**TEST Cases for CustomerRewards Component**

This document outlines the test cases for the CustomerRewards component, covering functionalities like rendering, data fetching, and user interaction.

**Test Cases:** Rendering

**Description:** This test verifies that the CustomerRewards component renders correctly without any errors.

*Test code*

test('renders CustomerRewards component correctly', () => {

  render(<CustomerRewards />);

*// No assertions needed here as rendering success implies passing test*

});

**Test Cases:** Data Fetching and Display

**Description:** This test ensures that the component fetches customer data using the fetch API and displays it on the UI.

***Test code***

test('Fetch is called and displayed in the UI', async () => {

  render(<CustomerRewards />);

  await waitFor(() => {

    expect(screen.getByText(/Harish/i)).toBeInTheDocument();

  });

});

**Test Cases:** Customer Switch and Totals Display

**Description:** This test simulates a user clicking on a customer (e.g., Harish) and verifies that the UI displays the customer's totals.

***Test code***

test('After customer switch totals is displayed', async () => {

  render(<CustomerRewards />);¬

  await waitFor(async () => {

    expect(screen.getByText(/Harish/i)).toBeInTheDocument();

    const anchorElement = screen.getByText(/Harish/i);

    userEvent.click(anchorElement);

    await waitFor(() => {

      expect(screen.getByText(/totals/i)).toBeInTheDocument();

    });

  });

});

**Test Cases for Rewards Component Function**

This document outlines the test cases for two functions from the rewards component: calculatePoints and calculateMonthlyPoints.

**Test Cases:** calculatePoints function

This function calculates the points earned based on a given purchase amount.

**Test Cases:** Returns 0 for amount less than or equal to 50

**Description**: Verifies that no points are awarded for purchases of $50 or less.

*Test code*

test('returns 0 for amount less than or equal to 50', () => {

  expect(calculatePoints(50)).toBe(0);

  expect(calculatePoints(20)).toBe(0);

});

* Returns amount minus 50 for amount between 50 and 100 (exclusive)
* Description: Tests that the function subtracts 50 from the purchase amount and returns the difference for purchases between $50 (exclusive) and $100.

test('returns amount minus 50 for amount between 50 and 100 (exclusive)', () => {

  expect(calculatePoints(70)).toBe(20);

  expect(calculatePoints(95)).toBe(45);

});

* Returns double the amount minus 150 for amount over 100
* **Description**: Ensures that purchases exceeding $100 earn double the points after subtracting a base of 150.

test('returns double the amount minus 150 for amount over 100', () => {

  expect(calculatePoints(120)).toBe(90);

  expect(calculatePoints(200)).toBe(250);

});

**CalculateMonthlyPoints function:**

This function calculates the total points earned within a specific month.

**Test Cases:** Calculates points correctly for different amounts and months.

**Description:** Uses a data-driven approach to test various scenarios with different purchase amounts, months, and expected point values.

*Test code*

describe('calculateMonthlyPoints', () => {

  test.each([

    [50, 'January', 0], *// Below 50, no points for January*

    [60, 'February', 10], *// Between 50 and 100, 10 points for February*

    [120, 'March', 90], *// Above 100, 90 points for March*

  ])('calculates points correctly for amount %i in %s', (*amount*, *month*, *expectedPoints*) => {

    const transactions = [{ amount, date: '2024-' + (new Date(month + ' 1, 2024').getMonth() + 1) + '-01' }];

    const result = calculateMonthlyPoints(transactions);

    expect(result[month]).toBe(expectedPoints);

  });

});

**Test Cases for CustomerTotalMonth Component**

This document outlines the test cases for the CustomerTotalMonth component, covering functionalities like data fetching, rendering, and error handling.

**Test Cases:** Successful Data Fetch

**Description:** This test verifies that the component fetches customer data using the usefetch hook, displays it on the UI, and logs a success message.

Test code

test('renders customer data and monthly totals correctly', async () => {

*// Mock usefetch to return mockData*

  usefetch.mockResolvedValue(mockData);

  await act(async () => {

    render(<CustomerTotalMonth />);

  });

*// Check if data fetch success message is logged*

  expect(log.warn).toHaveBeenCalledWith(' %c Data fetch Successfully !....', 'color:green; font-size:14px');

*// Check if customer names are rendered*

  expect(screen.getByText('Mathivanan')).toBeInTheDocument();

  expect(screen.getByText('Harini')).toBeInTheDocument();

*// Check if monthly totals are rendered correctly*

  expect(screen.getByText('January')).toBeInTheDocument();

  expect(screen.getByText('February')).toBeInTheDocument();

  expect(screen.getByText('657')).toBeInTheDocument(); *// Total for January*

  expect(screen.getByText('467')).toBeInTheDocument(); *// Total for February*

});

**Test Cases:** No Customer Data Available

**Description:** This test simulates a scenario where the usefetch hook returns an empty array, and the component displays an appropriate message.

*Test code*

test('handles no customer data available', async () => {

*// Mock usefetch to return empty array*

  usefetch.mockResolvedValue([]);

  await act(async () => {

    render(<CustomerTotalMonth />);

  });

*// Check if no customer data available message is rendered*

  expect(screen.getByText('Loading...')).toBeInTheDocument();

});

**Test Cases:** Data Fetch Error

**Description:** This test ensures that the component handles errors during data fetching by logging the error and potentially displaying an error message to the user (depending on implementation).

*Test code*

test('handles data fetch error', async () => {

*// Mock usefetch to throw an error*

  usefetch.mockRejectedValue(new Error('Fetch error'));

  await act(async () => {

    render(<CustomerTotalMonth />);

  });

*// Check if data fetch error message is logged*

  expect(log.error).toHaveBeenCalledWith('Custom hook Data fetch error!....');

*// You can also check if the error message is displayed to the user, if applicable*

});