Frontend Developer Exam:d

Objective

For this trial you are going to build a survey/assessment (a set of questions passed in) which then presents a recommendations screen (with results from Algolia and content from contentful) that can be embedded into any app. The goal is for the user to be presented with the assessment, start it, complete it (with error checking etc as described below) and then get a screen with recommendations.

We provide you with API keys and sample data to work with, but it is up to you to create the component as you see fit.

NOTE: While we generally encourage the use of generative AI for coding in our organization, for this test you are not allowed to use generative AI to generate the code (fine to ask quick questions but not to generate code). You will be required to use the upwork auto time tracking so we will be able to see a screenshot of what you are doing every minute or two and will check to be sure you did this work on your own. It is fine if you do not complete all the work in the allotted timeframe, just complete as much as you can of the core requirements.

Also note our goal in this test is not on the aesthetics of the UI, it can be very basic look and feel as long as you are using the required styling approach (as outlined), the UI can be very plain as long as the code is quality (and UI can be changed easily with CSS).

Technically your task is to create an embeddable assessment web component using React and StencilJS (or an equivalent technology) that can be integrated into any web application. Develop a separate parent React application that fetches data from Contentful and utilizes the embeddable component.

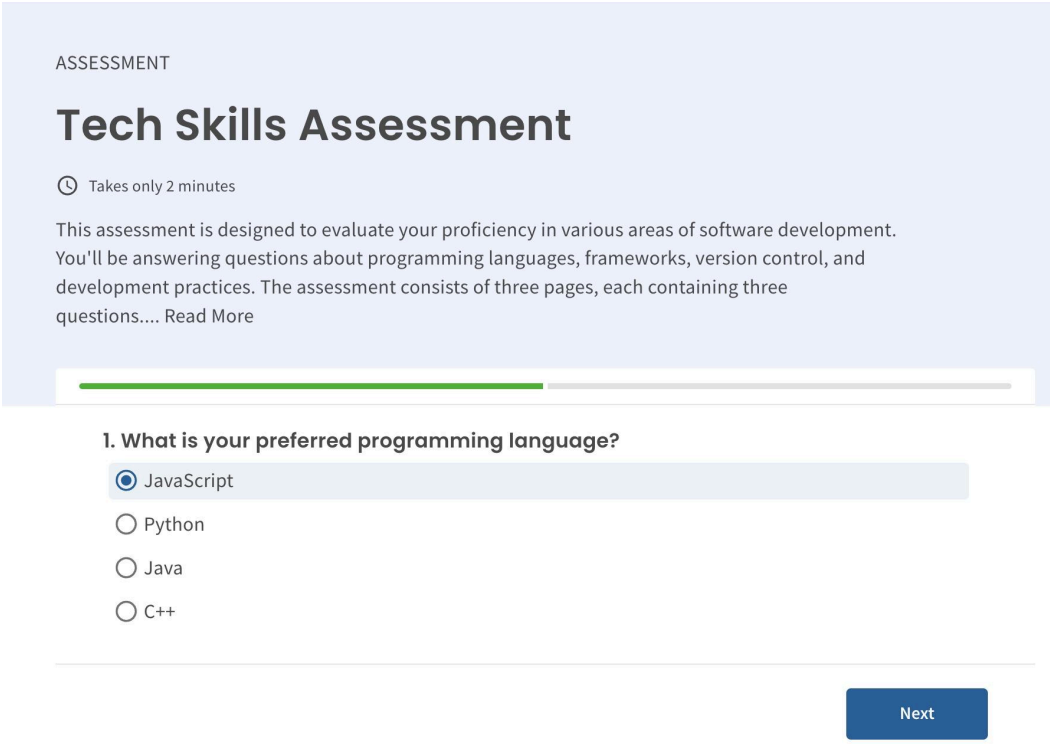
Requirements

1. Technology Stack:
    - Embeddable Component:
        - React
        - StencilJS (or equivalent for creating web components)
        - Algolia Search API integration (returns recommended resources based on assessment results)
    - Parent Application:
        - React
        - Contentful API integration
2. Embeddable Component Features:

    - Accept a set of questions as input via attributes/properties
    - Render questions across multiple pages
    - Support four question types: checkbox, radio group, text, and boolean
    - Implement a progress bar (configurable to show or hide)
    - On the last page, submit answers to Algolia and display recommended resources
    - Expose events for key interactions (e.g., completion, page changes)
3. Parent Application Features:
    - Fetch question data from Contentful
    - Embed the assessment component
    - Pass fetched data to the embeddable assessment component
    - Handle events emitted by the assessment component
4. Component Structure:
    - Create a web component that encapsulates the entire assessment functionality
    - Implement sub-components for different question types
    - Develop a progress bar component
    - Design a results page to display Algolia recommendations
5. State Management:
    - Manage the state of user responses within the web component
    - Track current page and overall progress
6. Styling:
    - Implement responsive design
    - Use shadow DOM for style encapsulation
    - Provide theming options via CSS custom properties
7. Contentful Integration:
    - Use the following Contentful details to fetch the assessment data in your parent React application:
        - Space ID: **h3n75a0xb6vi**
        - Access Token: **3R9BuNun6VNkwPQnoUFe-N_dVPA77YccpKmKGla7D54**
        - Content Type: **assessment**
    - Use Contentful's GraphQL API to fetch the assessment data. The GraphQL endpoint is:`https://graphql.contentful.com/content/v1/spaces/h3n75a0xb6vi`
    - Implement a GraphQL query to fetch all necessary fields for the assessment:
        - name: The name of the assessment
        - slug: A URL-friendly identifier for the assessment
        - intro: Rich text introduction to the assessment
        - questions: JSON data containing the assessment questions
        - resultsIntro: Rich text introduction for the results page
    - You may use any GraphQL client library of your choice (e.g., Apollo Client, urql, or graphql-request).
    - Implement proper error handling for cases where Contentful data might be missing or malformed.

- Optimize your query to fetch all required data in a single request.

8. User Interface:
   - Use the provided image as a reference for the basic layout and styling of the assessment, particularly for radio group questions. You are free to improve upon this design.
   - Implement a "Read More/Less" functionality for the introduction text:
     - Initially, show a truncated version of intro text, limited to 100 characters.
     - If the intro text exceeds 100 characters, add an ellipsis (...) at the end of truncated text, followed by a “Read More” link.
     - When the “Read More” link is clicked, it should expand to show the full text and change to “Read Less”.
     - Clicking “Read Less” should collapse the text back to its truncated state.
   - Add an estimated time to complete the assessment. This should be calculated based on the number of questions (you may assume each question takes an average of 30 seconds to answer).
   - The progress bar should reflect the user's progress through the assessment.
   - Here is an example of our current implementation — it does not have to look the same but if helpful you can use it as a guide: https://cm-qa-testing.netlify.app/assessments
   - (and below is a made up assessment that could be created with this code)



9. Question Handling and Navigation:

- All questions in the assessment are mandatory.
- The "Next" button should always be enabled.
- When the user clicks the "Next" button, the system should check if all questions on the current page are answered.
- If there are any unanswered questions, the user should not proceed to the next page. Instead, display error messages for each unanswered question.
- Error messages should be displayed below the unanswered questions and should be visually distinct (e.g., red text).
- Once the user has answered all questions on a page, clicking "Next" should take them to the next page of the assessment.

10. Error Handling and Validation:
    - Implement client-side validation to check for unanswered questions when the user clicks the "Next" button.
    - Display clear, user-friendly error messages for each unanswered question.
    - Ensure error messages are accessible and compliant with WCAG guidelines.
    - Provide visual cues (such as highlighting or icons) to quickly identify which questions require answers.
    - Implement smooth scrolling to the first unanswered question when the user attempts to proceed with unanswered questions.
    - After displaying error messages, allow users to answer the questions and try proceeding again without needing to re-click the "Next" button.
11. Algolia Integration and Recommendations:
    - After the user completes all questions, use Algolia to fetch relevant resources based on the user's answers.

      Use the following Algolia credentials to initialize the search client:

```
import algoliasearch from 'algoliasearch/lite';

const searchClient = algoliasearch(
  "4WK61QBPDU",
  "a3a8a3edba3b7ba9dad65b2984b91e69"
);


const index =
searchClient.initIndex('algolia-recommendation-data');
```

      Construct a search query based on the user's answers. The answers should be structured as follows:

```
const userAnswers = {
  preferredLanguage: ["JavaScript"],
 familiarFrameworks: ["React", "Vue.js"],
  gitExperience: ["Yes"],
```

```
  remoteWorkExperience: ["Yes"],
  contributeOpenSource: ["Yes"]
  // ... other answers

};
```

Use the following method to create facet filters for the Algolia query:

```
const facetFilters =
Object.values(userAnswers).flat().map(value =>
`relevantTo:${value}`);
```

Perform the Algolia search using these facet filters:

```
index.search('', {

  facetFilters: facetFilters
}).then(({ hits }) => {
  // Process and display the results

});
```

- Display all relevant results returned by Algolia. Each result should be presented as a card containing:
  - Resource title
  - Author
  - Type of resource (e.g., Book, Online Course)
  - A brief description
  - The resource image (using the provided `imageUrl`)User Interface for Recommendations:
- Design and implement a card-style layout for displaying the recommended resources.
- Each card should have a consistent design and contain all the information mentioned above.
- The cards should be responsive and display well on various screen sizes.
- Implement proper image loading techniques, such as lazy loading for images that are not immediately visible on the screen.
- Provide appropriate fallback or placeholder for images that fail to load.
- As a nice-to-have feature, consider implementing a carousel layout for the resource cards. This could improve the user experience, especially on mobile devices or when there are many recommendations.

12. Error Handling and Edge Cases:
    - Implement error handling for cases where the Algolia API request fails.

    - Handle cases where no results are returned from Algolia, providing a user-friendly message or fallback content.
    - Consider implementing a fallback recommendation list in case of API failures.

Tasks

1. Embeddable Component Development
    - Set up the project with React and StencilJS (or equivalent)
    - Develop the core assessment functionality
    - Implement the web component wrapper
    - Create sub-components for question types and progress bar
    - Integrate with Algolia for recommendations
    - Implement event emissions for key interactions
2. Parent Application Development
    - Set up a new React application
    - Implement Contentful data fetching using GraphQL
    - Embed the assessment web component
    - Handle data passing and event listening
3. Styling and Responsiveness
    - Implement responsive design within the web component
    - Set up CSS custom properties for theming
    - Ensure proper style encapsulation
4. Documentation
    - Provide clear documentation on how to use and integrate the web component
    - Document the API, attributes, properties, and events of the web component
    - Include setup instructions for the parent application
5. Nice-to-have Features
    - Testing:
        - Write unit tests for React components
        - Develop integration tests for the web component
        - Create a test harness to verify component in different environments
    - Implement a carousel layout for resource recommendation cards

Additional Guidelines

Version Control and Commit Process:

- Use Git for version control throughout your implementation process.
- Make small, frequent commits that represent logical units of work.
- Write clear, descriptive commit messages that explain the purpose of each change.
- Aim to commit after implementing each significant feature or fixing a bug.
- This practice will help us understand your development process and decision-making.

Evaluation Criteria

1. Component Design (20%)
    - Proper implementation as a reusable web component
    - Clean separation between embeddable component and parent application
    - Effective use of attributes, properties, and events
2. Functionality (20%)
    - Correct implementation of all required features
    - Smooth user experience throughout the assessment
    - Proper handling of data flow between parent app and web component
3. Code Quality (15%)
    - Clean, well-organized, and commented code
    - Proper use of React, StencilJS (or equivalent), and web component best practices
4. Contentful Integration (10%)
    - Proper implementation of GraphQL query to fetch assessment data
    - Efficient data fetching (single request for all required data)
    - Proper error handling and loading states
5. Algolia Integration and Recommendations (10%)
    - Correct implementation of Algolia search using provided credentials
    - Effective construction of search queries based on user answers
    - Clear and user-friendly display of all relevant recommended resources in a card-style layout
    - Appropriate error handling and fallback mechanisms for Algolia integration and image loading
    - Efficient implementation of image loading techniques
    - (Bonus) Implementation of a carousel layout for resource cards
6. UI/UX Design (10%)
    - Aesthetic and functional card design for resource recommendations
    - Responsive layout that works well on various device sizes
    - Intuitive and smooth user interaction with the recommendation cards
    - Effective use of visual hierarchy and typography
    - (Bonus) Creative and user-friendly implementation of the carousel feature
7. Error Handling and User Guidance (10%)
    - Effective implementation of error messages for unanswered questions
    - Clear visual indication of which questions require answers
    - Smooth and intuitive error resolution process for users
    - Appropriate balance between allowing navigation attempts and providing clear feedback on required fields
8. Version Control and Development Process (3%)
    - Appropriate use of Git with small, frequent commits
    - Clear and descriptive commit messages
    - Logical progression of development visible through commit history
9. Testing (2% Bonus)

    - Comprehensive unit and integration tests
    - Effective test harness for the web component

Submission Guidelines

1. Provide two GitHub repositories: a. The embeddable assessment web component b. The parent React application demonstrating component usage
    - Nice-to-have: If you choose to use a monorepo structure, consider using Nx or a similar tool.
2. Include README.md files in both repositories with:
    - Setup instructions
    - Usage examples
    - API documentation for the web component
    - Any assumptions or design decisions made
3. Deploy working demos of: a. The test harness for the web component (if implemented) b. The parent application with the component embedded
4. Ensure your commit history reflects your development process:
    - Small, frequent commits
    - Clear, descriptive commit messages
    - Logical progression of feature implementation and bug fixes
5. Submit your repository links and demo URLs to the hiring team.

If you have any questions about the exam or requirements, please email developer@crediblemind.com.

Good luck with your implementation!