DMIT2008 Assignment 5: Automated Testing on Dinder

Introduction

This assignment will test your knowledge of writing automated tests. This assignment will only cover the section “Testing React Next.js Applications” and your general knowledge of React and Javascript as whole. Why is testing important to learn? As your applications get used by more people and you continue to add features, testing is a way to make your site more stable and a way for you to provide a more dependable service.

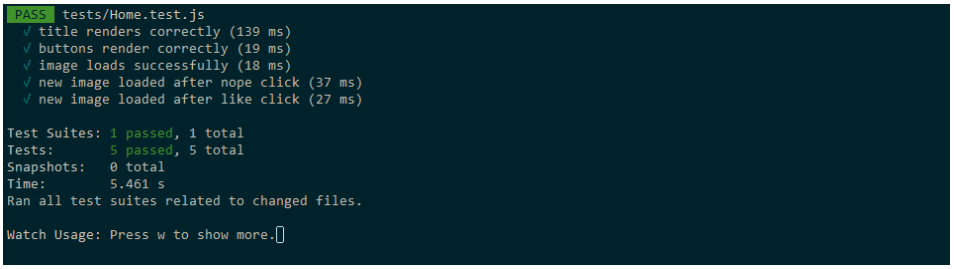
In this assignment we’ll be using “mocking” techniques to mock out the Dog API. To test we’ll be using Jest to mock out the REST API we’ll be using isomorphic fetch, and the mock service worker.

Overview of functionality

- Once the project is built and installed you should see something like the following.

  You’ll note that if you click “like” or “nope” there will be a new dog loaded.
- Your job is to write automated tests. When you run those automated tests it should look like the image below (if successful):

- Bonus: you should be able to write two more tests that test to see if the image is reloaded after you click the “like” button and the “nope” button.



Required Tasks

- Install and run the starter project, install libraries for testing, and add jest configuration.
    - Packages to install
        - Jest
        - React testing library
        - Isomorphic fetch
        - Mock service worker
    - Configure Jest.
        - Create a next.config.js that will allow us to run jest with our next application. Documentation reference for jest and next.js here
    - Write a script that that will run “npx jest --watch” and call that script “test”
- Create Home Test file and setup the Mock Service Worker so our application does not make external REST API requests.
    - Create a file named Home.test.js in the folder named “tests”.
    - Import the Home component and all other required testing modules.
    - Setup the server from mock service worker.
        - This should take one argument that is “http.get” to the the endpoint https://dog.ceo/api/breeds/image/random. The mocked response should take the format where the “TEST_URL” is a constant that is a mock url, you’ll need to user “HttpResponse.json” for the response as well.

```
{
    "status":"success",
    "message":TEST_URL
}
```

        - Ensure that the server is set up to “listen” before all of the tests.
        - Ensure that the server is “closed” after all of the tests are done.
- Write a test that checks to see if the title is rendered correctly.
    - Using jest define a test (use and async function)
        - Use “await” the “act” function that should use the “render” the “home” component.
        - Using “screen” to get the title by text (using getByText).
        - Using “expect” check if the element selected above is in the document.
- Write a test that checks to see if the buttons are rendered correctly

    - Using jest define a test (use and async function)
        - Use “await” the “act” function that should use the “render” the “home” component.
        - Using “screen” to get the nope button and the like button by text (using getByText).
        - Using “expect” check if the elements selected above are in the document.
- Write a test that checks to see if the image is loaded successfully.
    - Using jest define a test (use and async function)
        - Use “await” the “act” function that should use the “render” the “home” component.
        - Using “screen” get the image element by the test id. The test id is the data-testid prop on the image element in index.js.
        - Using “expect” check if the image selected above is in the document.
        - Using “expect” check if the src of the image is equal to the TEST_URL you have defined.
- Bonus:
    - Write a test that will check if a new image is loaded after clicking “Like”
    - Write a second test that will check if a new image is loaded after clicking “Nope”

Marking key

Tasks	Grade	Marks	Total
Testing Libraries installed and configured correctly. • Testing packages installed correctly • Next.config.js added to the right place and uses correct configuration. • Mock Service worker server is setup correctly. Server is setup with the correct mock url and the response is mocked correctly. • Mock service worker listens before all of the tests and is close after all of the tests are completed. • Package.json is configured correctly so that tests can be run with “npm run test”		**1** **1** **5** **3** **1**	
Title Test • Jest test is used correctly and uses an async function to test, description for title test is correct. • “Act” is “await”ed and home is rendered in the “act” callback. • Correct title element selected • Expect syntax used correctly and “toBeInTheDocument” is chained to the expect correctly.		**1** **3** **1** **3**	
Buttons Test • Jest test is used correctly and uses an async function to test, description for button rendering test is correct. • “Act” is “await”ed and home is rendered in the “act” callback. • Correct title element selected • Expect syntax used correctly and “toBeInTheDocument” is chained to the expect correctly.		**1** **3** **1** **3**	

Image Loading on render Test • Jest test is used correctly and uses an async function to test, description for button rendering test is correct. • "Act" is "await"ed and home is rendered in the "act" callback. • Correct image element selected using the function "getByTestId" • Expect syntax used correctly and "toBeInTheDocument" is chained to the expect correctly. • A Second Expect is used to ensure that the "src" attribute of the image element is equal to the mock value we passed into the mock service worker.		**1** **3** **1** **3** **3**	
Running the Jest Test Suite • Test suite passes and there are no errors (all green check marks), and there is no extra console.log messages.		**-3**	
Bonus • Write a test that will check if a new image is loaded after clicking "Like" • Write a second test that will check if a new image is loaded after clicking "Nope"		**[3]**	
Formatting and Style • Code Formatting and Style (run "npm run lint") and fix any errors that come up. • Incorrect implementation of the testing environment.		**-3** **-5**	

Marking Rubric

Marks	5 Marks Criteria
5	**Task was completed with the highest of proficiency adhering to best practices and followed subject matter guidelines all tasks were completed to a professional standard.**
4	**Task was completed well some minor mistakes. Well above average work shows good understanding of the task and high degree of competence**
3	**Satisfactory work some features missing or incorrectly implemented. Show a moderate level of understanding in the task with room for improvement.**
2	**Below average work. Task was poorly complete. Show understanding of the task and the requirements to implement but implementation was poorly executed.**
1	**Some of the task was completed. Showed a lack of understanding in the subject matter and very poorly executed**
0	**Not completed.**

Marks	3 Marks Criteria
3	Proficient shows a high degree of competence in completing task.
2	Capable above average degree of competence in completing task
1	Satisfactory shows a satisfactory degree of competence in completing task.
0	Shows a limited degree of competence in completing task.

Marks	1 Marks Criteria
1	Task Completed satisfactorily
0	Task was not executed.