JavaScript Reverse Linked List

Challenge

Given the `head` of a singly linked list, reverse the list, and return the reversed list.

1st Example

```
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]
```

2nd Example

```
Input: head = [1,2]
Output: [2,1]
```

3rd Example

```
Input: head = []
Output: []
```

Constraints

- `-5000 <= Node.val <= 5000`
- The number of nodes in the list is the range `[0, 5000]`.

Solution

```
const reverseList = (head) => {
    let backward = null,
        forward  = head;

    while (forward) {
        let n = forward.next;

        forward.next = backward;
        backward     = forward;
        forward      = n;
    }

    return backward;
};
```

Explanation

I've created a function called `reverseList` that takes a linked list as input and returns the reversed version of the list.

Inside the function, two variables `backward` and `forward` are declared and initialized. `backward` is set to `null`, indicating the initial reversed list is empty, and `forward` is set to the `head` node, representing the current node being processed.

A while loop is used to iterate through the linked list until `forward` becomes `null`, indicating the end of the list has been reached.

Inside the loop, a temporary variable `n` is assigned the value of

`forward.next`. This is done to store the reference to the next node before modifying it.

The `next` property of the `forward` node is then set to `backward`, effectively reversing the link between the current node and the previous node.

The `backward` variable is updated to hold the reference to the current node, which becomes the new `backward` node in the reversed list.

The `forward` variable is updated to hold the reference to the next node (`n`), allowing the loop to move forward to the next node in the original list.

Once the loop finishes, the reversed list is fully constructed, and the `backward` variable holds the reference to the new head node of the reversed list.

Finally, the `backward` variable is returned as the result of the function, representing the head node of the reversed list.

In summary, this function takes a linked list as input and reverses the order of the nodes in the list. It does this by iterating through the original list, updating the links between nodes to reverse the order, and returning the head node of the reversed list.

Author: Trevor Morin