// Global variables

let currentLayer = 1;

let isProcessing = false;

let uploadedFile = null;

let currentFileId = null;

let processingStartTime = null;

let stepTimers = {};

let stepStartTimes = {};

let fileUploaded = false;

// API base URL

const API\_BASE = 'http://localhost:8000/api';

// File upload handling

async function handleFileUpload(event) {

console.log('handleFileUpload called');

const file = event.target.files[0];

if (!file) return;

if (!file.name.toLowerCase().endsWith('.csv')) {

alert('Please upload a CSV file.');

return;

}

try {

const formData = new FormData();

formData.append('file', file);

const response = await fetch(`${API\_BASE}/upload`, {

method: 'POST',

body: formData

});

if (!response.ok) {

throw new Error('Upload failed');

}

const result = await response.json();

currentFileId = result.file\_id;

uploadedFile = file;

updateUploadUI(file);

updateSidebarStats(file);

fileUploaded = true;

hideLayerSelection();

hideFileUploadSection();

showSidebarAfterUpload();

showProcessingPipelineAfterUpload();

updateStepStatus('step-upload', 'completed');

checkAndShowProcessingPipeline();

} catch (error) {

console.error('Upload error:', error);

alert('File upload failed. Please try again.');

}

}

function updateUploadUI(file) {

const uploadArea = document.getElementById('file-upload-area');

uploadArea.classList.add('uploaded');

uploadArea.innerHTML = `

<div style="display: flex; align-items: center; justify-content: center; gap: 10px;">

<div style="width: 20px; height: 20px; background: #10b981; border-radius: 50%; display: flex; align-items: center; justify-content: center; color: white; font-size: 12px;">✓</div>

<h3 style="font-size: 20px; font-weight: 700; margin: 0; color: var(--text-primary);">File Uploaded Successfully!</h3>

</div>

<p style="color: var(--text-secondary); margin-top: 8px; font-size: 14px;">${file.name}</p>

<input type="file" id="csvFile" accept=".csv" style="display: none;" onchange="handleFileUpload(event)">

`;

}

function updateSidebarStats(file) {

document.getElementById('file-size').textContent = `${(file.size / 1024 / 1024).toFixed(1)}MB`;

}

// Drag and drop functionality

function setupDragDrop() {

const uploadArea = document.getElementById('file-upload-area');

uploadArea.addEventListener('dragover', (e) => {

e.preventDefault();

uploadArea.classList.add('dragover');

});

uploadArea.addEventListener('dragleave', (e) => {

e.preventDefault();

uploadArea.classList.remove('dragover');

});

uploadArea.addEventListener('drop', (e) => {

e.preventDefault();

uploadArea.classList.remove('dragover');

const files = e.dataTransfer.files;

if (files.length > 0) {

const file = files[0];

if (file.name.toLowerCase().endsWith('.csv')) {

document.getElementById('csvFile').files = files;

handleFileUpload({ target: { files: [file] } });

} else {

alert('Please upload a CSV file.');

}

}

});

}

// UI Management Functions

function hideLayerSelection() {

const layerSelection = document.querySelector('.layer-selection');

if (layerSelection) {

layerSelection.style.display = 'none';

}

}

function hideFileUploadSection() {

const fileUploadSection = document.querySelector('.file-upload-section');

if (fileUploadSection) {

fileUploadSection.style.display = 'none';

}

}

function showSidebarAfterUpload() {

const sidebar = document.querySelector('.sidebar');

if (sidebar) {

sidebar.classList.add('show');

}

}

function showProcessingPipelineAfterUpload() {

const pipelineSection = document.getElementById('processing-pipeline-section');

if (pipelineSection) {

pipelineSection.classList.add('show');

}

}

function checkAndShowProcessingPipeline() {

if (fileUploaded) {

const sidebar = document.querySelector('.sidebar');

if (sidebar) {

sidebar.classList.add('show');

}

const pipelineSection = document.getElementById('processing-pipeline-section');

if (pipelineSection) {

pipelineSection.classList.add('show');

}

}

}

// Layer selection

function selectLayer(layer) {

console.log('selectLayer called with layer:', layer);

currentLayer = layer;

// Update layer cards

document.querySelectorAll('.layer-card').forEach((card, index) => {

card.classList.toggle('active', index + 1 === layer);

});

// Update content sections

document.querySelectorAll('.content-section').forEach((section, index) => {

section.classList.toggle('active', index + 1 === layer);

});

checkAndShowProcessingPipeline();

}

// Step status management

function updateStepStatus(stepId, status) {

const step = document.getElementById(stepId);

const statusElement = step.querySelector('.step-status');

const timingElement = document.getElementById(`timing-${stepId.replace('step-', '')}`);

step.classList.remove('active', 'completed', 'error');

switch (status) {

case 'active':

step.classList.add('active');

statusElement.textContent = '';

startStepTimer(stepId);

break;

case 'completed':

step.classList.add('completed');

statusElement.textContent = '';

stopStepTimer(stepId);

break;

case 'error':

step.classList.add('error');

statusElement.textContent = '';

stopStepTimer(stepId, true);

break;

default:

statusElement.textContent = '';

if (timingElement) timingElement.textContent = '';

}

}

// Timer management

function startStepTimer(stepId) {

const stepName = stepId.replace('step-', '');

const timingElement = document.getElementById(`timing-${stepName}`);

stepStartTimes[stepId] = Date.now();

if (timingElement) timingElement.textContent = 'Executing...';

if (stepTimers[stepId]) {

clearInterval(stepTimers[stepId]);

}

stepTimers[stepId] = setInterval(() => {

const elapsed = Math.floor((Date.now() - stepStartTimes[stepId]) / 1000);

if (timingElement) timingElement.textContent = `Executing... ${elapsed}s`;

}, 1000);

}

function stopStepTimer(stepId, isError = false) {

const stepName = stepId.replace('step-', '');

const timingElement = document.getElementById(`timing-${stepName}`);

if (stepTimers[stepId]) {

clearInterval(stepTimers[stepId]);

delete stepTimers[stepId];

}

if (stepStartTimes[stepId]) {

const elapsed = Math.floor((Date.now() - stepStartTimes[stepId]) / 1000);

if (timingElement) {

timingElement.textContent = isError ? `Failed in ${elapsed}s` : `Completed in ${elapsed}s`;

}

delete stepStartTimes[stepId];

}

}

function resetAllTimers() {

Object.keys(stepTimers).forEach(stepId => {

clearInterval(stepTimers[stepId]);

delete stepTimers[stepId];

});

stepStartTimes = {};

const stepNames = ['upload', 'analyze', 'preprocess', 'chunking', 'embedding', 'storage', 'retrieval'];

stepNames.forEach(stepName => {

const timingElement = document.getElementById(`timing-${stepName}`);

if (timingElement) {

timingElement.textContent = '';

}

});

}

// Processing functions

async function startProcessing() {

if (!uploadedFile || !currentFileId) {

alert('Please upload a CSV file first!');

return;

}

if (isProcessing) {

alert('Processing is already in progress!');

return;

}

isProcessing = true;

processingStartTime = Date.now();

resetAllTimers();

const processBtn = document.getElementById('process-btn');

processBtn.disabled = true;

processBtn.textContent = 'Processing...';

try {

const config = gatherConfiguration();

const mode = getModeString(currentLayer);

const response = await fetch(`${API\_BASE}/process`, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({

file\_id: currentFileId,

mode: mode,

preprocessing: config.preprocessing,

chunking: config.chunking,

embedding: config.embedding,

storage: config.storage

})

});

if (!response.ok) throw new Error('Processing request failed');

await pollProcessingStatus();

} catch (error) {

console.error('Processing error:', error);

alert('An error occurred during processing. Please try again.');

const activeStep = document.querySelector('.process-step.active');

if (activeStep) updateStepStatus(activeStep.id, 'error');

isProcessing = false;

const processBtn = document.getElementById('process-btn');

processBtn.disabled = false;

processBtn.textContent = 'Start Processing';

}

}

async function pollProcessingStatus() {

const progressBar = document.getElementById('overall-progress');

const progressText = document.getElementById('progress-text');

const pollInterval = setInterval(async () => {

try {

const response = await fetch(`${API\_BASE}/status/${currentFileId}`);

if (!response.ok) throw new Error('Status check failed');

const status = await response.json();

updateProgressFromStatus(status, progressBar, progressText);

if (status.status === 'completed') {

clearInterval(pollInterval);

await processingCompleted(status);

} else if (status.status === 'error') {

clearInterval(pollInterval);

processingFailed(status.error);

}

} catch (error) {

console.error('Status polling error:', error);

}

}, 1000);

}

function updateProgressFromStatus(status, progressBar, progressText) {

const stepProgress = {

'preprocessing': 25,

'chunking': 50,

'embedding': 75,

'storage': 100,

'completed': 100

};

const currentStep = status.current\_step;

if (progressText) progressText.textContent = `Processing: ${currentStep.charAt(0).toUpperCase() + currentStep.slice(1)}`;

if (stepProgress[currentStep] && progressBar) {

progressBar.style.width = stepProgress[currentStep] + '%';

const stepMap = {

'preprocessing': 'step-preprocess',

'chunking': 'step-chunking',

'embedding': 'step-embedding',

'storage': 'step-storage'

};

Object.keys(stepMap).forEach(step => {

if (step === currentStep) {

updateStepStatus(stepMap[step], 'active');

} else if (Object.keys(stepProgress).indexOf(step) < Object.keys(stepProgress).indexOf(currentStep)) {

updateStepStatus(stepMap[step], 'completed');

}

});

}

}

async function processingCompleted(status) {

const progressText = document.getElementById('progress-text');

if (progressText) progressText.textContent = 'Processing Complete!';

if (status.results?.chunking) {

document.getElementById('total-chunks').textContent = status.results.chunking.total\_chunks.toLocaleString();

}

if (status.total\_time) {

document.getElementById('processing-time').textContent = Math.round(status.total\_time) + 's';

}

const processBtn = document.getElementById('process-btn');

if (processBtn) {

processBtn.textContent = 'Processed';

processBtn.disabled = true;

}

isProcessing = false;

setTimeout(showQuerySection, 1000);

}

function processingFailed(error) {

const progressText = document.getElementById('progress-text');

if (progressText) progressText.textContent = 'Processing Failed!';

alert(`Processing failed: ${error}`);

const processBtn = document.getElementById('process-btn');

if (processBtn) {

processBtn.disabled = false;

processBtn.textContent = 'Start Processing';

}

isProcessing = false;

}

// Query functions

function showQuerySection() {

const querySection = document.getElementById('query-section');

if (querySection) {

querySection.style.display = 'block';

setTimeout(() => {

querySection.classList.add('popup-show');

}, 100);

}

}

async function performQuery() {

const queryInput = document.getElementById('query-input');

const queryResults = document.getElementById('query-results');

const query = queryInput.value.trim();

if (!query) {

alert('Please enter a search query');

return;

}

if (!currentFileId) {

alert('No processed file available');

return;

}

try {

const response = await fetch(`${API\_BASE}/query`, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({

file\_id: currentFileId,

query: query,

config: { top\_k: 5, similarity\_metric: 'cosine' }

})

});

if (!response.ok) throw new Error('Query failed');

const result = await response.json();

displayQueryResults(result.results, queryResults);

} catch (error) {

console.error('Query error:', error);

displayMockResults(query, queryResults);

}

}

function displayQueryResults(results, container) {

if (!container) return;

container.innerHTML = results.map((item, index) => `

<div class="query-result-item">

<div class="query-result-title">Result ${index + 1} (Score: ${item.score?.toFixed(3) || 'N/A'})</div>

<div class="query-result-content">${item.content}</div>

</div>

`).join('');

}

function displayMockResults(query, container) {

if (!container) return;

const mockResults = [

{ content: `Customer data analysis for query: "${query}"`, score: 0.95 },

{ content: `Product performance metrics for query: "${query}"`, score: 0.87 },

{ content: `Market trend analysis for query: "${query}"`, score: 0.76 }

];

displayQueryResults(mockResults, container);

}

// Configuration management

function getModeString(layer) {

switch(layer) {

case 1: return 'fast';

case 2: return 'config';

case 3: return 'deep\_config';

default: return 'fast';

}

}

function gatherConfiguration() {

const config = {

preprocessing: {},

chunking: {},

embedding: {},

storage: {}

};

if (currentLayer >= 2) {

config.preprocessing = {

fill\_null\_strategy: getValue('null-handling', 'auto'),

remove\_stopwords\_flag: getChecked('remove-stopwords', false)

};

config.chunking = {

method: getValue('chunking-method', 'semantic'),

chunk\_size: getNumberValue('chunk-size', 512)

};

config.embedding = {

model\_name: getValue('embedding-model', 'all-MiniLM-L6-v2')

};

config.storage = {

backend: getValue('storage-backend', 'faiss')

};

}

if (currentLayer >= 3) {

config.preprocessing = {

...config.preprocessing,

remove\_punctuation: getChecked('remove-punctuation', false),

stemming: getChecked('stemming', false),

lemmatization: getChecked('lemmatization', false)

};

const nClusters = getNumberValue('num-clusters', null);

if (nClusters) config.chunking.n\_clusters = nClusters;

}

return config;

}

function getValue(id, defaultValue) {

const element = document.getElementById(id);

return element ? element.value : defaultValue;

}

function getNumberValue(id, defaultValue) {

const element = document.getElementById(id);

return element ? parseInt(element.value) : defaultValue;

}

function getChecked(id, defaultValue) {

const element = document.getElementById(id);

return element ? element.checked : defaultValue;

}

// Range slider updates

function updateRangeValue(sliderId, valueId) {

const slider = document.getElementById(sliderId);

const valueSpan = document.getElementById(valueId);

if (!slider || !valueSpan) return;

let value = slider.value;

if (sliderId.includes('overlap')) {

value += '%';

} else if (sliderId.includes('threshold') || sliderId.includes('temperature')) {

value = parseFloat(value).toFixed(2);

} else if (sliderId.includes('cache-size')) {

value += 'MB';

}

valueSpan.textContent = value;

}

// Reset functions

function resetProcessing() {

currentLayer = 1;

isProcessing = false;

uploadedFile = null;

currentFileId = null;

processingStartTime = null;

stepTimers = {};

stepStartTimes = {};

fileUploaded = false;

// Reset UI elements

const elementsToReset = [

{ id: 'query-section', action: 'hide' },

{ id: 'query-input', action: 'clear' },

{ id: 'query-results', action: 'clear' },

{ id: 'processing-pipeline-section', action: 'hide' },

{ id: 'sidebar', action: 'hide' },

{ id: 'file-upload-area', action: 'reset' },

{ id: 'process-btn', action: 'reset' }

];

elementsToReset.forEach(({ id, action }) => {

const element = document.getElementById(id);

if (!element) return;

switch (action) {

case 'hide':

element.style.display = 'none';

element.classList.remove('popup-show', 'show');

break;

case 'clear':

element.innerHTML = '';

break;

case 'reset':

if (id === 'file-upload-area') {

element.classList.remove('uploaded');

element.innerHTML = `

<div class="upload-icon">📁</div>

<h3>Upload CSV File</h3>

<p>Drag and drop your CSV file here or click to browse</p>

<input type="file" id="csvFile" accept=".csv" onchange="handleFileUpload(event)">

`;

} else if (id === 'process-btn') {

element.textContent = 'Start Processing';

element.disabled = false;

}

break;

}

});

// Show initial sections

document.querySelector('.layer-selection').style.display = 'flex';

document.querySelector('.file-upload-section').style.display = 'block';

// Reset step statuses

const steps = ['step-upload', 'step-preprocess', 'step-chunking', 'step-embedding', 'step-storage'];

steps.forEach(stepId => updateStepStatus(stepId, 'pending'));

resetAllTimers();

setupDragDrop(); // Re-setup drag and drop

}

function resetEntireProcess() {

if (confirm('Do you want to reset the entire process? This will clear all data and return to the start page.')) {

resetProcessing();

}

}

function resetConfiguration() {

if (confirm('Reset all configurations?')) {

resetEntireProcess();

}

}

function saveConfiguration() {

const config = gatherConfiguration();

const configJson = JSON.stringify(config, null, 2);

const blob = new Blob([configJson], { type: 'application/json' });

const url = URL.createObjectURL(blob);

const a = document.createElement('a');

a.href = url;

a.download = `chunking\_config\_layer\_${currentLayer}.json`;

a.click();

URL.revokeObjectURL(url);

alert('Configuration saved successfully!');

}

// Event listeners

document.addEventListener('DOMContentLoaded', function() {

// Query input enter key

const queryInput = document.getElementById('query-input');

if (queryInput) {

queryInput.addEventListener('keypress', function(e) {

if (e.key === 'Enter') performQuery();

});

}

// Range slider initializations

document.querySelectorAll('input[type="range"]').forEach(slider => {

const valueId = slider.id + '-value';

const valueElement = document.getElementById(valueId);

if (valueElement) {

updateRangeValue(slider.id, valueId);

}

slider.addEventListener('input', function() {

updateRangeValue(slider.id, valueId);

});

});

initializeApp();

});

// Initialize application

function initializeApp() {

console.log('Initializing CSV Chunking Optimizer...');

setupDragDrop();

console.log('Application initialized successfully!');

}