

Serino Code Review and Technical Exam Answers Explanation

1. Question 1

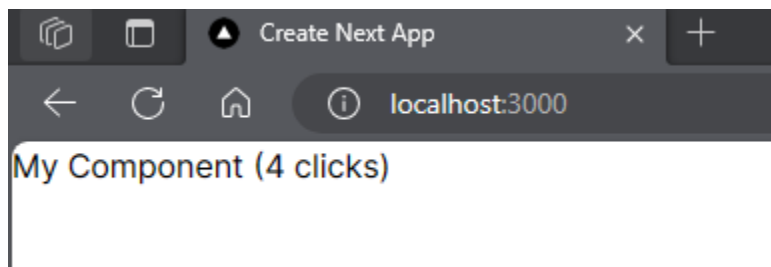
a) Issues I found in the original code:

- Binding issue in "clickHandler"
 - In the original code, the "clickHandler" function is not explicitly bound to the class instance. In Javascript, the value of "this" inside a function is determined by how the function is called. In a class, the value of "this" inside a function will refer to the instance of the class. Because the "clickHandler" is passed as an event listener, the binding of "this" can lead to errors when trying to access "this.setState".
- Incorrect state update in "clickHandler"
 - With the original code, the intention was to increment the "clicks" state by 1. However, the code is trying to access "this.clicks", which is wrong. To access the current state and update it, it should be "this.state.clicks + 1"
- These codes doesn't do anything since no props are being passed.
 - `<h3>{props.headerText}</h3>`
 - `{props.children}`

b) Explanation and reason for refactoring.

- The original code is written in a class structure. While this is still valid, using functional components along with hooks for state management provide an easier way of managing state and makes the code more readable.

Here's what it looks like once the code is executed.



I also created a unit test to check the reliability of my code. This has 3 test cases.

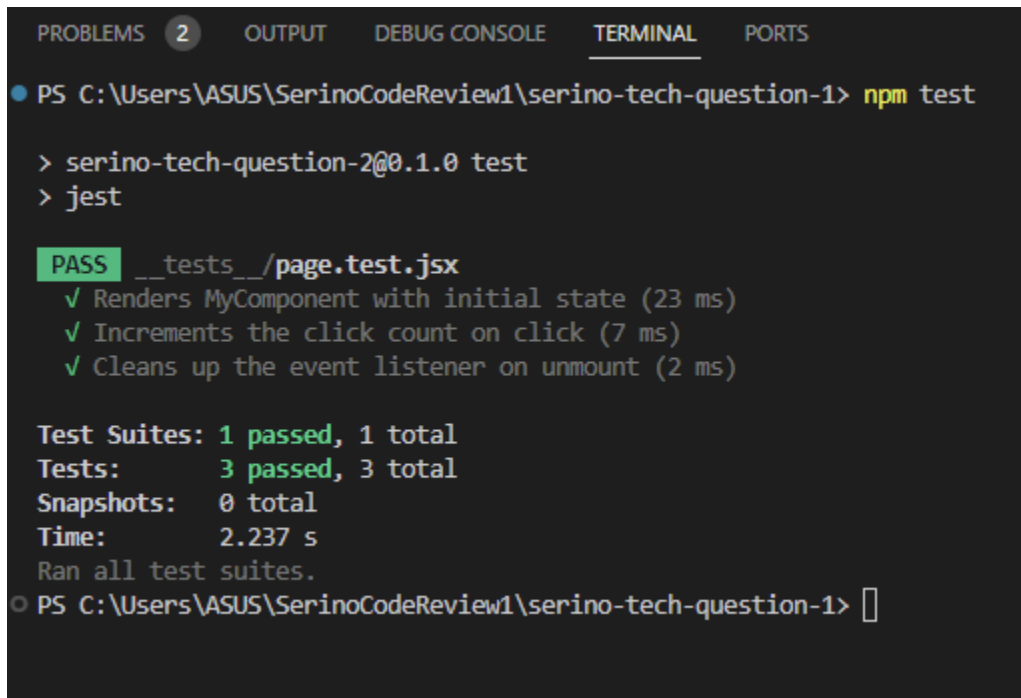
1. Test if it renders the component with initial state.
2. Test if the click count increments on click.
3. Test for cleaning up event listener when it's unmounted (for preventing unexpected behavior)

Here's a screenshot for reference:

```
serino-tech-question-1 > __tests__ > page.test.jsx > test('Cleans up the event listener on unmount') callback

1  import { render, fireEvent, cleanup, act } from '@testing-library/react';
2  import '@testing-library/jest-dom';
3  import MyComponent from '../app/page';
4
5  // Cleanup after each test
6  afterEach(cleanup);
7
8  test('Renders MyComponent with initial state', () => {
9    // Render the component
10   const { getByText } = render(<MyComponent headerText="Test Header">Test Content</MyComponent>);
11
12   // Check if the initial state is rendered
13   expect(getByText('My Component (0 clicks)')).toBeInTheDocument();
14 });
15
16 test('Increments the click count on click', () => {
17   // Render the component
18   const { getByText, getByTestId } = render(<MyComponent headerText="Test Header">Test Content</MyComponent>);
19
20   // Click the component
21   fireEvent.click(getByTestId('my-component'));
22
23   // Check if the click count has increased
24   expect(getByText('My Component (1 clicks)')).toBeInTheDocument();
25 });
26
27
28 test('Cleans up the event listener on unmount', () => {
29   // Render the component
30   const { unmount } = render(<MyComponent headerText="Test Header">Test Content</MyComponent>);
31
32   // Manually trigger the cleanup function
33   unmount();
34 });
```

And here's the result of the unit test:



```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
● PS C:\Users\ASUS\SerinoCodeReview1\serino-tech-question-1> npm test

> serino-tech-question-2@0.1.0 test
> jest

PASS __tests__/page.test.jsx
  ✓ Renders MyComponent with initial state (23 ms)
  ✓ Increments the click count on click (7 ms)
  ✓ Cleans up the event listener on unmount (2 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        2.237 s
Ran all test suites.
○ PS C:\Users\ASUS\SerinoCodeReview1\serino-tech-question-1> []
```

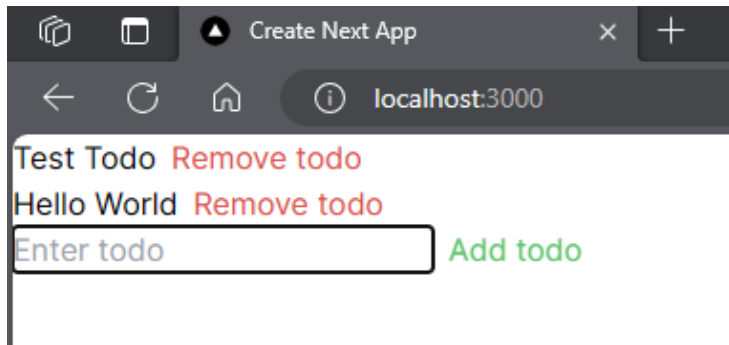
2. Question 2

To complete the **“ToDoList”** component, it is necessary to update the **“todosReducer”**, which serves as a reducer function responsible for handling state changes based on actions such as **“ADD_TODO”** and **“REMOVE_TODO”**. Additionally, within the **“ToDoList”** component, two essential functions need to be implemented.

- **addTodo:**
 - This function manages the addition of a new todo item. It is triggered when the user interacts with the UI to add a new todo. Inside this function, the **“ADD_TODO”** action is dispatched, providing the new todo's text as a payload. The added state, **“newTodo”**, ensures that the input value for adding a new todo is appropriately managed.
- **removeTodo:**
 - This function is responsible for removing a specific todo item. When the user interacts with the UI to remove a todo, the **“REMOVE_TODO”** action is dispatched, including the unique identifier (id) of the todo item as payload. This ensures that the correct todo is targeted for removal.

Additionally, the **key** prop is introduced to each list item when mapping over the **todos** array. This **key** prop assists React in efficiently updating the UI when items are added or removed.

Here's what it looks like when the code is executed.

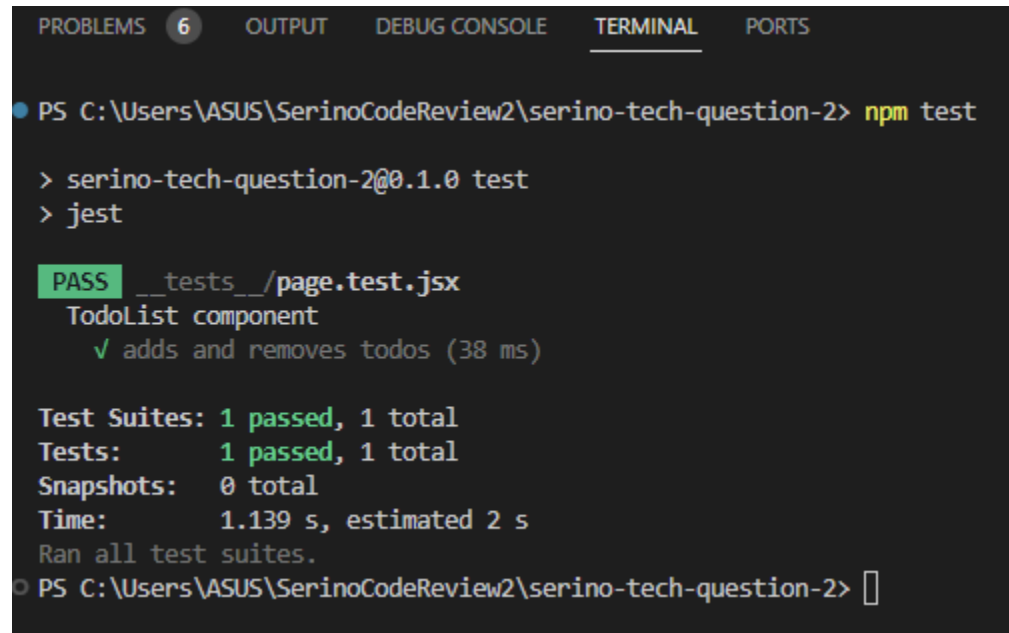


I also created a unit test to check the reliability of my code. This test case simulates user interaction to add and remove a todo from the "TodoList" component. It then verifies whether the added todo is present in the document and if it gets correctly removed. Please see screenshot of the unit test for reference. Below it is the result of the unit test.

```
serino-tech-question-2 > __tests__ > page.test.jsx > describe('TodoList component') callback > test('Adds and removes todos') callback

1  import { render, screen, fireEvent, waitFor } from '@testing-library/react';
2  import '@testing-library/jest-dom';
3  import TodoList from '../app/page';
4
5  describe('TodoList component', () => {
6
7    test('Adds and removes todos', async () => {
8
9      // Render the TodoList component
10     render(<TodoList />);
11
12     // Get input field and "Add todo" button
13     const inputField = screen.getByPlaceholderText('Enter todo');
14     const addButton = screen.getByText('Add todo');
15
16     // Simulate adding a new todo
17     fireEvent.change(inputField, { target: { value: 'Test Todo' } });
18     fireEvent.click(addButton);
19
20     // Check if the added todo is in the document
21     const addedTodo = screen.getByText('Test Todo');
22     expect(addedTodo).toBeInTheDocument();
23
24     // Get the "Remove todo" button and simulate clicking it
25     const removeButton = screen.getByText('Remove todo');
26     fireEvent.click(removeButton);
27
28     // Wait for the removal to complete and check if the todo is no longer in the document
29     await waitFor(() => {
30       const removedTodo = screen.queryByText('Test Todo');
31       expect(removedTodo).not.toBeInTheDocument();
32     });
33
34   });
35
36 });
```

Results of unit test:



```
PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS C:\Users\ASUS\SerinoCodeReview2\serino-tech-question-2> npm test

> serino-tech-question-2@0.1.0 test
> jest

PASS ___tests___/page.test.jsx
  TodoList component
    ✓ adds and removes todos (38 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.139 s, estimated 2 s
Ran all test suites.
● PS C:\Users\ASUS\SerinoCodeReview2\serino-tech-question-2> 
```

3. Question 3

To solve this problem, I created a Javascript function named “groupByStatus” which takes an array of objects as input, where each object represents an entity with “id”, “name”, and “status”. The function organizes this input data into a new structure based on the “status” field, creating an object where keys are in the format “status-<status value>” and values are arrays containing objects with the corresponding status. For example, all entities with a status of 1 are grouped under the key “status-1”, and similarly for other status values.

The code uses a forEach loop to iterate through the input data, generating keys based on the “status” field and attaching the entities to the respective arrays within the result object. If a status key doesn’t exist in the result object, it is created along with an empty array. The final output is an object representing the input data grouped by status. Additionally, I added comments on my code to provide a clear understanding of each section’s purpose and functionality. This structure is modular and can be used to organize and analyze data based on different status values. The accompanying unit test ensures that the function works as expected by comparing its output against the given expected result.

Here's a screenshot of the unit test.

```
... JS serinoTechExam.js JS groupByStatus.test.js X
JS groupByStatus.test.js > test('groups data by status correctly') callback
1 // Import the function to be tested
2 const groupByStatus = require('./serinoTechExam.js');
3
4 // Test case 1
5 test('groups data by status correctly', () => {
6   const initialData = [
7     {
8       "id": 1,
9       "name": "John Doe",
10      "status": 1
11    },
12    {
13      "id": 2,
14      "name": "Jane Doe",
15      "status": 2
16    },
17    {
18      "id": 3,
19      "name": "Adam Rocket",
20      "status": 2
21    },
22    {
23      "id": 4,
24      "name": "Luis Rocket",
25      "status": 1
26    }
27  ];
28
29   const expectedResult = {
30     "status-1": [
31       {
32         "id": 1,
33         "name": "John Doe",
34         "status": 1
35       },
36       {
37         "id": 4,
38         "name": "Luis Rocket",
39         "status": 1
40       }
41     ],
42     "status-2": [
43       {
44         "id": 2,
45         "name": "Jane Doe",
46         "status": 2
47       },
48       {
49         "id": 3,
50         "name": "Adam Rocket",
51         "status": 2
52       }
53     ]
54   };
55
56   // Call the function with the initial data
57   const result = groupByStatus(initialData);
58
59   // Check if the result matches the expected result
60   expect(result).toEqual(expectedResult);
61 });
```

This is the result of the unit test:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\ASUS\code\SerinoExam> npx jest

console.log

```
{
  'status-1': [
    { id: 1, name: 'John Doe', status: 1 },
    { id: 4, name: 'Luis Rocket', status: 1 }
  ],
  'status-2': [
    { id: 2, name: 'Jane Doe', status: 2 },
    { id: 3, name: 'Adam Rocket', status: 2 }
  ]
}
```

at Object.log (serinoTechExam.js:52:11)

PASS ./groupByStatus.test.js

✓ groups data by status correctly (3 ms)

Test Suites: 1 passed, 1 total

Tests: 1 passed, 1 total

● Snapshots: 0 total

Time: 0.648 s, estimated 1 s

Ran all test suites.

PS C:\Users\ASUS\code\SerinoExam>