**Testing**

During the testing stage of software development, the application's actual outcomes are compared to those that were expected.

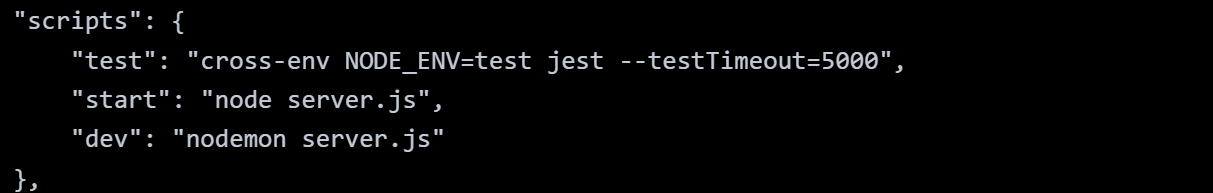
As the name implies, unit testing involves testing individual pieces of code or an application. For instance, in REST API, we are able to create distinct tests for each endpoint, and in a ReactJs application, we can test each component separately.

To start writing tests, you need three npm packages: jest, supertest, and cross-env.

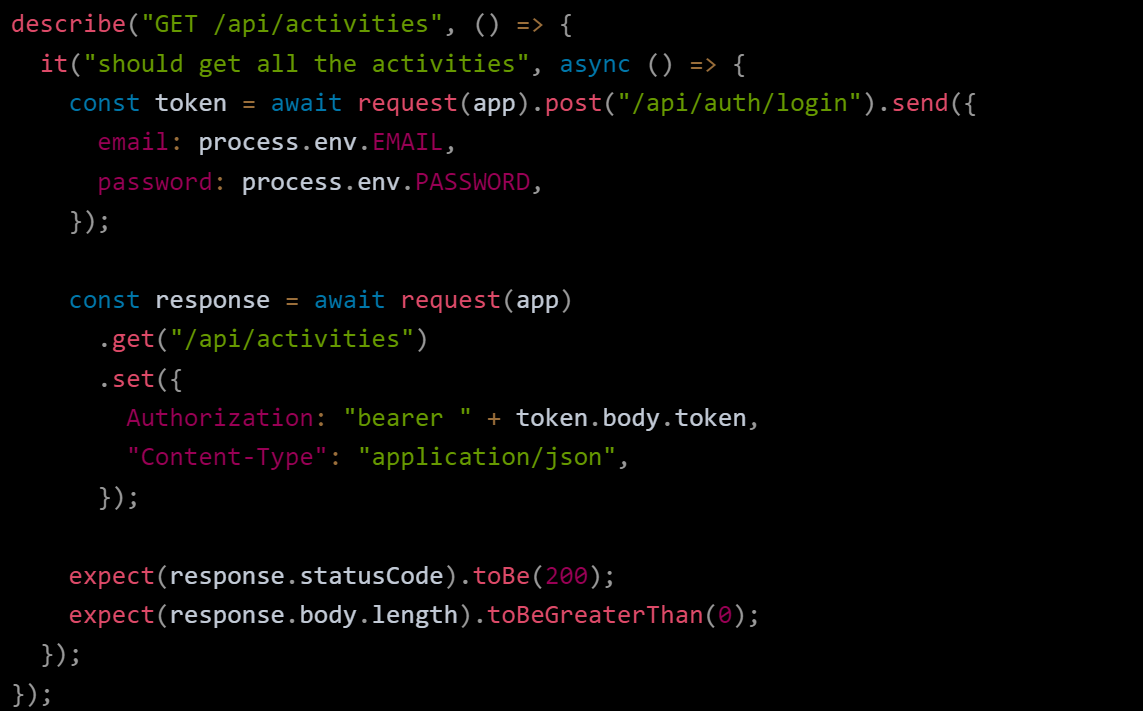
npm i --save-dev jest supertest cross-env

**jest**: Jest is a framework for testing JavaScript code. Unit testing is its main usage of it. **supertest**: Using Supertest, we can test endpoints and routes on HTTP servers. **cross-env**: You can set environmental variables inline within a command using cross-env.

Open your package.json file and add the test script to the scripts.



In this case, testTimeout is set to 5000 because it's possible for certain requests to take a while to complete, and cross-env is being used to set environment variables and jest to run test suites.

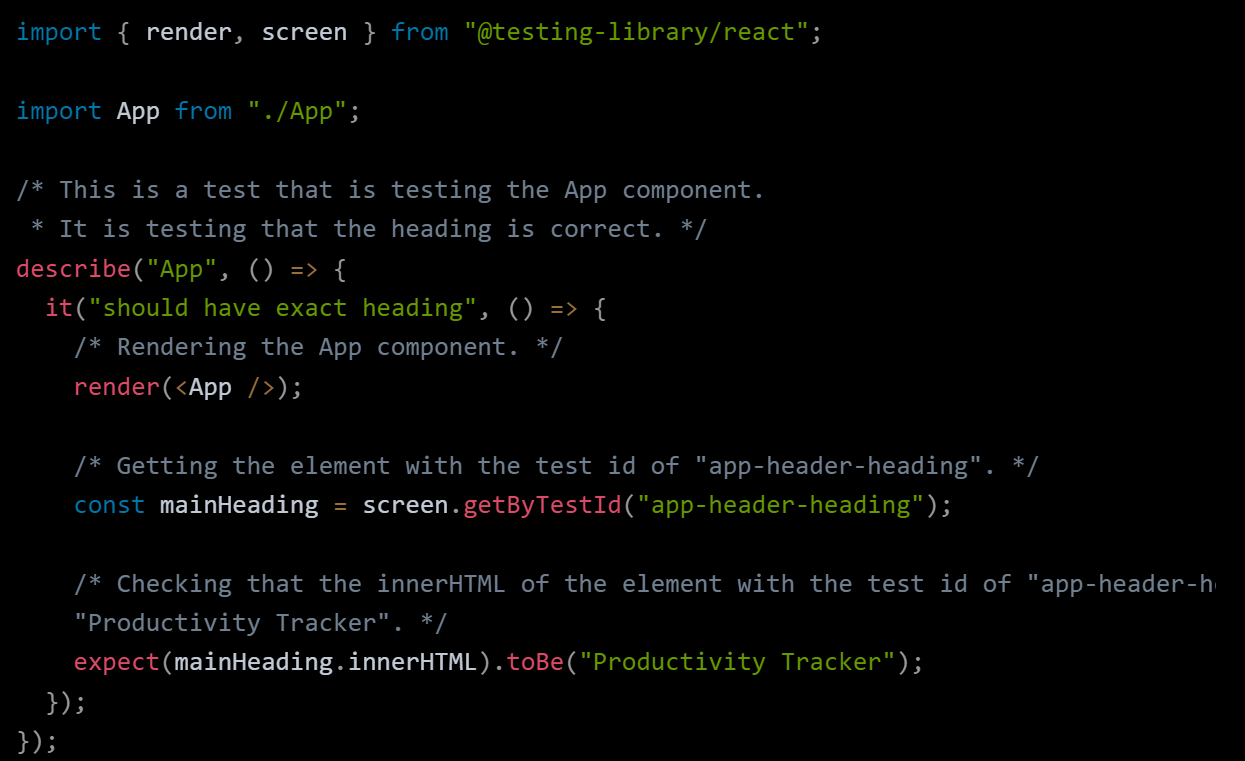
In the above code,

* We use **describe** to describe the unit test. Even though it is not required, it will be useful to identify tests in test results.
* In **it**, we write the actual test code. Write the expected result in the first argument, and then in the second argument, write a callback function that contains the test code.
* In the callback function, the request is sent to the endpoint first, and the **expected** and actual responses are then compared. The test passes if both answers match, else, it fails.

It's a good practice to create a test file for every component.

We have 2 components that need to be tested - <App /> and <Login />. So create App.test.js and Login.test.js in the src/ folder. **But for now create only the** App.test.js file.

Now, open the App.test.js file and paste the below code.



This is the general flow of writing a test for a React component:

* Render the component -> Write elements you want to interact with -> Interact with those elements -> Assert that the outcomes are as expected.

**Cheatsheet**

[**Get the printable cheat sheet**](https://github.com/testing-library/react-testing-library/raw/main/other/cheat-sheet.pdf)

A short guide to all the exported functions in React Testing Library

* **render** const {/\* \*/} = render(Component) returns:
  + unmount function to unmount the component
  + container reference to the DOM node where the component is mounted
  + all the queries from DOM Testing Library, bound to the document so there is no need to pass a node as the first argument (usually, you can use the screen import instead)

*import* {render, fireEvent, screen} *from* '@testing-library/react'  
  
test('loads items eventually', *async* () => {  
 render(<Page />)  
  
 *// Click button*  
 fireEvent.click(screen.getByText('Load'))  
  
 *// Wait for page to update with query text*  
 *const* items = *await* screen.findAllByText(/Item #[0-9]: /)  
 expect(items).toHaveLength(10)  
})

**Queries**[**​**](https://testing-library.com/docs/react-testing-library/cheatsheet#queries)

**Difference from DOM Testing Library**

The queries returned from render in React Testing Library are the same as DOM Testing Library except they have the first argument bound to the document, so instead of getByText(node, 'text') you do getByText('text')

See [Which query should I use?](https://testing-library.com/docs/queries/about#priority)

|  | **No Match** | **1 Match** | **1+ Match** | **Await?** |
| --- | --- | --- | --- | --- |
| **getBy** | throw | return | throw | No |
| **findBy** | throw | return | throw | Yes |
| **queryBy** | null | return | throw | No |
| **getAllBy** | throw | array | array | No |
| **findAllBy** | throw | array | array | Yes |
| **queryAllBy** | [] | array | array | No |

* **ByLabelText** find by label or aria-label text content
  + getByLabelText
  + queryByLabelText
  + getAllByLabelText
  + queryAllByLabelText
  + findByLabelText
  + findAllByLabelText
* **ByPlaceholderText** find by input placeholder value
  + getByPlaceholderText
  + queryByPlaceholderText
  + getAllByPlaceholderText
  + queryAllByPlaceholderText
  + findByPlaceholderText
  + findAllByPlaceholderText
* **ByText** find by element text content
  + getByText
  + queryByText
  + getAllByText
  + queryAllByText
  + findByText
  + findAllByText
* **ByDisplayValue** find by form element current value
  + getByDisplayValue
  + queryByDisplayValue
  + getAllByDisplayValue
  + queryAllByDisplayValue
  + findByDisplayValue
  + findAllByDisplayValue
* **ByAltText** find by img alt attribute
  + getByAltText
  + queryByAltText
  + getAllByAltText
  + queryAllByAltText
  + findByAltText
  + findAllByAltText
* **ByTitle** find by title attribute or svg title tag
  + getByTitle
  + queryByTitle
  + getAllByTitle
  + queryAllByTitle
  + findByTitle
  + findAllByTitle
* **ByRole** find by aria role
  + getByRole
  + queryByRole
  + getAllByRole
  + queryAllByRole
  + findByRole
  + findAllByRole
* **ByTestId** find by data-testid attribute
  + getByTestId
  + queryByTestId
  + getAllByTestId
  + queryAllByTestId
  + findByTestId
  + findAllByTestId

**Async**[**​**](https://testing-library.com/docs/react-testing-library/cheatsheet#async)

The [dom-testing-library Async API](https://testing-library.com/docs/dom-testing-library/api-async) is re-exported from React Testing Library.

* **waitFor** (Promise) retry the function within until it stops throwing or times out
* **waitForElementToBeRemoved** (Promise) retry the function until it no longer returns a DOM node

**Events**[**​**](https://testing-library.com/docs/react-testing-library/cheatsheet#events)

See [Events API](https://testing-library.com/docs/dom-testing-library/api-events)

* **fireEvent** trigger DOM event: fireEvent(node, event)
* **fireEvent.\*** helpers for default event types
  + **click** fireEvent.click(node)
  + [See all supported events](https://github.com/testing-library/dom-testing-library/blob/master/src/event-map.js)
* **act** wrapper around [react act](https://react.dev/reference/react/act); React Testing Library wraps render and fireEvent in a call to act already so most cases should not require using it manually

**Other**[**​**](https://testing-library.com/docs/react-testing-library/cheatsheet#other)

See [Querying Within Elements](https://testing-library.com/docs/dom-testing-library/api-within), [Config API](https://testing-library.com/docs/react-testing-library/api#configure), [Cleanup](https://testing-library.com/docs/react-testing-library/api#cleanup),

* **within** take a node and return an object with all the queries bound to the node (used to return the queries from React Testing Library's render method): within(node).getByText("hello")
* **configure** change global options: configure({testIdAttribute: 'my-data-test-id'})
* **cleanup** clears the DOM ([use with afterEach](https://testing-library.com/docs/react-testing-library/api#cleanup) to reset DOM between tests)

**Text Match Options**[**​**](https://testing-library.com/docs/react-testing-library/cheatsheet#text-match-options)

Given the following HTML:

<div>Hello World</div>

***Will* find the div:**

*// Matching a string:*  
getByText('Hello World') *// full string match*  
getByText('llo Worl', {exact: false}) *// substring match*  
getByText('hello world', {exact: false}) *// ignore case*  
  
*// Matching a regex:*  
getByText(/World/) *// substring match*  
getByText(/world/i) *// substring match, ignore case*  
getByText(/^hello world$/i) *// full string match, ignore case*  
getByText(/Hello W?oRlD/i) *// advanced regex*  
  
*// Matching with a custom function:*  
getByText((content, element) => content.startsWith('Hello'))