"Test Report "

Project Name:

Dynamic Load Component using React

|  |  |  |  |
| --- | --- | --- | --- |
| S.no | Test Case Name | Test Case Description | Status |
| 1 | generatedNumber function should generate unique numbers | Checks whether the generatedNumber function generates unique numbers by adding them to a circular linked list. | Passed |
| 2 | CircularLinkedList class should maintain a maximum length | This test verifies that the CircularLinkedList class maintains a maximum length by appending elements and updating the length and head. | Passed |
| 3 | AdvertisementCarousel renders without errors | Renders the AdvertisementCarousel component and checks for any rendering errors. | Passed |
| 4 | AdvertisementCarousel does not display the same ad consecutively | Ensures that the AdvertisementCarousel component displays different advertisements consecutively. | Passed |
| 5 | AdvertisementCarousel changes ad after specified delay | Checks whether the AdvertisementCarousel component changes the displayed advertisement after the specified delay. | Passed |

Steps Involved in each Test Case.

Test Case 1: generatedNumber Function Unique Number Generation

1. Create an instance of the CircularLinkedList class.
2. Set the maximum length of the circular list to be tested.
3. Create an empty array called generatedNumbers to store the generated numbers.
4. Loop through a range of numbers from 0 to advertisements.length - 4.
5. Inside the loop, call the generatedNumber function with the advertisements array and the circular list instance.
6. Append the generated number to the generatedNumbers array.
7. After the loop, create a Set from the generatedNumbers array to remove duplicates.
8. Verify that the size of the uniqueNumbers set is equal to the length of the generatedNumbers array.
9. If the sizes match, it means all generated numbers are unique.

Test Case 2: CircularLinkedList Class Maximum Length Verification

1. Create an instance of the CircularLinkedList class.
2. Set the maximum length of the circular list to be tested.
3. Loop through a range of numbers from 1 to advertisements.length - 3.
4. Inside the loop, append each number to the circular list.
5. After the loop, verify that the length of the circular list is equal to the expected maximum length.
6. Verify that the head of the circular list is correctly updated after exceeding the maximum length.

Test Case 3: AdvertisementCarousel Rendering without Errors

1. Render the AdvertisementCarousel component with the specified advertisements, name, and delay.
2. If the component renders without throwing any errors, the test case is considered successful.

Test Case 4: AdvertisementCarousel Non-Consecutive Advertisements Display

1. Render the AdvertisementCarousel component with the specified advertisements, name, and delay.
2. Create an empty array called displayedAds to track the sequence of displayed advertisements.
3. Get the image elements with the alt text 'Advertisement'.
4. Iterate over the image elements and push their src attribute to the displayedAds array.
5. Wait for the specified delay.
6. Get the new image elements after the delay.
7. Iterate over the new image elements and compare the src attribute with the corresponding element in the displayedAds array.
8. Ensure that the current advertisement is not the same as the previous one.
9. Push the current advertisement to the displayedAds array for future comparisons.

Test Case 5: AdvertisementCarousel Advertisement Change after Delay

1. Render the AdvertisementCarousel component with the specified advertisements, name, and delay.
2. Get the image elements with the alt text 'Advertisement'.
3. Retrieve the src attribute of the first image element as the initial advertisement.
4. Wait for the specified delay.
5. Get the updated image elements after the delay.
6. Verify that the src attribute of the first image element is not equal to the initial advertisement.
7. Ensure that the updated advertisement exists in the original advertisements array.