**PROJECT DEVELOPMENT PHASE**

**CODE-LAYOUT, READABILITY, REUSABILITY**

|  |  |
| --- | --- |
| **Date** | **03 NOVEMBER 2023** |
| **Team ID** | **NM2023TMID04737** |
| **Project name** | **A CRM Application For Managing Loan App and Fraudulent Prevention** |

Code layout, readability, and reusability are crucial aspects of software development for a CRM application like the one you mentioned. They contribute to maintainability, collaboration, and long-term success of the project. Here are some best practices to ensure code layout, readability, and reusability in your CRM application:

1. **Trigger Class:**

trigger LoanTrigger on LoanApplication\_\_c (before insert,After insert){

    if(trigger.isBefore)

    {

        if(trigger.isInsert)

        {

            LoanApplicantException.Application(trigger.new);

        }

    }

    else if(trigger.isInsert && trigger.isAfter)

    {

        LoanApplicantException.pending(trigger.new);

    }

1. **Handler Class:**

public class Transactionlo {

    public static void trans(list<loantransaction\_\_c> ls)

    {

        string s='';

        decimal i=0;

        for(loantransaction\_\_c l:ls)

        {

             s=l.pan\_number\_\_c;

             i=l.Amount\_\_c;

             break;

        }

        pendingloan\_\_c p= [ select pan\_number\_\_c, netamount\_\_c from pendingloan\_\_c where pan\_number\_\_c=:s];

        p.NetAmount\_\_c=p.NetAmount\_\_c-i;

        update p;

        if(p.NetAmount\_\_c==0)

        {

            loanapplication\_\_c la=[select dayspassed\_\_c,netamount\_\_c from loanapplication\_\_c where pan\_number\_\_c=:s];

            loanclearence\_\_c lc= new loanclearence\_\_c();

            lc.TimeTaken\_\_c=la.DaysPassed\_\_c;

            lc.PAN\_NUMBER\_\_c=s;

            lc.AmountCleared\_\_c=la.NetAmount\_\_c;

            insert lc;

            RecordDelete.rec(s);

        }

    }

}

1. **Apex Scheduler:**

public class DayUpdateSchedulable implements Schedulable{

    public static void execute(SchedulableContext sc)

    {

        list<pendingloan\_\_c> p= [select pan\_number\_\_c,netamount\_\_c,interest\_\_c from pendingloan\_\_c];

        list<loanapplication\_\_c> l=[select pan\_number\_\_c,dayspassed\_\_c from loanapplication\_\_c];

        for(pendingloan\_\_c po:p)

        {

            for(loanapplication\_\_c lp:l)

            {

                if((po.PAN\_NUMBER\_\_c==lp.PAN\_Number\_\_c)&&(math.mod(integer.valueOf(lp.dayspassed\_\_c), 30)==0))

                   {

                       po.NetAmount\_\_c=po.NetAmount\_\_c+(po.NetAmount\_\_c\*po.Interest\_\_c)/100;

                       po.InterestLimit\_\_c=po.InterestLimit\_\_c+1;

                    }

                else if((po.PAN\_NUMBER\_\_c==lp.PAN\_Number\_\_c)&&(math.mod(integer.valueOf(lp.dayspassed\_\_c), 30)==0)&&(math.mod(integer.valueOf(po.InterestLimit\_\_c),5)==0))

                {

                    po.Interest\_\_c=po.Interest\_\_c+(po.Interest\_\_c/2);

                    po.NetAmount\_\_c=po.NetAmount\_\_c+(po.NetAmount\_\_c\*po.Interest\_\_c)/100;

                    po.InterestLimit\_\_c=po.InterestLimit\_\_c+1;

                }

            }

        }

        update p;

    }

}

1. **Consistent Code Layout:**
   * Maintain a consistent code layout throughout the project. Use a consistent and well-defined coding style, such as PEP 8 for Python or Google's Java Style Guide for Java.
   * Indentation: Use a standard number of spaces or tabs for indentation, and be consistent across the entire codebase.
   * Line Length: Limit line lengths to improve readability. A common guideline is to keep lines under 80-100 characters.
2. **Meaningful Variable and Function Names:**
   * Use descriptive and meaningful names for variables, functions, and classes. Avoid cryptic abbreviations or acronyms.
   * Follow a naming convention consistently (e.g., camel case or snake case) for variables and functions.
3. **Comments and Documentation:**
   * Include comments to explain complex logic, algorithms, and any non-trivial code sections.
   * Write clear and concise documentation for classes, functions, and APIs. Use tools like Doxygen or Javadoc to auto-generate documentation.
4. **Modular Code Structure:**
   * Organize your code into modules or packages with clear responsibilities. Separate concerns to improve maintainability.
   * Implement the Single Responsibility Principle, where each class or function should have a single, well-defined purpose.
5. **Use of Design Patterns:**
   * Apply design patterns where appropriate to solve recurring design problems in a standardized and reusable way. For example, the Factory, Singleton, or Strategy pattern may be useful in a CRM application.
6. **Avoid Code Duplication:**
   * Refactor and eliminate code duplication to ensure that similar functionality is implemented in a single place.
   * Create utility functions or libraries for commonly used code.
7. **Unit Testing:**
   * Write unit tests for critical components of your CRM application. Testing ensures code correctness and helps prevent regressions.
   * Use a testing framework and follow Test-Driven Development (TDD) principles where feasible.
8. **Error Handling:**
   * Implement robust error handling to provide clear error messages and gracefully handle exceptions.
   * Create custom exceptions or error classes for better error identification.
9. **Version Control:**
   * Use a version control system (e.g., Git) to manage changes and collaborate with other developers.
   * Maintain clear commit messages and follow a branching strategy for feature development and bug fixes.
10. **Code Reviews:**
    * Conduct regular code reviews with team members to ensure code quality and consistency.
    * Encourage constructive feedback and suggestions for improvement.
11. **Dependency Management:**
    * Use package managers (e.g., npm, pip, Maven) to manage external dependencies and libraries.
    * Keep dependencies up-to-date to benefit from bug fixes and new features.
12. **Code Reusability:**
    * Encapsulate reusable components and functions as libraries or modules that can be used across the application or in future projects.
    * Follow the DRY (Don't Repeat Yourself) principle to reduce redundancy.
13. **Coding Standards and Guidelines:**
    * Define and enforce coding standards and guidelines for your development team.
    * Create a coding style guide or document to ensure consistency.
14. **Code Comments and TODOs:**
    * Add comments with "TODO" markers to highlight areas that need attention, improvements, or future work.
15. **Continuous Integration (CI):**
    * Implement CI/CD pipelines to automate build, test, and deployment processes, ensuring code quality and consistency.
16. **Performance Optimization:**
    * Profile and optimize critical sections of the code to ensure the application runs efficiently.