web resource parameters instead.
* Use early-bound classes for better performance and readability of your code. Use late-bound classes only when you need dynamic entity and attribute names.
* Use query expressions or LINQ queries instead of FetchXML queries when possible, as they are easier to write and maintain.
* Use the ExecuteMultipleRequest class to perform bulk operations, such as create, update, or delete, on multiple records. This reduces the number of service calls and improves performance.
* Use the Change Tracking feature to synchronize data changes between Dataverse and external systems. This avoids retrieving all the records and comparing them for changes.
* Use optimistic concurrency to handle concurrent updates to the same record by different users. This prevents data loss and ensures data integrity.
* Use tracing to debug and troubleshoot your code. Use the Trace Control tool to enable and disable tracing and set the trace level.
* Use the Plug-in Profiler tool to profile and debug your plug-ins. You can simulate the execution context of your plug-in and analyze the performance and exceptions.
* Use the Code Analysis tool to check your code for common issues and best practices. You can run the tool from Visual Studio or as a command-line tool.
* Use the Solution Checker tool to validate your solutions for quality and performance issues. You can run the tool from the Power Platform admin center or as a Power Apps CLI command.
* Use the Performance Center to measure the performance of your forms and views. You can enable the Performance Center from the browser console or by using a keyboard shortcut.
* Test your code for cross-browser and cross-device compatibility. Use the BrowserStack tool to test your code on different browsers, devices, and resolutions.

Additional References:

* Pls refer C# coding conventions link for more information –
  + <https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions>
* Pls refer Javascript coding conventions link for more information –
  + <https://www.w3.org/wiki/JavaScript_best_practices>
  + <https://www.w3schools.com/js/js_best_practices.asp>
  + [JavaScript Coding Best Practices for Microsoft Dynamics 365](https://community.dynamics.com/blogs/post/?postid=883bb4ad-5d6b-4b62-84d9-10cad07b1d4a)
* Pls refer Plugin/Workflow coding conventions link for more information –
  + [Microsoft Dynamics CRM Plugin Best Practices](https://community.dynamics.com/blogs/post/?postid=c75ac8b3-516e-49ca-ad82-6b9102f3fd06)
* Pls refer Azure Function coding conventions link for more information – <https://learn.microsoft.com/en-us/azure/azure-functions/functions-best-practices>
* General Development best practices –
  + [Dynamics 365 CRM Development Standards and Best Practices](https://community.dynamics.com/blogs/post/?postid=87ccd57d-b1c4-4695-972e-2ddfe40dbd99)
  + [Best practices (Developer Guide for Dynamics 365 Customer Engagement (on-premises)) | Microsoft Learn](https://learn.microsoft.com/en-us/dynamics365/customerengagement/on-premises/developer/best-practices-sdk?view=op-9-1)
* Best Practices for ALM –
  + [Best practices for ALM in Dynamics 365 applications - Dynamics 365 | Microsoft Learn](https://learn.microsoft.com/en-us/dynamics365/guidance/implementation-guide/application-lifecycle-management-product)
* Best Practices for reports –
  + [Report best practices for Dynamics 365 Customer Engagement (on-premises) | Microsoft Learn](https://learn.microsoft.com/en-us/dynamics365/customerengagement/on-premises/analytics/best-practices-reports?view=op-9-1)
  + <https://learn.microsoft.com/en-us/power-bi/create-reports/service-dashboards-design-tips>
* Dataverse best practices –
  + [Best practices and guidance when coding for Microsoft Dataverse - Power Apps | Microsoft Learn](https://learn.microsoft.com/en-us/power-apps/developer/data-platform/best-practices/)
* Canvas App development best practices –
  + <https://powerapps.microsoft.com/en-us/blog/powerapps-canvas-app-coding-standards-and-guidelines/>