```
// questionnaire.js - Handles the logic for the AI-powered business context questionnaire.

export default class QuestionnaireManager {
    constructor(niceClassManager) {
        this.niceClassManager = niceClassManager;
        this.questionnaireModal = document.getElementById('questionnaireModal');
        this.questionContainer = document.getElementById('questionContainer');
        this.prevBtn = document.getElementById('prevBtn');
        this.nextBtn = document.getElementById('nextBtn');
        this.analyzeBtn = document.getElementById('analyzeBtn');

        // This will be populated with questions from the backend
        this.questions = [];
        this.currentQuestionIndex = 0;
        this.questionnaireResponses = {};
        this.isInitialized = false;

        // Bind event handlers to the class instance
        this.prevBtn?.addEventListener('click', () => this.previousQuestion());
        this.nextBtn?.addEventListener('click', () => this.nextQuestion());
        this.analyzeBtn?.addEventListener('click', () => this.submitQuestionnaire());

        // Fix for the modal close button
        const closeBtn = document.querySelector('#questionnaireModal .modal-close');
        closeBtn?.addEventListener('click', () => this.close());
    }

    async initialize() {
        if (this.isInitialized) return;

        console.log('QuestionnaireManager initialized.');
        await this.fetchQuestions();
        this.isInitialized = true;
    }

    async fetchQuestions() {
        try {
            const response = await fetch('/questionnaire/questions');
            if (!response.ok) {
                throw new Error('Failed to fetch questions from the backend.');
            }
            this.questions = await response.json();
            // Filter out any invalid question objects just in case
            this.questions = this.questions.filter(q => q && q.id);
            console.log('Fetched questions:', this.questions);
            this.renderQuestion(this.currentQuestionIndex);
        } catch (error) {
            console.error('Error fetching questions:', error);
            this.showError('Failed to load questionnaire questions. Please try again later.');
            this.close();
        }
    }
```

```
open() {
    if (this.questionnaireModal) {
        this.questionnaireModal.classList.remove('hidden');
    }
    this.currentQuestionIndex = 0;
    this.renderQuestion(this.currentQuestionIndex);
}

close() {
    if (this.questionnaireModal) {
        this.questionnaireModal.classList.add('hidden');
    }
}

renderQuestion(index) {
    if (!this.questionContainer || index < 0 || index >= this.questions.length) {
        return;
    }

    const questionData = this.questions[index];
    const html = this.generateQuestionHtml(questionData);
    this.questionContainer.innerHTML = html;
    this.restoreCurrentAnswer(questionData);

    this._attachInputEventListeners();

    this.updateNavigation();
}

generateQuestionHtml(questionData) {
    let html = `
        <div class="question active">
            <h4>${questionData.title}</h4>
            ${questionData.followUp ? `<p class="follow-up-text">${questionData.followUp}<
            <div class="sub-question-container">
    `;

    if (questionData.type === 'multi') {
        html += questionData.parts.map(part => `
            <div class="sub-question">
                <label for="${part.id}">${part.label}</label>
                ${this.generateInputHtml(part)}
            </div>
        `).join('');
    } else {
        html += this.generateInputHtml(questionData);
    }

    html += `
            </div>
        </div>
    `;
    return html;
```

```
}

generateInputHtml(inputData) {
    switch (inputData.type) {
        case 'text':
            return `<input type="text" id="${inputData.id}" name="${inputData.id}" placeho
        case 'textarea':
            return `<textarea id="${inputData.id}" name="${inputData.id}" rows="4" placeho
        case 'radio':
            return `
                <div class="radio-group">
                    ${inputData.options.map(option => `
                        <label class="radio-item">
                            <input type="radio" name="${inputData.id}" value="${option}">
                            <span>${option}</span>
                        </label>
                    `).join('')}
                </div>
            `;
        case 'checkbox':
            return `
                <div class="checkbox-list">
                    ${inputData.options.map(option => `
                        <label class="checkbox-item">
                            <input type="checkbox" name="${inputData.id}" value="${option}
                            <span>${option}</span>
                        </label>
                    `).join('')}
                </div>
            `;
        default:
            return '';
    }
}

_handleEnterKey(event) {
    // Prevent the default form submission behavior when Enter is pressed
    if (event.key === 'Enter') {
        event.preventDefault();

        // If the "Next" button is visible, click it to proceed
        if (this.nextBtn && !this.nextBtn.classList.contains('hidden')) {
            this.nextBtn.click();
        }
    }
}

_attachInputEventListeners() {
    // Find all text inputs and textareas in the current question
    const inputs = this.questionContainer.querySelectorAll('input[type="text"], textarea')

    // Add the keydown event listener to each input
    inputs.forEach(input => {
```

```
        // Bind the event listener to the class instance to maintain `this` context
        input.addEventListener('keydown', this._handleEnterKey.bind(this));
    });
}

storeCurrentAnswer() {
    const currentQuestion = this.questions[this.currentQuestionIndex];
    if (!currentQuestion) return;

    if (currentQuestion.type === 'multi') {
        currentQuestion.parts.forEach(part => {
            const inputElement = this.questionContainer.querySelector(`#${part.id}`);
            if (inputElement) {
                if (part.type === 'radio') {
                    const selected = this.questionContainer.querySelector(`input[name="${p
                    this.questionnaireResponses[part.id] = selected ? selected.value : '';
                } else if (part.type === 'checkbox') {
                     const selectedOptions = Array.from(this.questionContainer.querySelect
                     this.questionnaireResponses[part.id] = selectedOptions;
                } else {
                    this.questionnaireResponses[part.id] = inputElement.value;
                }
            }
        });
    } else if (currentQuestion.type === 'checkbox') {
        const selectedOptions = Array.from(this.questionContainer.querySelectorAll(`input[
        this.questionnaireResponses[currentQuestion.id] = selectedOptions;
    } else {
        const inputElement = this.questionContainer.querySelector(`#${currentQuestion.id}`
        if (inputElement) {
            this.questionnaireResponses[currentQuestion.id] = inputElement.value;
        }
    }
}

restoreCurrentAnswer(questionData) {
    if (questionData.type === 'multi') {
        questionData.parts.forEach(part => {
            const value = this.questionnaireResponses[part.id];
            if (value) {
                if (part.type === 'radio') {
                    const radio = this.questionContainer.querySelector(`input[name="${part
                    if (radio) radio.checked = true;
                } else if (part.type === 'checkbox') {
                    const values = this.questionnaireResponses[part.id] || [];
                    values.forEach(val => {
                        const checkbox = this.questionContainer.querySelector(`input[name=
                        if (checkbox) checkbox.checked = true;
                    });
                } else {
                    const inputElement = this.questionContainer.querySelector(`#${part.id}
                    if (inputElement) inputElement.value = value;
                }
```

```
            }
        });
    } else if (questionData.type === 'checkbox') {
        const values = this.questionnaireResponses[questionData.id] || [];
        values.forEach(value => {
            const checkbox = this.questionContainer.querySelector(`input[name="${questionD
            if (checkbox) checkbox.checked = true;
        });
    } else {
        const value = this.questionnaireResponses[questionData.id];
        const inputElement = this.questionContainer.querySelector(`#${questionData.id}`);
        if (inputElement && value) {
            inputElement.value = value;
        }
    }
}

previousQuestion() {
    if (this.currentQuestionIndex > 0) {
        this.storeCurrentAnswer();
        this.currentQuestionIndex--;
        this.renderQuestion(this.currentQuestionIndex);
    }
}

nextQuestion() {
    if (this.currentQuestionIndex < this.questions.length - 1) {
        this.storeCurrentAnswer();
        this.currentQuestionIndex++;
        this.renderQuestion(this.currentQuestionIndex);
    }
}

updateNavigation() {
    if (this.prevBtn) {
        this.prevBtn.classList.toggle('hidden', this.currentQuestionIndex === 0);
    }
    if (this.nextBtn) {
        this.nextBtn.classList.toggle('hidden', this.currentQuestionIndex === this.questio
    }
    if (this.analyzeBtn) {
        this.analyzeBtn.classList.toggle('hidden', this.currentQuestionIndex < this.questi
    }
}

showError(message) {
    alert(message);
    console.error(message);
}

showSuccess(message) {
    alert(message);
    console.log(message);
```

```
}

async submitQuestionnaire() {
    console.log('Submitting questionnaire for AI analysis');
    this.storeCurrentAnswer();

    const analyzeBtn = document.getElementById('analyzeBtn');
    if (!analyzeBtn) return;

    const originalText = analyzeBtn.textContent;

    try {
        analyzeBtn.disabled = true;
        analyzeBtn.textContent = 'Analyzing...';

        const authToken = localStorage.getItem('authToken');
        if (!authToken) {
            throw new Error('Please log in to use AI analysis');
        }

        const requestPayload = { responses: this.questionnaireResponses };

        const response = await fetch('/questionnaire/analyze', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${authToken}`
            },
            body: JSON.stringify(requestPayload)
        });

        if (!response.ok) {
            const errorData = await response.json();
            throw new Error(errorData.detail || 'AI analysis failed');
        }

        const result = await response.json();

        if (result.success && result.recommendations) {
            this.applyAIRecommendations(result.recommendations);
            this.close();
            this.showSuccess('AI analysis complete! Search form auto-filled.');
        } else {
            throw new Error('Analysis failed without recommendations');
        }
    } catch (error) {
        this.showError(`AI analysis failed: ${error.message}. Please fill the form manuall
    } finally {
        analyzeBtn.disabled = false;
        analyzeBtn.textContent = originalText;
    }
}
```

```
    applyAIRecommendations(recommendations) {
        console.log('Applying AI recommendations to search form');

        if (recommendations.trademark) {
            const trademarkInput = document.getElementById('trademark');
            if (trademarkInput) {
                trademarkInput.value = recommendations.trademark;
            }
        }

        if (recommendations.suggested_classes) {
            this.niceClassManager.setSelectedClasses(recommendations.suggested_classes);
        }

        if (recommendations.thresholds) {
            const thresholdMappings = {
                'phonetic': 'phoneticThreshold',
                'visual': 'visualThreshold',
                'conceptual': 'conceptualThreshold'
            };
            Object.entries(recommendations.thresholds).forEach(([type, value]) => {
                const slider = document.getElementById(thresholdMappings[type]);
                const display = document.getElementById(type + 'Value');
                if (slider && display) {
                    const percentage = Math.round(value * 100);
                    slider.value = percentage;
                    display.textContent = `${percentage}%`;
                }
            });
        }
    }
}
```