|  |  |  |  |
| --- | --- | --- | --- |
| **System Name** | Insurance > Agency Management | **Request Type** |  |
| **Redmine No** |  | **Priority** | Normal |
| **Helpdesk Case ID** |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Details** | | | | | |
| **Description** | **Insurance > Agency Management**  Unit testing to ensure  1. Agents can easily see how well the business is doing this year. This includes the number of policies submitted and the total premium collected.  2. Agents have the freedom to add and update customer details for specific insurance products before issuing a policy.  3. The Agent portal simplifies things by showing customer details for specific policies and give agents a quick look at their yearly performance. | | | | |
| **Testing Type** | Unit Testing  Integration Testing  Data Integrity Testing | | | | |
| **Impact Module** | The system enables successful saving, adding, deleting, and retrieving of details for a specific customer associated with a particular policy. | | | | |
| **Test Scope** | | | |  | |
| 1. The testing focus is on the functionalities relevant to an insurance agent's role. 2. Ensures that features align with an agent's tasks and responsibilities within the insurance system. | | | | | |
|  | |  |  | |
| **Solution (Bug/ Improvement)** | | | | | |
| **Solution (to User)** | **Improvement**  1. **Efficient Task Completion:**  Agents utilize the portal for streamlined task execution, significantly reducing the time required compared to manual processes.  2.**Comprehensive Performance Assessment:**  - The portal enables agents to complete proposals, access customer details, and evaluate their performances with ease and effectiveness. | | | | |
| **Technical Solution** |  | | | | |
| **Technical Limitation** |  | | | | |
|  | |  |  | |
| **Bug Details** | | | | | |
| **Steps to Reproduce** |  | | | | |
| **Root Cause** |  | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Code Review Checklist** | | **Not Applicable** | **Done** |
|  | Has the specification properly been translated into code? |  |  |
|  | Has proper use of language conventions been made? |  |  |
|  | Are structured programming constructs used throughout? |  |  |
|  | Are all comments unambiguous and correct? |  |  |
|  | Are all data types and data declarations proper? |  |  |
|  | Is the external interface correctly coded? |  |  |
|  | Is the interface consistent with the architectural design? |  |  |
|  | Has error handling been specified? |  |  |
|  | Are all input arguments used? |  |  |
|  | Are all output arguments produced? |  |  |
|  | Are all local variables used only after initialization and of the correct scope? |  |  |
|  | Are the naming convention followed? |  |  |
|  | Is there any hard coding? |  |  |
|  | Does each requirement that applies to the routine have its own test case? |  |  |
|  | Test case for every loop termination. |  |  |
|  | Test case for resource release (Closing DB connection, release object etc.) |  |  |
|  | Test case for out of scope and exceptional error handling. |  |  |
|  | Test case for integration and data verification. |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Checklist for Unit Testing** | | **Not Applicable** | **Done** |
|  | Unit tests case with appropriate data has been developed to exercise and validate all specified application requirements, functions, and objectives. |  |  |
|  | Has a list of common errors been used to write test cases to detect errors? |  |  |
|  | Have all boundaries been tested-maximum, minimum, and off-byone boundaries? |  |  |
|  | Do test cases check for the wrong kind of data, for example, a negative number of employees in a program? |  |  |
|  | Retesting is required to verify that the problem solution eliminates the problem and does not introduce new errors. |  |  |
|  | For each type of test conducted, the test results are compared with the expected results and recorded. |  |  |

|  |  |
| --- | --- |
| **Program Name** Isf | |
| **RIAG\_PROFILE.jsp → iimsweb\_ri\WebContent\WEB-INF\classes\ri**  **RIAG\_ejbPROFILEBEAN.java -→iimsejb\_ri\ejbModule\ri**  **RIAG\_ejbPROFILEBEANHome.java -→iimsejb\_ri\ejbModule\ri**  **RIAG\_ejbPROFILEBEANRemote.java -→iimsejb\_ri\ejbModule\ri**  **RIAG\_cjbPROFILE.java → iimsutil\source\ri**  **RIAG\_cPROFILE.java → iimsweb\_ri\WebContent\WEB-INF\classes\ri**  **RIAG\_PROFILE\_COPY.jsp → iimsweb\_ri\JavaSource\ri** | |
| **Remarks** | |
|  | |
|  | |
| Prepared By: DILIPKUMAR.P SIVARAMAN.V SANTHOSH.ADate : 03/01/2024 | **Reviewed By :**  **Date :** |
| **Please refer to the softcopy of the Test Plan in the URL as below:** | |

|  |  |
| --- | --- |
| Term | Description |
| Coding Convention | **Coding conventions** are a set of guidelines for a specific [programming language](https://en.wikipedia.org/wiki/Programming_language) that recommend [programming style](https://en.wikipedia.org/wiki/Programming_style), practices and methods for each aspect of a piece program written in this language. These conventions usually cover file organization, [indentation](https://en.wikipedia.org/wiki/Indent_style), [comments](https://en.wikipedia.org/wiki/Comment_(computer_programming)), [declarations](https://en.wikipedia.org/wiki/Declaration_(computer_science)), [statements](https://en.wikipedia.org/wiki/Statement_(programming)), [white space](https://en.wikipedia.org/wiki/Whitespace_(computer_science)), [naming conventions](https://en.wikipedia.org/wiki/Identifier_naming_convention),[programming practices](https://en.wikipedia.org/wiki/Best_Coding_Practices), [programming principles](https://en.wikipedia.org/wiki/Category:Programming_principles), [programming rules of thumb](https://en.wikipedia.org/wiki/Category:Programming_rules_of_thumb), architectural best practices, etc.  **Example**  1) Separation of code and data  2) Consistent Temporary Name  3) File and Folder Organization  4) Limit line Length  5) Avoid Deep Nesting  6) Capitalize SQL Special Words  7) Consistent Naming Scheme (camelcase or underscore)  8) Code Grouping |
| Structured programming | **Structured programming** is a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) aimed at improving the clarity, quality, and development time of a [computer program](https://en.wikipedia.org/wiki/Computer_program) by making extensive use of [subroutines](https://en.wikipedia.org/wiki/Subroutines), [block structures](https://en.wikipedia.org/wiki/Block_(programming)), [for](https://en.wikipedia.org/wiki/For_loop) and [while loops](https://en.wikipedia.org/wiki/While_loop)—in contrast to using simple tests and jumps such as the *[goto](https://en.wikipedia.org/wiki/Goto)* statement which could lead to "[spaghetti code](https://en.wikipedia.org/wiki/Spaghetti_code)" which is difficult both to follow and to maintain. |
| Comments unambiguous | Code need to be described and commented to for easier maintenance. Comments can be added at the function definition which can be previewed whenever it is used. But avoid obvious comments. |
| Data type declaration | The correct data type should be used and declared. E.g. a double/single should be used for normal computation instead of long etc. |
| Multiple entry points | Extra care need to be taken for function/code with multiple entry point as integration testing will need to be carried out for each of the entry points. |

**Glossary (Not to be printed**