Solution: Writing more test scenarios

Here is the completed solution code for the App.test.js file:

```
import { fireEvent, render, screen } from "@testing-library/react";
import FeedbackForm from "./FeedbackForm";

describe("Feedback Form", () => {
  test("User is able to submit the form if the score is lower than 5 and additional feedback is prov
    const score = "3";
    const comment = "The pizza crust was too thick";
    const handleSubmit = jest.fn();
    render(<FeedbackForm onSubmit={handleSubmit} />);

    const rangeInput = screen.getByLabelText(/Score:/);
    fireEvent.change(rangeInput, { target: { value: score } });

    const textArea = screen.getByLabelText(/Comments:/);
    fireEvent.change(textArea, { target: { value: comment } });

    const submitButton = screen.getByRole("button");
    fireEvent.click(submitButton);

    expect(handleSubmit).toHaveBeenCalledWith({
      score,
      comment,
    });
  });

  test("User is able to submit the form if the score is higher than 5, without additional feedback",
    const score = "9";
    const handleSubmit = jest.fn();
    render(<FeedbackForm onSubmit={handleSubmit} />);

    const rangeInput = screen.getByLabelText(/Score:/);
    fireEvent.change(rangeInput, { target: { value: score } });

    const submitButton = screen.getByRole("button");
    fireEvent.click(submitButton);

    expect(handleSubmit).toHaveBeenCalledWith({
      score,
      comment: ""
    });
```

Steps

Step 1

The first test scenario has the following specification:

User is able to submit the form if the score is lower than 5 and additional feedback is provided

The test scenario already contains some initial code that acts as boilerplate before getting to the bulk of the test, in particular:

- Two variables that hold the desired state of the form, a score of 3 and an additional comment.

- A mock function that is called when submitting the form.
- The rendering of the form component.
- The final assertion that should make the test pass.

If you run as it is, the test will fail stating that the mock function has not been called at all. That is because no interactions have occurred yet and it's your task to write those.

```
expect(jest.fn()).toHaveBeenCalledWith(...expected)

Expected: {"comment": "The pizza crust was too thick", "score": "3"}

Number of calls: 0

  11 |     // You have to write the rest of the test below to make the assertion pass
  12 |
> 13 |     expect(handleSubmit).toHaveBeenCalledWith({
     |                          ^
  14 |       score,
  15 |       comment,
  16 |     });

  at Object.<anonymous> (src/App.test.js:13:26)
```

The first user interaction that needs to happen is to set the score as 3. The following code achieves that:

```
const rangeInput = screen.getByLabelText(/Score:/);
fireEvent.change(rangeInput, { target: { value: score } });
```

The first line grabs a reference to the range input component by using the global screen object from react-testing-library and the query `getByLabelText` to find a label that contains the exact text `Score:`

Then, a change event is simulated on the input, passing as the event an object with the `value` property set to the variable `score`: `event.target.value = score`

After that, a second user interaction is required to set the additional comment. This is the code that accomplishes that:

```
const textArea = screen.getByLabelText(/Comments:/);
fireEvent.change(textArea, { target: { value: comment } });
```

No further explanation is needed regarding those two lines, since they mimic the same interaction as with the range input.

Last but not least, a submission of the form should be simulated by calling the below two lines:

```
const submitButton = screen.getByRole("button");
fireEvent.click(submitButton);
```

In this particular instance, the button is referenced by using a different query on the global screen object, `getByRole`.

This query looks for an element whose role attribute is set to `"button"`, which is inherent in all button HTML tags.

The form is finally submitted via firing a click event on the button instance.

If you run the command `npm test` in your terminal, the test should pass now.

Let's now cover the second scenario.

User is able to submit the form if the score is higher than 5, without additional feedback

The below represents the code you need to write to make the test pass.

```
const rangeInput = screen.getByLabelText(/Score:/);
fireEvent.change(rangeInput, { target: { value: score } });

const submitButton = screen.getByRole("button");
fireEvent.click(submitButton);
```

This test is simpler and there is nothing new besides skipping any interaction with the text area, since no additional feedback is required when the score provided is higher than 5, being 9 in this scenario.

Step 2

If you run `npm test` after adding the lines above, both of your tests should pass successfully, with the terminal providing the below output.

```
 PASS  src/App.test.js
  Feedback Form
    ✓ User is able to submit the form if the score is lower than 5 and additional feedback is provided (78 ms)
    ✓ User is able to submit the form if the score is higher than 5, without additional feedback (20 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        2.379 s
Ran all test suites related to changed files.

Watch Usage
 › Press a to run all tests.
 › Press f to run only failed tests.
 › Press q to quit watch mode.
 › Press p to filter by a filename regex pattern.
 › Press t to filter by a test name regex pattern.
 › Press Enter to trigger a test run.
```