TEST PLAN REPORT FOR QUESTION COMPONENTS

Report by Sriramkumar Raja Natarajan (11038532)

1. Introduction

The test suite is designed to verify the core functionalities of question components of our application built with React. It covers the testing plan of the question components of the application, including adding and deleting questions, navigating between pages, and ensuring that the form interactions work as expected. The tests simulate user interactions such as form submissions, button clicks, and navigation, ensuring that the application's state and routing behave correctly under various conditions.

2. Test Items

- Checks that the title and description input fields are rendered correctly with their placeholders.
- Ensures that typing into the input fields updates their values.
- Validates that submitting the form triggers the *addQuestion* function, clears the inputs, and navigates the user to the posts page.
- Tests that clicking the "Back" button correctly navigate back to the previous page.
- Confirms that all the questions provided are rendered, displaying both their titles and descriptions.
- Verifies the presence of the "Ask a Question" button.
- Tests that clicking the "Ask a Question" button navigates the user to the /ask page.
- Ensures that clicking a question's "Delete" button calls the *deleteQuestion* function with the correct question ID.

3. Features to be Tested

a. ***QuestionForm* component:**
   i. Proper rendering of input fields with the correct placeholders.
   ii. Handling user input by updating input values.
   iii. Correct navigation upon successful submission.

b. ***QuestionsList* component:**
   i. Rendering a list of questions including displaying their titles and descriptions.
   ii. Presence and functionality of the "Ask a Question" button.
   iii. Navigation to the appropriate page when the button is clicked.

4. Features not to be tested

a. **Styling and UI Presentation:** Visual appearance (CSS, layout, colors, fonts) and responsive design aspects are not covered by these tests.

5. Testing Approach

a. **Unit testing:**
   i. **Isolated Component testing:** Individual components, such as the *QuestionForm* and *QuestionsList*, are tested in isolation to verify that their internal logic and rendering behave as expected. For instance, the *QuestionForm* is unit tested for input value changes, form submission, and navigation triggers.
   ii. **Mocking External Dependencies:** Functions such as *fetch* and *useNavigate* are mocked so that each component's functionality can be validated without relying on actual API calls or navigation behavior. This isolation ensures that tests pinpoint the exact behavior of the component itself.

b. **Integration testing:**
   i. **User flow simulation:** Integration tests simulate complete user flows, such as submitting a question in the *QuestionForm* and navigating to a different route. They confirm that the flow—from user input to context update and then to navigation—works correctly in tandem.
   ii. **Cross-Component Communication:** By combining multiple components (e.g., *QuestionForm* with routing and context providers), integration tests verify that the communication between these components behaves as expected. This ensures that the system works as a whole, rather than just isolated parts.

c. **End-to-End testing using Cypress:**
   i. **Posting a Question:** The user navigates to the "Ask a Question" section, fills out the title and description, and submits the question, which is then verified on the posts page.

ii. **Deletion:** Finally, the user deletes the question and confirms that it has been removed from the display.

6. Item Pass/Fail Criteria

a. **QuestionForm Component:**

   i. **Pass Criteria:**

      1. The component renders input fields for the title and description with the correct placeholders.
      2. User input correctly updates the input values.
      3. Submitting the form triggers the addQuestion function.
      4. Navigation is triggered to the /posts page upon successful form submission.
      5. The "Back" button navigates to the previous page when clicked.

   ii. **Fail Criteria:**

      1. The input fields are missing, incorrectly rendered, or placeholders are incorrect.
      2. The user inputs are not reflected in the input field values.
      3. The addQuestion function is not called on submission, or the inputs are not cleared.
      4. Navigation does not occur or occurs with incorrect parameters.
      5. Clicking the "Back" button does not trigger navigation or navigates to an unintended route.

b. **QuestionsList component:**

   i. **Pass Criteria:**

      1. All provided questions (titles and descriptions) are correctly rendered in the list.
      2. The "Ask a Question" button is present on the page.
      3. Clicking the "Ask a Question" button triggers navigation to the /ask route.
      4. Clicking the "Delete" button for a question calls the deleteQuestion function with the correct question ID.

   ii. **Fail criteria:**

      1. Questions are not rendered correctly (missing titles/descriptions).
      2. The "Ask a Question" button is missing or non-functional.
      3. Navigation does not occur or is triggered with an incorrect route.
      4. The delete operation does not call deleteQuestion, or is called with incorrect data.

7. Environmental needs

a. **Development framework:** React.js
b. **Testing frameworks:** Vitest, Cypress

8. Test Deliverables

a. Test cases document (Jest and Cypress test scripts).
b. Test execution report (pass or fail status for each test case).
c. Bug report (if any issues are found).

9. Testing Tasks:

a. Write unit tests to verify toggleFavorite updates localStorage correctly.
b. Implement Cypress tests for UI interactions (adding or removing favorites).
c. Perform manual testing to ensure UI reflects state changes.
d. Log and report bugs for unexpected behavior.

10. Responsibilities

a. **Frontend Developer:** Ensures the UI correctly reflects favorite state changes.
b. **QA Tester:** Runs Jest and Cypress tests and reports bugs.
c. **Team Lead:** Reviews test results and approves fixes.

11. Schedule

a. **Frontend Developer:**
    i. Feature development & coding: 3 days
    ii. Debugging & testing: 2 days
b. **QA Tester:**
    i. Test planning & case creation: 1 day
    ii. Test execution (including regression testing): 3 days
    iii. Final test rounds & defect verification: 1 day
c. **Team Lead:**
    i. Sprint planning & team coordination: 1 day
    ii. Monitoring progress & issue resolution: 2 days
    iii. End-of-week review & reporting: 2 days

12. Risk and Contingencies

a. **Risk:** Flaky Tests / Unstable Test Environment
    i. **Contingency:** Implement retry logic, stabilize test setup using proper mocks and fixtures, and regularly update test suites to handle environmental changes**.**
b. **Risk:** Delayed Feature Delivery Impacting Testing Schedule
    i. **Contingency:** Prioritize testing critical functionality, maintain a backlog of regression tests, and adjust sprint planning to accommodate feature delays.
c. **Risk:** Communication Breakdowns Between Developers, QA, and Team Lead
    i. **Contingency:** Schedule regular check-ins, use collaborative tools for real-time updates, and establish clear documentation to ensure alignment across roles.
d. **Risk:** Unexpected Bugs or Integration Issues
    i. **Contingency:** Reserve time for ad-hoc testing and debugging, set up rapid feedback loops, and maintain an agile approach to quickly address and resolve defects.
e. **Risk: Dependency on External Services**
    i. **Contingency:** Use mocks and stubs for external APIs and services, ensuring tests can run reliably even if external systems are unavailable or slow.

13. Approvals

a. **Project Lead:** Manu Janardana
b. **QA Tester:** Sriramkumar Raja Natarajan
c. **Professor:** Prof. Peter Dillinger