**Quality Code Recommendations for Salesforce developments**

* The purpose of this document it to provide some recommendations that helps to optimize the validation process to deploy code on the different organizations avoiding comments on code review process with know issues that happens regularly.
* The objective of these recommendations is to improve the quality of the code and reduce the time on code review phase and avoid re-working to reach the AL code standards
* Recommendations- General recommendations for coding
  + Be careful with the format of your code, remove blank line not used, indentation, etc.
  + Avoid modifying code in developer console directly, the code modify in this tool sometimes breaks the format of the code.
  + You can use the following extension in Visual Studio code to format your code, specially for new classes.
  + Remove system.debug or console.log
  + Substitute literals values by constants
  + For new classes or big changes, you should analyze the quality of the code with some of the following tools: CodeSense, Apex PMD or Salesforce Code Analyzer, these tools give you an idea of quality of the code, they identify unnecessary complexity, variables not used u other common issues. The recommendation is to use CodeSense using the Cognizant extension.
  + Ensure queries executed contains at least on filter condition to optimize the query.
  + Be careful if the change requested is for CORE or Specific country, Api names of new components need to be set up accordingly.
* This project is totally modularized, so be aware, before using method on loops, review the code of the methods, something these methods could perform some DML operation that hit the Salesforce limit.
  + Avoid nested loops
  + Avoid the use of unnecessary queries. E.g. use getAll() methods to get information from custom metadata Types
  + Use label to manage translations. Also, in case of dynamic message instead of concatenating string values, try to use the method String.format combine with labels to be translate to different language.
  + Use DAY\_ONLY(convertTimezone()) instead of DateTime.newInstance(startDate, Time.newInstance(0, 0, 0, 0)).format('yyyy-MM-dd\'T\'hh:mm:ss\'.000+0000\‘’)

· Don’t call a controller method in Service Layer.

· Use the object name as iterator in a loop, unless there is a reserve word like case

* Don’t use static values on picklist options, define a constant and use this one.
* Use more the helper classes to move come code that could be reused.
* Add sharing on the class
* Query field should be ordered
* Use String.isNotBlank to check null or empty strings.
* In queries add on filter in each line, avoid to split the same filters in several lines or add several filters in the same line.
* Use And instead of OR if possible, specially in batch process or in where could have a mass process.
* When new class are created be aware of key codes, in the following link you can see the current code name convention. https://confluencehhc.airliquide.com/display/OCCEP/Capability+Key+Codes, In case of any doubt please connect with your tech lead to add the proper key and not break the Chinese wall between Core and country specific code.
* Don't use trigger context variables outside of trigger handler. Eg: if (trigger.isInsert |) in and XX\_SM\_yyyyy.cls or XX\_EM\_yyyyy.cls
* Add descritive names for variables, list, maps, etc
* Follwo the name convention in case of doubt pleae take a look in the following link 03.06.04 Naming Convention for Elements
* Dont create ducplicate code.
* Use constant instead of hardcoding string where is possible
* Pase the code sense or any other static tool into your code to prevent the creation of variable that are not use, complex method with cognitive complexity more than 15.
* Dont use icon on labels.
* Ensure that your test class cover all business scenarios
* Review the code after conflicts resolutions. It could be additional changes in the code, as for removed some lines that was added before.
* Don't use custom labels as variable to identify functionalities that could be performed based on the persona. For this you must use custom permission combined with a permission set. Be aware that label could be override in any moment or translate and this will create unnecessary issues.
* Recommendations- Other metadata components
* Remove permission on profiles and add permission on Permission Sets
* Attach flow screenshot for new flows or flow modifications.
* Fields should contain descriptions and indicate the US where this field was requested.
* Be careful with names, this should be aligned between the name of the component and the name on the XML package, also considering case sensitive.
* Validation rules contain bypass (NOT($Permission.CXFCO\_BypassCustomValidations) )
* Recommendations- Test Classes
* Ensure that changes done in an existing class or new class are covered in a test class.
* Use testData Factory classes that already created as testDataUtil,Test\_PatientHelper,test\_ProductHelper, …
* In the data creation try to bypass some triggers, if possible, to reduce the number of operations and create unnecessary data for the test.
* In test classes ensure test/stop method are present.
* Do not use system.assert method, use assert method directly.
* assert should be out of start/stop Methods
* In test classes should be considered positive and negative scenarios if possible.
* Need to be ensure that not only the code should be covered, also it should be added the proper assert to consider the maximum of possible scenarios.
* Recommendations- LWC
* In order to reduce the numbers of queries in Apex Classes. For simple queries or operation, please replace the use of apex class by GrahpQL.
* With GraphQL you can get records or related records also in the case of perform some unitary DML for a single record, we could used createRecord or UpdateRecord, so you don’t need to use an apex class for these operation avoid additional code.
* In the .js of you LWC you can add the following imports.
* import { getRecord, getFieldValue,updateRecord ,createRecord} from "lightning/uiRecordApi";
* import { gql, graphql } from "lightning/uiGraphQLApi";
* In the following files you could find same examples already implemented in AL.
* fsappPassQuestionnaire.js
* fsappSyncData.js
* fsappPerformTask.js
* Please check the following links for more information. Unleashing the Power of GraphQL and Lightning Web Components (LWC) in App Development (linkedin.com)
* In the following link you see more information about the use of GraphQL in Salesforce: Limitations and best practices.
* https://developer.salesforce.com/docs/platform/graphql/guide/query-limits.html#semi-join-and-anti-join-queries https://developer.salesforce.com/docs/platform/graphql/guide/graphql-wire-lwc.html o https://developer.salesforce.com/docs/platform/graphql/guide/graphql-wire-lwc-best.html https://developer.salesforce.com/blogs/2023/06/explore-the-graphql-wire-adapter-now-in-beta
* https://developer.salesforce.com/docs/platform/graphql/guide/graphql-wire-lwc-limitations.html. (Take a look specially into this link)
* Apex cannot be used as controller class for LWC that are used in Salesforce mobile application.
* Add in the controller, Cacheable=true when there is not any DML and the date recover is no get from any button(@AuraEnabled(cacheable = true))
* Use the teckoConstants.js for global constants
* Use the class TEKCO\_GenericController to perform DML operations from LWC.
* All LWC must to contain the proper label to translate to different languages,
* Recommendations- Specific classes
* In case of queries or DML operations that need to be executed in a without sharing context, it should be used the following classes.
* WithoutSharingDmlHandler, this class contains Insert, update and delete operation.
* WithoutSharingSOQLHandler, this should contain only sql queries.
* Bear in mind the class UserHelper, this class provide information about the user who is executed the transaction, here you can use several methods and don’t need to create new ones.
* Use the class TEKCO\_GenericBatchObject to create batch process, this class contains the structure for a batch process.
* Recommendations – Improve Pipeline execution time
* DevOps Recommendations
* Every story should have its own feature branch, with all commits mentioning the issue key (US / Improvement / Tasks / Bugs for US that is already in production, or for regression testing)
* Should be in a parent issue's feature branch, with commits mentioning the parent AND child issue key, and merge requests on promotions should contain the child issue key in description (Bugs for US that are not yet Ready for production).
* There is a rule already in place to follow this stringently without skipping it while doing commit.
* Avoid committing other story / bug in single story feature branches as a common that will result in blocking while prod deployment if one is not Ready for production status.
* Avoid resolving conflicts in feature branches that will lead in creating more conflicts / bugs during prod deployment.
* Don’t do changes directly in QA/UAT, Reach out to DevOps / Tech leads.
* Avoid posting client credentials (consumer keys, secrets, API keys, passwords) on the PDS confluence page. Instead, provide instructions on how to obtain the credentials or contact the person responsible for generating them directly.
* When a bug is fixed on the user story feature branch. It is mandatory that the issue key it is mention on commit messages and on the MR Request description, This little action enable th3e Jira - Gitlab integration to reflect the information properly.
* For the improvements linked to a main user story, don't create new branches, changes should be added on the feature branch of the main User Story.