```
// results.js - Results display manager
export default class ResultsManager {
    constructor(authManager, appController) {
        this.authManager = authManager;
        this.appController = appController;
        this.currentResults = null;
        this.currentSearchId = null;
        this.currentSearchMode = 'basic';
    }

    displayResults(results, searchMode) {
        this.currentResults = results;
        this.currentSearchMode = searchMode;
        // ANNOTATION: The unique search_id from the backend is now stored.
        this.currentSearchId = results.search_id;

        const resultsContent = document.getElementById('resultsContent');
        if (!results || !results.matches || results.matches.length === 0) {
            this.showEmptyState();
            this.updateDownloadSection(false); // Hide downloads if no results
            return;
        }

        resultsContent.innerHTML = this.generateResultsHTML(results);
        this._addResultEventListeners();

        // ANNOTATION: Make the download section visible now that a search is complete.
        this.updateDownloadSection(true);
    }

    _addResultEventListeners() {
        const commonLawBtn = document.getElementById('start-common-law-search');
        const checkboxes = document.querySelectorAll('.result-checkbox');
        const selectAllCheckbox = document.getElementById('selectAllResults');

        const updateButtonState = () => {
            if (commonLawBtn) {
                const anyChecked = Array.from(checkboxes).some(cb => cb.checked);
                commonLawBtn.disabled = !anyChecked;
            }
        };

        checkboxes.forEach(checkbox => {
            checkbox.addEventListener('change', updateButtonState);
        });

        if (selectAllCheckbox) {
            selectAllCheckbox.addEventListener('change', (event) => {
                checkboxes.forEach(cb => cb.checked = event.target.checked);
                updateButtonState();
            });
        }
```

```
    if (commonLawBtn) {
        commonLawBtn.addEventListener('click', () => {
            const selectedRows = [];
            document.querySelectorAll('.result-checkbox:checked').forEach(cb => {
                const row = cb.closest('tr');
                selectedRows.push({
                    mark_identification: row.dataset.mark,
                    owner: row.dataset.owner,
                    serial_number: row.dataset.serial
                });
            });

            if (this.appController && selectedRows.length > 0) {
                this.appController.performSelectionBasedInvestigation(selectedRows);
            }
        });
    }

    // ANNOTATION: Attaches click event listeners to all the download buttons.
    document.getElementById('downloadCsvBtn')?.addEventListener('click', () => this.downlo
    document.getElementById('downloadJsonBtn')?.addEventListener('click', () => this.downl
    document.getElementById('downloadTxtBtn')?.addEventListener('click', () => this.downlo
    document.getElementById('downloadVariationsBtn')?.addEventListener('click', () => this
    document.getElementById('downloadReportMdBtn')?.addEventListener('click', () => this.d
    document.getElementById('downloadReportDocxBtn')?.addEventListener('click', () => this

    updateButtonState();
}

generateResultsHTML(results) {
    let html = `
        <div class="results-summary">
            <h4>Search Results for "${results.query_trademark}"</h4>
            <div class="summary-stats">
                <span class="stat">${results.total_matches} matches</span>
                <span class="stat">${results.execution_time_ms.toFixed(1)}ms</span>
                <span class="stat">${this.currentSearchMode.toUpperCase()} mode</span>
            </div>
        </div>
        <div class="results-table-container">
            <div class="table-controls">
                <label class="select-all-container">
                    <input type="checkbox" id="selectAllResults"> Select All
                </label>
                <div class="common-law-control">
                    <button id="start-common-law-search" class="btn btn-secondary" disable
                        Investigate Selected
                    </button>
                    <div class="info-container">
                        <span class="info-trigger">[What's this?]</span>
                        <div class="info-popup">
                            Select rows to perform a deep-dive Common Law investigation on
                        </div>
```

```
                    </div>
                </div>
            </div>
            <table class="results-table">
                <thead>
                    <tr>
                        <th>Select</th><th>Trademark</th><th>Serial #</th><th>Owner</th><t
                    </tr>
                </thead>
                <tbody>
        `;

        results.matches.forEach(match => {
            const classesText = Array.isArray(match.nice_classes) ? match.nice_classes.join(',
            const riskColor = this.getRiskColor(match.risk_level);
            html += `
                <tr class="result-row" data-mark="${this.escapeHtml(match.mark_identification)
                    <td>
                        <label class="checkbox-container"><input type="checkbox" class="result
                    </td>
                    <td>${this.escapeHtml(match.mark_identification)}</td>
                    <td>${match.serial_number}</td>
                    <td>${this.escapeHtml(match.owner || 'N/A')}</td>
                    <td>${match.status_code || 'N/A'}</td>
                    <td>${classesText}</td>
                    <td>
                        <span class="risk-badge" style="background-color: ${riskColor}">
                            ${match.risk_level.toUpperCase()}
                        </span>
                    </td>
                </tr>
            `;
        });

        html += `</tbody></table></div>`;
        return html;
    }

// new shiot - trying to fix color error
    getRiskColor(riskLevel) {
        const colors = {
            'high': '#dc3545',
            'very_high': '#721c24', // Added for consistency with potential backend data
            'medium': '#ffc107',
            'low': '#28a745',
            'unknown': '#6c757d'
        };
        // Use optional chaining and a fallback for safety
        return colors[riskLevel?.toLowerCase()] || colors.unknown;
    }


    async downloadResults(format) {
```

```
if (!this.currentSearchId) {
    alert('Cannot download. A unique search ID was not found.');
    return;
}

// ANNOTATION: This function now uses the fetch API to include authentication.
const downloadUrl = `/download/${this.currentSearchId}?format=${format}&search_mode=${

// Simple loading indicator - you can make this more sophisticated later
const originalCursor = document.body.style.cursor;
document.body.style.cursor = 'wait';

try {
    // ANNOTATION: The authManager provides the necessary 'Authorization' header.
    const response = await fetch(downloadUrl, {
        method: 'GET',
        headers: this.authManager.getAuthHeaders()
    });

    if (!response.ok) {
        // Try to parse error from server, otherwise use status text
        let errorDetail = response.statusText;
        try {
            const errorJson = await response.json();
            errorDetail = errorJson.message || errorJson.detail || errorDetail;
        } catch (e) {
            // Ignore if response is not JSON
        }
        throw new Error(`Download failed: ${errorDetail}`);
    }

    // ANNOTATION: The file data is received as a 'blob'.
    const blob = await response.blob();

    // ANNOTATION: Extract the filename from the 'Content-Disposition' header sent by
    const disposition = response.headers.get('Content-Disposition');
    let filename = `trademark_search_results.${format}`; // A fallback filename
    if (disposition && disposition.indexOf('attachment') !== -1) {
        const matches = /filename[^;=\n]*=(['"]).*?\2|[^;\n]*)/.exec(disposition);
        if (matches != null && matches[1]) {
            filename = matches[1].replace(/['"]/g, '');
        }
    }

    // ANNOTATION: Create a temporary URL for the blob and a hidden link to trigger th
    const url = window.URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.style.display = 'none';
    a.href = url;
    a.download = filename;

    document.body.appendChild(a);
    a.click();
```

```
            // ANNOTATION: Clean up by removing the temporary URL and link.
            window.URL.revokeObjectURL(url);
            a.remove();

        } catch (error) {
            console.error('Download error:', error);
            alert('Could not download file: ' + error.message);
        } finally {
            // Restore the cursor
            document.body.style.cursor = originalCursor;
        }
    }

    updateDownloadSection(visible) {
        const downloadSection = document.getElementById('downloadSection');
        if (downloadSection) {
            downloadSection.classList.toggle('hidden', !visible);
        }
    }

    showEmptyState() {
        const resultsContent = document.getElementById('resultsContent');
        if(resultsContent) {
            resultsContent.innerHTML = `<div class="empty-state">
                <h4>No Matches Found</h4>
                <p>Your search did not return any potential conflicts. Try adjusting your term
            </div>`;
        }
    }

    escapeHtml(text) {
        if (!text) return '';
        const div = document.createElement('div');
        div.textContent = text;
        return div.innerHTML;
    }
}
```