

Prompt used:

“what updates do I need to add to make my questionnaire work”

Copilot:

“what updates do I need to add to make my questionnaire work”

Copilot:

Copilot:

“To make your questionnaire work, you need to ensure the following updates are implemented:

1. Dynamic Question Rendering

“To make your questionnaire work, you need to ensure the following updates are implemented:

1. Dynamic Question Rendering

1. Dynamic Question Rendering

You need to dynamically load questions from the questions.js file into the respective sections (employeeAssessment and organizationAssessment) in your HTML. Add a function to render the questions dynamically.”

```
import { employeeQuestions, organizationQuestions } from
'./questions.js';

function renderQuestions(assessmentType) {
  const containerId = `${assessmentType}Assessment`;
  const container = document.getElementById(containerId);
  const questions = assessmentType === 'employee' ?
employeeQuestions.categories[0].questions :
organizationQuestions.questions[0].questions;

  container.innerHTML = ''; // Clear existing questions

  questions.forEach((category, categoryIndex) => {
    category.questions.forEach((question, questionIndex) => {
      const questionElement = document.createElement('div');
      questionElement.classList.add('question', 'hidden');
      questionElement.setAttribute('data-question', categoryIndex
* 100 + questionIndex + 1);

      questionElement.innerHTML = `
        <p class="text-xl mb-6">${question.question}</p>
        <div class="space-y-4">
          ${question.answers.map((answer, index) => `
            <div class="flex items-center">
```

```

        <input type="radio"
id="\${assessmentType}Option\${categoryIndex * 100 + questionIndex +
1}_\${index}" name="\${assessmentType}Answer\${categoryIndex * 100 +
questionIndex + 1}" value="\${answer.score}" class="h-4 w-4 text-accent
focus:ring-accent border-gray-300">

        <label
for="\${assessmentType}Option\${categoryIndex * 100 + questionIndex +
1}_\${index}" class="ml-3 block text-textlight">\${answer.option ||
answer.text}</label>

    </div>

    `).join('')
</div>

    <p class="error-message hidden text-red-500
mt-2">Please select an answer.</p>

    `;

    container.appendChild(questionElement);

    });

});

// Show the first question
container.querySelector('.question').classList.remove('hidden');
}

```

2. Call renderQuestions on Start

Update the startEmployee and startOrganization button event listeners to call the renderQuestions function.

```

document.getElementById('startEmployee').addEventListener('click',
function() {

    renderQuestions('employee');

    showSection('employeeAssessment');

    resetAssessment('employee');

});

```

```
document.getElementById('startOrganization').addEventListener('click',
function() {

    renderQuestions('organization');

    showSection('organizationAssessment');

    resetAssessment('organization');

});
```

3. Track Responses

Add logic to track user responses for each question. Create an object to store responses.

```
let employeeResponses = {};
let organizationResponses = {};

function saveResponse(assessmentType, questionNumber) {
    const containerId = `${assessmentType}Assessment`;
    const question = document.querySelector(`#${containerId}
.question[data-question="${questionNumber}"]`);
    const selectedInput =
question.querySelector('input[type="radio"]:checked');

    if (selectedInput) {
        const responses = assessmentType === 'employee' ?
employeeResponses : organizationResponses;
        responses[questionNumber] = parseInt(selectedInput.value);
    }
}
```

“Update the **Next** button event listeners to save responses before moving to the next question.”

```
document.getElementById('employeeNext').addEventListener('click',
function() {
    saveResponse('employee', currentEmpQuestion);
    if (!validateQuestion('employee', currentEmpQuestion)) return;

    // Existing logic...
});

document.getElementById('orgNext').addEventListener('click', function()
{
    saveResponse('organization', currentOrgQuestion);
    if (!validateQuestion('organization', currentOrgQuestion)) return;
```

```
    // Existing logic...
  });
```

“4. Submit Responses

Handle the submission of responses when the user reaches the last question.

```
“
document.getElementById('employeeNext').addEventListener('click',
function() {
    if (currentEmpQuestion === totalEmpQuestions) {
        saveResponse('employee', currentEmpQuestion);
        const employeeScore =
calculateEmployeeScore(employeeResponses);
        console.log('Employee Responses:', employeeResponses);
        console.log('Employee Score:', employeeScore);
        showSection('employeeFeedback');
    }
});

document.getElementById('orgNext').addEventListener('click', function()
{
    if (currentOrgQuestion === totalOrgQuestions) {
        saveResponse('organization', currentOrgQuestion);
        const organizationScore =
calculateOrganizationScore(organizationResponses);
        console.log('Organization Responses:', organizationResponses);
        console.log('Organization Score:', organizationScore);
        showSection('organizationFeedback');
    }
});
```

“5. Update Progress Bar

Ensure the progress bar updates dynamically based on the number of questions.

```
“
function updateEmployeeProgress() {
    const progress = (currentEmpQuestion /
employeeQuestions.categories[0].questions.length) * 100;
    document.getElementById('employeeProgressBar').style.width =
`${progress}%`;
    document.getElementById('employeeProgressText').textContent =
`Question ${currentEmpQuestion} of
${employeeQuestions.categories[0].questions.length}`;
```

```

}

function updateOrgProgress() {
  const progress = (currentOrgQuestion /
organizationQuestions.questions[0].questions.length) * 100;
  document.getElementById('orgProgressBar').style.width =
`${progress}%`;
  document.getElementById('orgProgressText').textContent = `Question
${currentOrgQuestion} of
${organizationQuestions.questions[0].questions.length}`;
}

```

6. Error Handling

Ensure error messages are displayed if no answer is selected.

```

function validateQuestion(assessmentType, questionNumber) {
  const containerId = `${assessmentType}Assessment`;
  const question = document.querySelector(`#${containerId}
.question[data-question="${questionNumber}"]`);
  const inputs = question.querySelectorAll('input[type="radio"]');
  const error = question.querySelector('.error-message');

  const answered = Array.from(inputs).some(input => input.checked);

  if (!answered) {
    error.classList.remove('hidden');
    question.classList.add('shake');
    setTimeout(() => question.classList.remove('shake'), 500);
    return false;
  }

  error.classList.add('hidden');
  return true;
}

```

7. Feedback Display

Ensure the feedback sections (employeeFeedback and organizationFeedback) display the results dynamically based on the scores.

These updates will make your questionnaire functional and interactive. Let me know if you need further assistance!

“

Prompt used:

“

how should the [question.js](#) be structured for dynamic question insertion?

”

Copilot:

“

You can update your questions.js file to match the following structure:

“

```
export const employeeQuestions = {
  categories: [
    {
      "category": "Password & Access Management",
      "questions": [
        // Keep your existing question objects here
      ]
    },
    // Add other categories here
  ]
};
```

```
export const organizationQuestions = {
  questions: [
    {
      "category": "Identity & Access Management",
      "questions": [
        // Keep your existing question objects here
      ]
    },
    // Add other categories here
  ]
};
```

“

This update is a good way to dynamically render the questions and handle navigation between them. Let me know if you need further assistance!

”