React Testing Notes

React Testing Notes

1. **Introduction to Testing in React**
  - Testing ensures that the application behaves as expected and helps catch bugs early.
  - Popular tools: Jest (test runner), React Testing Library (for DOM testing).

2. **Setup Jest and React Testing Library**
  - Install dependencies:
    ```
    npm install --save-dev jest @testing-library/react @testing-library/jest-dom
    ```
  - Configure Jest in `package.json` or create a `jest.config.js` file.
  - Add a script in `package.json`:
    ```json
    "scripts": {
      "test": "jest"
    }
    ```

3. **Writing Tests**
  - Create test files next to the component files (e.g.,

`Component.test.js`).

   - Import necessary functions and components:

     ```javascript
     import { render, screen } from '@testing-library/react';
     import '@testing-library/jest-dom';
     import Component from './Component';
     ```

4. **Common Queries in React Testing Library**
   - `getByText`: Finds an element by its visible text.
   - `getByRole`: Finds an element by its role (e.g., button, textbox).
   - `getByTestId`: Finds an element by a `data-testid` attribute.
   - Examples:

     ```javascript
     const button = screen.getByRole('button', { name: /submit/i });
     const title = screen.getByText(/welcome/i);
     const customElement = screen.getByTestId('custom-element');
     ```

5. **User Interaction Testing**
   - Simulate user interactions like clicking or typing using `fireEvent` or `userEvent`:

     ```javascript
     import { fireEvent } from '@testing-library/react';

     fireEvent.click(screen.getByRole('button'));
         fireEvent.change(screen.getByRole('textbox'), { target: { value:
     ```

'New value' } });

6. **Asserting Outputs**
   - Use `expect` with matchers from Jest:
     ```javascript
     expect(screen.getByText(/success/i)).toBeInTheDocument();
     expect(screen.getByRole('button')).toBeDisabled();
     ```

7. **Mocking and Spying**
   - Mock functions for API calls or event handlers using `jest.fn()`:
     ```javascript
     const mockFunction = jest.fn();
     ```
   - Mock modules for API calls:
     ```javascript
     jest.mock('./api', () => ({
       fetchData: jest.fn(() => Promise.resolve({ data: 'value' }))
     }));
     ```

8. **Snapshot Testing**
   - Capture the rendered output and compare it with a stored snapshot:
     ```javascript
     import renderer from 'react-test-renderer';
     ```

```
it('matches snapshot', () => {
  const tree = renderer.create(<Component />).toJSON();
  expect(tree).toMatchSnapshot();
});
```

9. **Tips for Writing Effective Tests**
   - Focus on testing user behavior rather than implementation details.
   - Avoid over-mocking; test the component as close to real-world usage as possible.
   - Group related tests together using `describe`.

10. **Running Tests**
    - Run all tests: `npm test` or `yarn test`.
    - Run tests in watch mode: `npm test -- --watch`.
    - Run a specific test file: `npm test -- <filename>`.

11. **Debugging Tests**
    - Use `screen.debug()` to print the current DOM state.
    - Run tests with `--verbose` flag for detailed output.

12. **Advanced Topics**
    - Testing Context and Reducers
    - Mocking API calls with `msw` (Mock Service Worker)
    - Testing React Router components