****

**Guidelines**

**Project Name**: Find the Interest Amount for current year (emicalculator.net)

**Team Members:**

1. Vyshnavi Garikapati - 899386
2. Nimmanapalli Sai Abhijith Reddy - 899433
3. Navitha Jutur Chinna Reddy - 899490
4. Anusha Vontela - 899447
5. Balla Divya Bhargavi - 899424

**Change Log History:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Document Version** | **Reviewer** | **Description** |
| 19-02-2021 | V1 | Venkatesh Indra mohan(Technical trainer) | Initial Document |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

****

**Contents:**

1. Scope
2. Testing Approach
3. Testing Techniques
4. Testing Environment
5. Testing Tools
6. Templates
7. Functional Requirement Document
8. Test Scenarios
9. Test Cases
10. Test Suites
11. Coding Standards
12. Testing Standards
13. Team Communication

****

**EMI CALCULATOR**

**1. Scope –**

This Document is going to give the complete details regarding the project. This includes Testing Approaches, Testing Techniques, Coding Standards, Testing Standards and Communication channels etc.

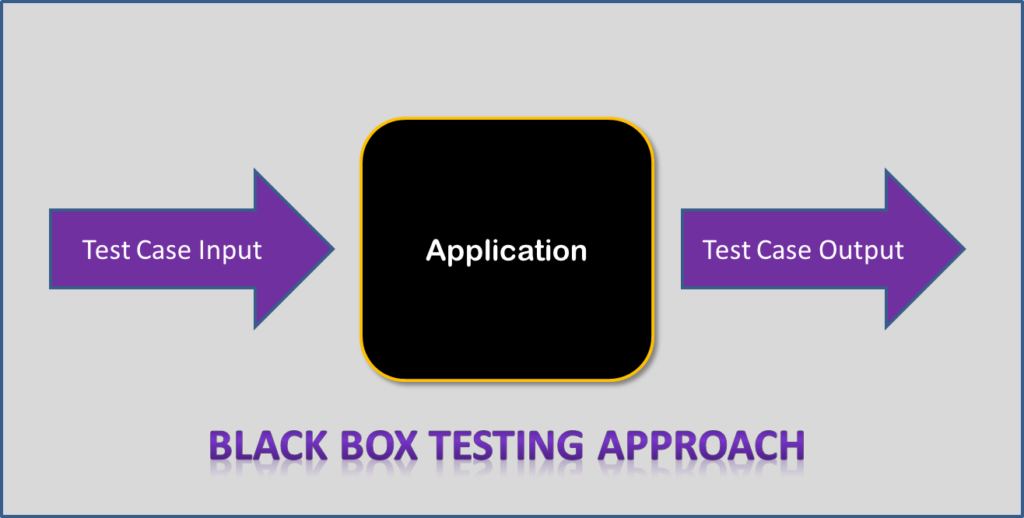
This document will be reviewed by –

* Trainer
* Academy Coach
* Business Unit (BU) Person
* Project Team

**2. Testing Approach –**

A). Black Box Testing - The project is based on the Black Box Testing approach for functional testing. Test cases will use the input values and will give the output without knowing the internal structure of fields. Below added picture best describes the Black Box testing approach. Test cases will be given and output will be asserted.

* Requirements are obtained (To Check Functionality and UI)
* Test data will be prepared
* Test Cases will be prepared
* Test Cases will be given as an input
* Output of the test case will be fetched and asserted
* Output will be stored

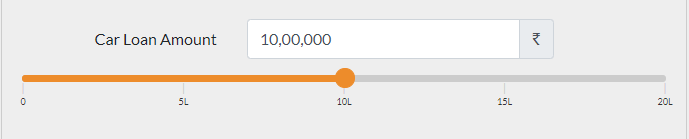


****

B). Manual Testing –The Project is based on “Manual Testing” for GUI testing. The knowledge of the testing team based on manual testing is expected to use for GUI Testing. Based on the Requirements GUI is expected to be tested. As per the “Manual Testing” standard team will be looking into the requirements and the web application and will be tested based on best knowledge to tester’s team. Manual Testing will include the “Texts”, “Labels”, “Input Box”, “Scales” and “Tables”.

**3. Testing Techniques-**

* Functional Testing –
  + Smoke Testing – Smoke Testing will be covered using 2% to 5% amount of total test cases.
  + Regression Testing – Regression Testing will be covered approx. 20% amount of the total test cases.
* Equivalence Partitioning –
  + Equivalence Partitioning will cover the different classes. In this technique input values will be divided into different classes. For example –



Based on above image there will be 3 classes for the input values as mentioned bellow –

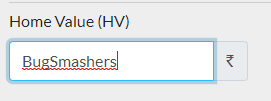
1. Input Values can be Less than the minimum value -> (-10L< 0)

2. Input Values can be Greater than the maximum value -> (25L > 20L)

3. Input Values can be in between minimum and maximum -> (0<10L<20L)

* Fault Insertion –
  + ****Fault Insertion will cover the different fault insertion tests in the fields. In this technique input values will be fault values for the fields –

**Team BugSmashers**



Based on above image, there will be different aspects which are expected to be covered in fault insertion. Below mentioned input values are expected to be covered in fault insertion.

1. Input Value – “BugSmashers”

2.Input Value – “Bug100”

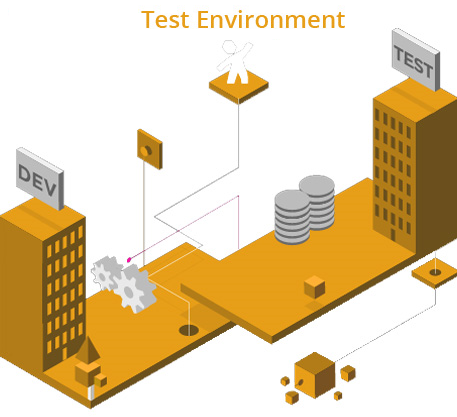
3.Input value – “@#$%”

4. Input value – “#$%FF56”

**4. Testing Environment –**

Testing environment will contain –

* + System and applications,Test data,Front-end running environment,Browsers, Hardware, Networks.



It is one of the most important tools of the tester, which plays a key role in achieving a success in the testing. It provides all the pre-requisites that are required to perform the task of testing on a particular software product. These pre-requisites may contain operating system (on which software product intends to work), servers, users PCs, memory, space, any specific software or application, test drivers, stubs, etc.

****

**15. Testing Tools –**

To execute the project below tools will be required –

* Eclipse – IDE will be used for running and holding the complete project. It will work as an engine for the testing environment. Eclipse Neon 3 is expected to be used.
* Selenium – Open-Source library to automate the browser for performing automation testing. Selenium 3.141.59 version is expected to use.
* Java – Object Oriented Programming language which will be used to write the automation script. java version "1.8.0\_281" is expected to be used.
* Test – NG – Testing framework which will be used to manage the test cases and the test scenarios. TestNG – 6.10
* Maven – Build automation tool which will be used to manage the project flow. Maven Version - 1.7.0.20160603-1933.
* Apache POI – Apache POI library is expected to use for the data driven framework for test data management. Apache POI – Version –4.1.2.

**6. Templates –**

A well-written test case should:

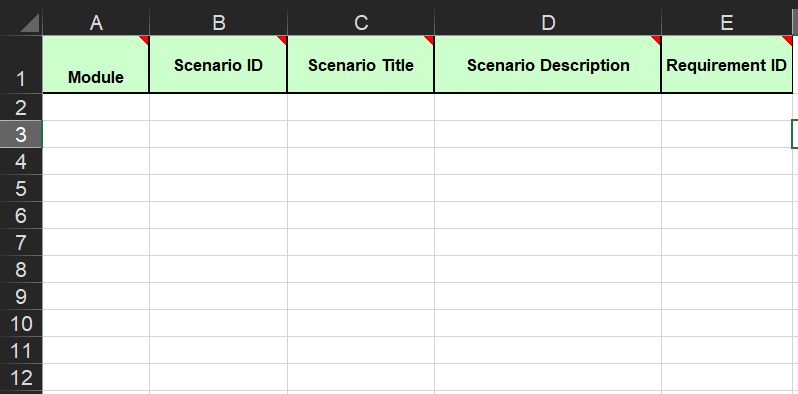
* Easy to understand and execute
* Create Test Cases with End User’s perspective
* Use unique test case id
* Have a clear description
* Add proper pre & postconditions
* Specify the exact expected result
* Test cases should be reusable & maintainable
* Utilize testing techniques

If you follow the best practices to write test cases then anyone in the team can understand and execute the well-written test case easily. It should be easy to read and understand, not only for whoever wrote it but also for other testers as well.

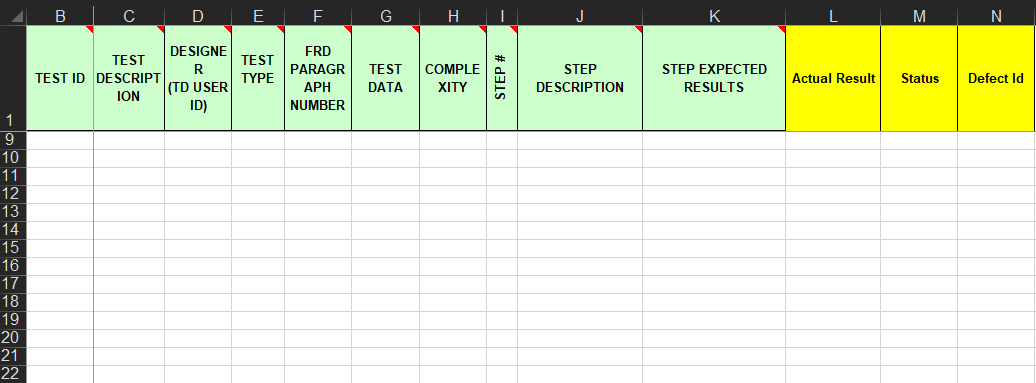
For manual test case preparation, the below templates are expected to be used –

****

1. Test Scenarios –

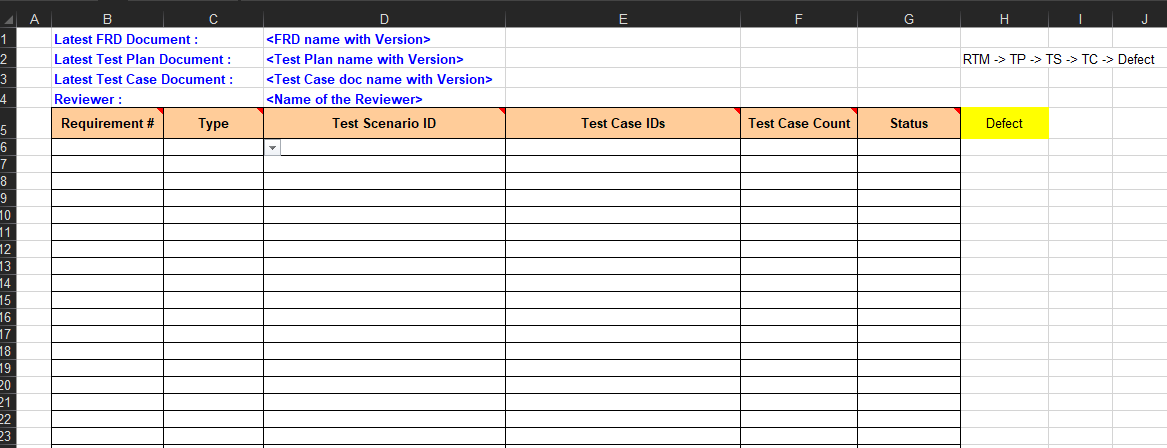


****2. Test Cases –

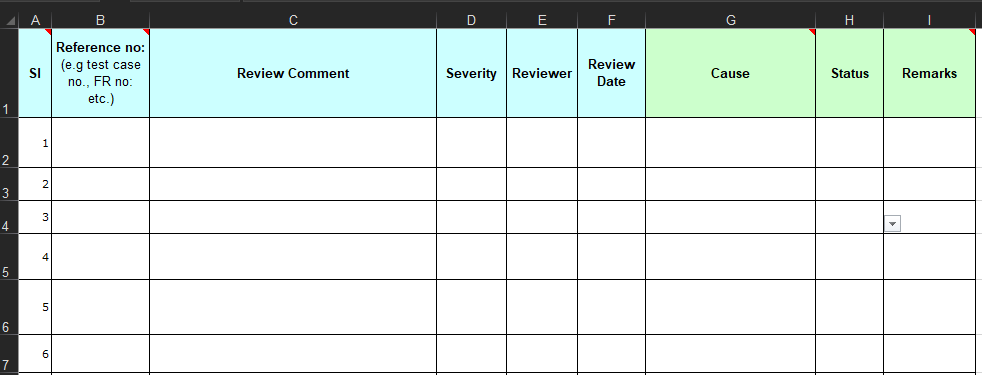


****

3. RTM –

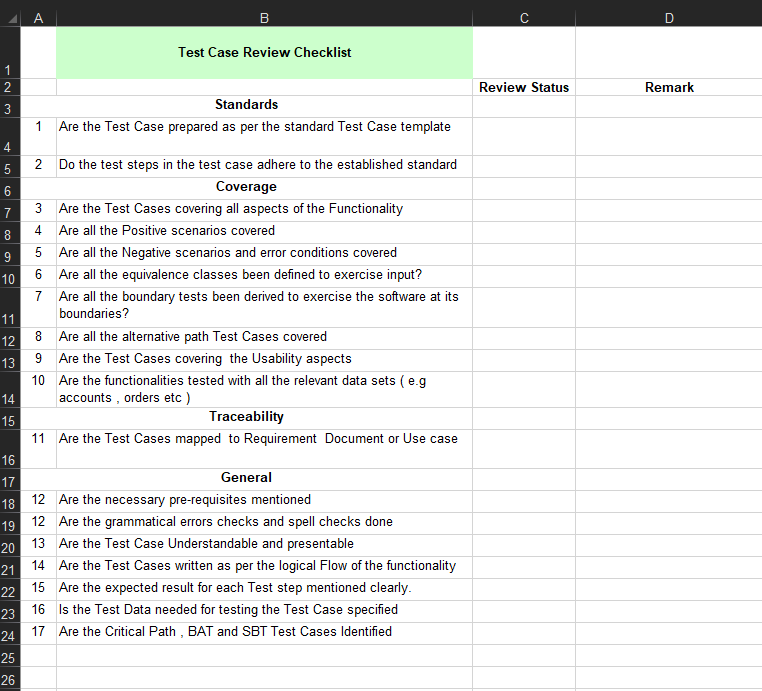


4. Review Comments –



****

5. Review Check List –



**7. Functional Requirement Document –**

Document Contains complete requirement for the project functionalities of web application like – “Car Loan”, “Home Loan EMI Calculator”, “Loan calculator”, “EMI Calculator”, “Loan Amount Calculator”, “Loan Tenure Calculator”, “Interest Rate Calculator”. Flow of the functions is expected to be covered in the FRD. All the required flow for project is expected to be taken from FRD. Behavior of fields and values constraints for application is expected to be present is the FRD.

**8. Test Scenarios –**

In Test Scenarios we are expected to cover all the possible testing scenarios for the project – including

* Home Page Validation
* Car Loan EMI Calculator Validation
* Home Loan EMI Calculator Validation
* EMI Calculator
* Loan Amount Calculator
* Loan Tenure Calculator

**9. Test Cases –**

In Test Cases we are expected to cover all the possible test cases to insure the coverage of the testing. Test cases will have valid data input validation and invalid data input validation. Test cases will be covered using negative values, positive values with numeric input and will cover alphanumeric, alphabetic, and special character value input validation and null input validation as well.

Test Data is expected to be used in below formats -

* 100000
* 1020WER
* 230@#$
* Bugsmasher
* @#$\*&
* 1.23.
* “ ”

**10. Test Suites –**

Test Suites are expected to be framed to cover the “Regression”, “Smoke” and “Default” testing coverage. Test Cases and Scenarios will be grouped according the “Suites” and based on testing is expected to be done. Regression suite will cover the “Test Cases” which are expected to be performed multiple times. Smoke suite will cover the “Build Verification” suite to ensure the build is working. Default suite will be covering all the necessary test cases including the “Regression” and “Smoke” suites test cases.

**11. Coding Standards:**

* 1. Naming Conventions:

Some of the naming conventions for User defined Variables,Parameters, Packages, Classes, Methods, Projects are given below:

* ****Meaningful and understandable name help anyone to understand the reason of using it.

Example:“phoneNumber” is a variable name for phone number

* The names should be written in camel case starting with small letters.

Example: loanTenureCalculator

* 1. Indentation:

Proper indentation is very important to increase the readability of the code. For making the code readable, use White spaces properly. Some of the spacing conventions are given below:

* There must be a space after giving a comma and semicolon between two function arguments.

Example: getDetails(age, degree)

for(i=0; i<=10; i++)

* Each nested block should be properly indented and spaced.
* All braces should start from a new line and the code following the end of braces also start from a new line.

Example:

public Indentation()

{

while (n > 0)//while loop

{

System.out.println();

n++;

}

if (Cond1 == val1) // if- else if - else conditional statements

{

System.out.println("Cond1 is val1");

} else if (Cond1 == val2)

{

System.out.println("Cond2 is val2");

}

else

{

System.out.println("No condition is satisfied");

}

switch (Val1) // Switch construct

{

case 1:

System.out.println("This is Java");

break;

case 2:

System.out.println("This is C++");

break;

default:

System.out.println("This can be anything");

break;

}

}

****

* 1. Compiler Warnings:

Compiler and linker warnings shall be treated as errors and fixed. Even though the program will continue to compile in the presence of warnings, they often indicate problems which may affect the behaviour, reliability, and portability of the code.

* 1. Inline Comments:

Inline comments should be used to make the code clearer to a programmer trying to read and understand it. Writing a well-structured program lends much to its readability even without inline comments. Use inline comments to generalize what a block of code, conditional structure, or control structure is doing. Do not use overuse inline comments to explain program details which are readily obvious to an intermediately skilled programmer. A rule of thumb is that inline comments should make up 20% of the total lines of code in a program, excluding the header documentation blocks.

* 1. Exception handling:

Here are some tips for exception handling that we follow -

1. Never swallow the exception in catch block:

catch (NoSuchMethodException e)

{

   return null;

}

Doing this not only return “null” instead of handling or re-throwing the exception, but it also totally swallows the exception, losing the cause of error forever. And when you do not know the reason of failure, how you would prevent it in future? Never do this!!

2. Declare the specific exceptions that your method can throw:

Declare the specific checked exceptions that your method can throw. If there are just too many such checked exceptions, you should probably wrap them in your own exception and add information to in exception message. You can also consider code refactoring also if possible.

public void foo() throws SpecificException1, SpecificException2

{

//Correct way

}

#### 3. Do not catch the Exception class rather catch specific sub classes:

|  |
| --- |
| try  {     someMethod();  } catch (Exception e)  {     LOGGER.error("method has failed", e);  } |

The problem with catching Exception is that if the method you are calling later adds a new checked exception to its method signature, the developer’s intent is that you should handle the specific new exception. If your code just catches Exception (or Throwable), you’ll never know about the change and the fact that your code is now wrong and might break at any point of time in runtime.

#### 4. Never catch Throwable class:

Well, its one step more serious trouble. Because java errors are also subclasses of the Throwable. Errors are irreversible conditions that can not be handled by JVM itself. And for some JVM implementations, JVM might not actually even invoke your catch clause on an Error.

**12. Testing Standards**

* 1. **Testing must be done with unexpected and negative inputs:**

Testing should be done with correct data and test cases as well as with flawed test cases to make sure the system is leak proof. Test cases must be well documented to ensure future reuse for testing at later stages. This means that the test cases must be enlisted with proper definitions and descriptions of inputs passed and respective outputs expected. Testing should be done for functional as well as the non-functional requirements of the software product.

* 1. **Inspecting test results properly:**

Quantitative assessment of tests and their results must be done. The documentation should be referred to properly while validating the results of the test cases to ensure proper testing. Testing must be supported by automated tools and techniques as much as possible. Besides ensuring that the system does what all it is supposed to do, testers also need to ensure that the system does not perform operations which it is not supposed to do.

* 1. **Validating assumption:**

The test cases should never be developed based on assumptions or hypothesis. They must always be validated properly. For instance, assuming that the software product is free from any bugs while designing test cases may result in extremely weak test cases.

* 1. **General guidelines:**
  + The tests for a function or method should

Test for success:

* Test the function with typical, valid input values
* Test with valid, edge-case inputs
* If the function has multiple parameters, test them in different combinations

Test for failure:

* + Test that the function fails correctly (e.g., raises the expected type of exception) when given likely invalid inputs (for example, if the user passes an invalid user\_id as a parameter)
  + Test that the function fails correctly when given bizarre input
  + Test that the function behaves correctly when given Unicode characters as input
  + Each test module, class and method should be able to be run on its own.
  + Tests should not be tightly coupled to each other, changing a test should not affect other tests.
  + ****It should be quick and easy to see what went wrong when a test fails, or to see what a test does and how it works if you have to debug or update a test.
  + Record or take screenshot for the failed scenarios.
  + From all available selenium locators, mostly use in the CSS Selector or xpath.
  + Mostly avoid hard waits, use instead implicit or explicit waits.

1. **Team Communication:**
   1. Assertive communication:

In cases when there are open bugs or critical issues, it’s really not important *who* caused them but rather *how to fix* them. The point is to get your software product ready, and you have to communicate how to do that. It is very important not to hurt anyone’s feelings during this process. Make sure you practice [assertive communication methods](http://www.mindtools.com/pages/article/Assertiveness.htm) carefully and regularly.

* 1. Chat:

By this I mean chatting electronically, such as through instant messages. A great deal of technical details can be shared on chat easily and in real time. It really helps if the chat is set to save the conversation so you can search for details later from the chat history.

We are expected to use WhatsApp and Microsoft Teams

* 1. Email:

While email may be helpful for reports or technical documentation, they are not effective for status updates or to discuss important schedules or changes.So, for reporting and documenting we are expected to use Outlook and Gmail.

* 1. Shared documents:

If email is where documents go to disappear, then a network share or cloud folder is a place where they are at least discoverable, and to more people than just those on the email. Functions like change history and regular backups can be a huge help dealing with the huge set of documents.

We are expected to use OneDrive

Video conference:

Organizing a video conference is a very easy yet effective method for meetings. Of course, you will have to prepare for them in advance to make sure your message is getting through. Here are some tips:

* Confirm that the microphone, camera, and connection work properly. Technical equipment can always go wrong, and even a service pack can ruin your settings. Put a placeholder for preparation in your calendar before important meetings.
* Make notes for the meeting on a medium you prefer—paper, simple text file, or note client using a cloud service. Having the agenda jotted up beforehand can help you stick to the point and keep the conversation on topic.

We are expected to use Microsoft Teams, Google Meet and Zoom. However we are free to choose the tool as per the needs.

* 1. Meeting minutes:

If there have been problems with people forgetting the outcome of a meeting or arguing about it, considering taking down and posting minutes and action items.

We are expected to post timing in chat or if necessary make a call and remind.

* 1. Follow-up:

Setting up a meeting a few days after major meetings can help motivate the action items being processed. Team members also can check the decisions made previously on these meetings and change them, if necessary.

****