**Sample Functional Requirements:**

1. The application should receive input of 2 digits

2. It should perform arithmetic operations like +-\*/

3. It should handle edge cases like divide by 0

4. It should have clear button

5. It should display output in text box

**Sample Non Functional Requirements:**

1. The calculator should provide near-instantaneous responses to user input, ensuring a smooth and responsive user experience within 2 sec.
2. It should be able to handle a minimum of 100 simultaneous user sessions without a significant decrease in performance.
3. The calculator should be compatible with popular web browsers (e.g., Chrome, Firefox, Safari) and operating systems (e.g., Windows, macOS, Linux).
4. The colour of digits and operator button should be black and display area is white
5. It should close after session timeout.

**critical business test scenarios for basic calculator application**

1. Verify that the calculator can perform basic arithmetic operations (addition, subtraction, multiplication, and division) accurately.
2. Test with both positive and negative numbers.
3. Ensure the calculator handles invalid inputs gracefully (e.g., alphabetic characters, special characters).
4. Verify that attempting to divide by zero results in an appropriate error message.

**Non critical business test scenarios for basic calculator application**

1. Test the calculator's ability to store and display a history of calculations.
2. Test the application's readability in both light and dark modes.
3. Test for auto-correct and auto-complete functionalities in case users make common typing errors.
4. **Business-Critical Test Scenarios:**
   * **Essential Functions:** These scenarios focus on critical functionalities that are vital for the core business processes. Failure in these areas can have a significant impact on the overall business operation.
   * **High-Impact Features:** Test scenarios related to features that, if not functioning properly, could result in financial loss, legal issues, or damage to the organization's reputation.
   * **Security and Compliance:** Testing scenarios related to data security, privacy, and compliance with industry regulations are crucial, especially if the application deals with sensitive information.
5. **Non-Business-Critical Test Scenarios:**
   * **Secondary Functions:** Scenarios covering features that, while still important, may not directly impact the core business processes. Failure in these areas might be inconvenient but not catastrophic.
   * **User Experience:** Testing scenarios related to the user interface, usability, and overall user experience, ensuring a pleasant interaction but not directly affecting critical business operations.
   * **Performance under Load:** While performance is important, scenarios related to extreme load or stress conditions might fall into this category if the impact on business operations is not immediate.

Q2.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Scenario ID** | **Test Scenario** | **Testcase ID** | **Test Case** | **Severity: C – Critical, M – Major, S – Small** | **Test Steps** | **Test data** | **Expected O/p** | **Actual O/p** | **Status** | **Comments** |
| TS1.1 | Validate Gmail Login | TC1.1 | Validate with correct username and pwd | C | 1) Open gmail.com 2) Enter username and pwd 3) Click Login button | Username: Arun Pwd: \*\*arun\*\* | Lands on Inbox page | Lands on InboxPage | Passed |  |
| TC1.2 | Validate with Incorrect username and correct pwd | C | 1) Open gmail.com 2) Enter username and pwd 3) Click Login button | Username: Aruna Pwd: \*\*arun\*\* | Display error msg "Invalid userid or pwd" | Lands on InboxPage | Failed | allow to home page after wrong credentials |
| TC1.3 | Validate with correct username and Incorrect pwd | C | 1) Open gmail.com 2) Enter username and pwd 3) Click Login button | Username: Arun Pwd: \*\*arun | Display error msg "Invalid userid or pwd" |  |  |  |
| TC1.4 | Validate with Incorrect username and Incorrect pwd | C | 1) Open gmail.com 2) Enter username and pwd 3) Click Login button | Username: Aruna Pwd: \*\*arun | Display error msg "Invalid userid or pwd" |  |  |  |

Test case optimization technique.

1. **Equivalence Partitioning:**
   * Identify input ranges that are expected to behave the same way.
   * Create test cases that cover each partition, reducing redundancy and increasing coverage.
2. **Boundary Value Analysis:**
   * Focus on testing values at the edges of input domains.
   * Test cases for minimum, maximum, and values just beyond the boundaries can help uncover boundary-related defects.
3. **Pairwise Testing:**
   * Instead of testing all possible combinations of input parameters, focus on testing all possible pairs.
   * Reduces the number of test cases significantly while maintaining a good level of coverage.
4. **Orthogonal Array Testing:**
   * Similar to pairwise testing, but extends to cover interactions among three or more variables.
   * Select an orthogonal array based on the number of parameters and their possible values.
5. **Risk-Based Testing:**
   * Prioritize test cases based on the perceived risk of failure in specific functionalities or modules.
   * Focus on critical and high-risk areas to ensure that testing efforts are concentrated where they are most needed.
6. **Code Coverage Analysis:**
   * Use tools to analyze code coverage during testing.
   * Identify areas of the code that have not been exercised and create additional test cases to cover those gaps.
7. **Mutation Testing:**
   * Introduce small changes (mutations) to the code and see if the test cases can detect these changes.
   * Helps ensure that the test suite is sensitive to changes in the code.
8. **Combinatorial Testing:**
   * Identify combinations of input parameters that are likely to cause defects.
   * Use combinatorial test design tools to generate test cases covering various combinations efficiently.
9. **Use Case Prioritization:**
   * Prioritize test cases based on the importance of specific use cases in the software.
   * Ensure that critical functionalities are thoroughly tested before less critical ones.
10. **Regression Test Selection:**
    * Identify and execute only those test cases that are affected by recent code changes.
    * Minimize the time and resources required for regression testing while ensuring that modified code is adequately tested.

## Sample Requirements Traceability Matrix (RTM)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Requirement ID | Requirement Description (Business Requirement / User Story) | Design Element | Test Case ID(s) | Defect ID(s) |
| BR-001 | Customers must be able to register for an account. | Registration form | TC-001: Successful registration | DE-001: Registration form error |
| BR-002 | Customers must be able to log in to their accounts. | Login form | TC-002: Successful login | DE-002: Login form error (invalid credentials) |
| US-001 | As a customer, I want to browse products by category so I can easily find what I'm looking for. | Product category navigation | TC-003: Browse products by category | N/A |
| US-002 | As a customer, I want to view detailed product information including images, descriptions, and prices before making a purchase. | Product detail page | TC-004: View product details | DE-003: Missing product image |
| US-003 | As a customer, I want to add items to my shopping cart so I can purchase them later. | Shopping cart functionality | TC-005: Add item to cart | TC-006: Remove item from cart |
| US-004 | As a customer, I want to checkout and pay for my order using a secure payment gateway. | Secure checkout process | TC-007: Successful checkout with credit card | TC-008: Checkout process with different payment methods |
| BR-003 | The website must display product availability in real-time. | Inventory management system integration | TC-009: Verify product availability | N/A |

**Sample Defect Status Report for E-commerce Website Project (Based on the RTM)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Defect ID | Description | Date Reported | Status | Assigned To | Resolution | Target Resolution Date |
| DE-001 | Registration form error (e.g., missing field validation) | 2024-03-05 | Open | Developer A | Under investigation | 2024-03-10 |
| DE-002 | Login form error (e.g., invalid credentials message not clear) | 2024-03-07 | In Progress | Developer B | Fix implemented, testing in progress | 2024-03-09 |
| DE-003 | Missing product image | 2024-03-06 | Fixed | QA Tester | Image added, functionality verified | 2024-03-07 |
| DE-004 | Cannot add item to cart (e.g., functionality not working) |  | New | To be assigned | Awaiting further investigation | 2024-03-08 |

Day 3

Q1

// You are using Java

import java.util.\*;

class Cart

{

public List<Item> items; // create an list items object

public Cart()

{ items= new ArrayList<>(); //create an empty List

}

public void addItem(Item item)

{ items.add(item); // adds item object in items cart

}

public void removeItem(Item item)

{ items.remove(item); // adds item object in items cart

}

public double calc\_Total()

{ double totalPrice=0.0;

for(Item item: items)

{ totalPrice+=item.getPrice();

}

return totalPrice;

}

public List<Item> getItems()

{ return items;}

}

class Item

{ public String name;

public double price;

public Item(String name, double price) //setting created item's name, price

{ this.name= name;

this.price=price;

}

public String getName()

{ return name;}

public double getPrice()

{ return price;}

}

public class Main{

public static void main(String args[])

{ Scanner scan= new Scanner(System.in);

int numofItems=scan.nextInt();

scan.nextLine();

Cart cart=new Cart();

for(int i=1;i<=numofItems;i++)

{ String name= scan.nextLine();

double price=scan.nextDouble();

scan.nextLine();

Item item =new Item(name,price);

cart.addItem(item); //adding current item in cart

}

int numofelementstoremove= scan.nextInt();

scan.nextLine();

for(int i=1;i<=numofelementstoremove;i++)

{ String nametoremove= scan.nextLine();

for( Item item:cart.getItems())

{ if(item.getName().equals(nametoremove))

{ cart.removeItem(item);

break;

}

}

}

double totalPrice=cart.calc\_Total(); //call total

System.out.println("Total price is "+totalPrice);

}

}

**Input:**

2

Laptop

999.99

Headphones

49.99

1

Headphones

**Output**:

Total price is 999.99

Q2

// You are using Java

import java.util.\*;

class InsufficientBalanceException extends Exception

{

public InsufficientBalanceException(){

super();}

}

public class Main{

public static void main(String args[])

{Scanner scan=new Scanner(System.in);

try

{ double bal=scan.nextDouble();

double withdraw=scan.nextDouble();

if( withdraw>bal)

throw new InsufficientBalanceException();

}

catch (InsufficientBalanceException e)

{ System.out.println("Insuffiecint balance");

}

}

}

Input:

45

50

Output

Insuffiecint balance

: