Week 2 Development Todo List - File Upload & Processing

Phase 2: File Upload & Processing

Goal: Let users upload files and extract text for chatbot training

---

Day 8: File Upload Infrastructure

Morning (3-4 hours)

- [ ] **Install File Processing Dependencies**

```bash
npm install multer formidable
npm install pdf-parse mammoth xlsx
npm install sharp tesseract.js  # For OCR
npm install file-type mime-types
```

- [ ] **Configure Supabase Storage**
  - Create storage bucket in Supabase dashboard
  - Set up bucket policies for file access
  - Configure file upload size limits
  - Test bucket permissions

Afternoon (3-4 hours)

- [ ] **Create File Upload Components**

```javascript
// components/upload/FileDropzone.jsx
// components/upload/FileUploadCard.jsx
// components/upload/UploadProgress.jsx
// components/upload/FilePreview.jsx
```

- [ ] **Build File Upload UI**
  - Drag & drop interface
  - File type validation
  - Upload progress indicators
  - File size validation (max 10MB for free tier)

Evening (2-3 hours)

☐ **Create Upload API Routes**

```javascript
// app/api/upload/route.js
// app/api/files/[id]/route.js
// app/api/files/delete/route.js
```

☐ **Basic file validation and storage**

- Check file types (PDF, DOCX, XLSX, images)
- Store files in Supabase Storage
- Save file metadata to database

Day 9: PDF Text Extraction

Morning (3-4 hours)

☐ **PDF Processing Setup**

```javascript
// lib/processors/pdfProcessor.js
```

☐ **Implement PDF Text Extraction**

- Use pdf-parse library
- Handle multi-page documents
- Extract text while preserving structure
- Handle password-protected PDFs (basic error handling)

Afternoon (3-4 hours)

☐ **PDF Processing API**

```javascript
// app/api/process/pdf/route.js
```

☐ **Text Chunking Function**

```javascript
// lib/utils/textChunking.js
```

- Split text into 500-1000 word chunks
- Preserve sentence boundaries

- Maintain context between chunks
- Add metadata (page numbers, sections)

Evening (2-3 hours)

- [ ] **Update Database Schema**

```sql
-- Add processing status and chunks table
CREATE TABLE content_chunks (
  id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
  training_data_id UUID REFERENCES training_data(id) ON DELETE CASCADE,
  content TEXT NOT NULL,
  chunk_index INTEGER NOT NULL,
  metadata JSONB DEFAULT '{}',
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

Day 10: Document Processing (Word & Excel)

Morning (3-4 hours)

- [ ] **Word Document Processing**

```javascript
// lib/processors/docxProcessor.js
```

- Use mammoth library for DOCX files
- Extract text content
- Handle formatting and styles
- Process tables and lists

Afternoon (3-4 hours)

- [ ] **Excel File Processing**

```javascript
// lib/processors/xlsxProcessor.js
```

- Use xlsx library
- Process multiple worksheets

- Convert tabular data to readable text
- Handle formulas and formatting

Evening (2-3 hours)

☐ **Document Processing API Routes**

```javascript
// app/api/process/docx/route.js
// app/api/process/xlsx/route.js
```

☐ **Error Handling & Validation**

- Handle corrupted files
- File size validation
- Processing timeout handling
- User feedback for failed uploads

Day 11: Image Processing & OCR

Morning (3-4 hours)

☐ **Image OCR Setup**

```javascript
// lib/processors/imageProcessor.js
```

☐ **Implement OCR with Tesseract.js**

- Extract text from images (PNG, JPG, JPEG)
- Handle different image qualities
- Support multiple languages (English primary)
- Optimize image preprocessing

Afternoon (3-4 hours)

☐ **Image Processing API**

```javascript
// app/api/process/image/route.js
```

☐ **Image Optimization**

- Resize large images before OCR

- Enhance image quality for better text recognition
- Handle different image formats
- Progress tracking for OCR processing

Evening (2-3 hours)

☐ **File Type Detection & Routing**

```javascript
// lib/utils/fileTypeHandler.js
```

- Automatically detect file types
- Route to appropriate processor
- Handle unsupported file types
- Provide user feedback

Day 12: File Management Dashboard

Morning (3-4 hours)

☐ **Create File Management Components**

```javascript
// components/dashboard/FileManager.jsx
// components/dashboard/FileList.jsx
// components/dashboard/FileCard.jsx
// components/dashboard/ProcessingStatus.jsx
```

Afternoon (3-4 hours)

☐ **Build File Management Page**

- `app/dashboard/files/page.jsx`
- Display uploaded files with status
- Show processing progress
- File preview capabilities
- Delete/re-process options

Evening (2-3 hours)

☐ **File Operations**

- View file content (processed text)
- Delete files and associated data
- Re-process failed files
- Bulk operations (select multiple files)

Day 13: Processing Queue & Background Jobs

Morning (3-4 hours)

☐ **Implement Processing Queue**

```javascript
// lib/queue/fileProcessor.js
```

- Queue system for file processing
- Handle multiple files simultaneously
- Priority processing (smaller files first)
- Retry failed processing

Afternoon (3-4 hours)

☐ **Background Processing**

```javascript
// lib/workers/processFile.js
```

- Async file processing
- Real-time status updates
- Progress tracking
- Error logging and recovery

Evening (2-3 hours)

☐ **Processing Status API**

```javascript
// app/api/processing/status/[id]/route.js
// app/api/processing/retry/[id]/route.js
```

- Get processing status
- Retry failed processing

- Cancel processing
- Get processing logs

Day 14: Integration & Testing

Morning (3-4 hours)

☐ **File Upload Integration**

- Connect upload UI to chatbot creation
- Add files to existing chatbots
- Update training data when files are processed
- File management in chatbot settings

Afternoon (3-4 hours)

☐ **Comprehensive Testing**

- Test all supported file formats
- Test large file uploads
- Test concurrent file processing
- Test error scenarios (corrupted files, network issues)

Evening (2-3 hours)

☐ **Performance Optimization**

- Optimize file processing speed
- Memory usage optimization
- Database query optimization
- Frontend loading states

Week 2 Code Examples

File Upload Component Example

```javascript
```

```
// components/upload/FileDropzone.jsx
'use client';
import { useState, useCallback } from 'react';
import { Upload, File, AlertCircle } from 'lucide-react';

export default function FileDropzone({ onFileUpload, maxSize = 10 * 1024 * 1024 }) {
  const [isDragOver, setIsDragOver] = useState(false);
  const [uploading, setUploading] = useState(false);

  const handleDrop = useCallback(async (e) => {
    e.preventDefault();
    setIsDragOver(false);
    setUploading(true);

    const files = Array.from(e.dataTransfer.files);
    const validFiles = files.filter(file => {
      const validTypes = ['application/pdf', 'application/vnd.openxmlformats-officedocument.wordprocessingml.documer
                    'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet', 'image/png', 'image/jpeg'];
      return validTypes.includes(file.type) && file.size <= maxSize;
    });

    for (const file of validFiles) {
      await onFileUpload(file);
    }
    setUploading(false);
  }, [onFileUpload, maxSize]);

  return (
    <div
      className={`border-2 border-dashed rounded-lg p-8 text-center transition-colors
        ${isDragOver ? 'border-blue-500 bg-blue-50' : 'border-gray-300'}
        ${uploading ? 'opacity-50 pointer-events-none' : ''}`}
      onDrop={handleDrop}
      onDragOver={(e) => { e.preventDefault(); setIsDragOver(true); }}
      onDragLeave={() => setIsDragOver(false)}
    >
      <Upload className="mx-auto h-12 w-12 text-gray-400 mb-4" />
      <h3 className="text-lg font-medium text-gray-900 mb-2">
        {uploading ? 'Uploading...' : 'Upload your files'}
      </h3>
      <p className="text-sm text-gray-500">
        Drag and drop your files here, or click to browse
        <br />
      Supports: PDF, Word, Excel, Images (max 10MB)
    </p>
  </div>
 );
}
```

PDF Processor Example

```
javascript
```

```
// lib/processors/pdfProcessor.js
import pdf from 'pdf-parse';

export async function processPDF(buffer) {
  try {
    const data = await pdf(buffer);

    // Extract text and metadata
    const extractedText = data.text;
    const metadata = {
      pages: data.numpages,
      info: data.info
    };

    // Clean and structure the text
    const cleanedText = cleanText(extractedText);

    // Split into chunks
    const chunks = chunkText(cleanedText, 800);

    return {
      success: true,
      text: cleanedText,
      chunks,
      metadata
    };
  } catch (error) {
    return {
      success: false,
      error: error.message
    };
  }
}

function cleanText(text) {
  return text
    .replace(/\s+/g, ' ')          // Multiple spaces to single
    .replace(/\n\s*\n/g, '\n\n')    // Multiple newlines to double
    .trim();
}

function chunkText(text, maxLength = 800) {
  const sentences = text.split(/[.!?]+/).filter(s => s.trim().length > 0);
  const chunks = [];
  let currentChunk = '';

  for (const sentence of sentences) {
    if ((currentChunk + sentence).length > maxLength && currentChunk.length > 0) {
      chunks.push(currentChunk.trim());
      currentChunk = sentence;
    } else {
      currentChunk += sentence + '. ';
    }
  }

  if (currentChunk.trim().length > 0) {
    chunks.push(currentChunk.trim());
  }

  return chunks;
}
```

File Upload API Route

```javascript
```

```javascript
// app/api/upload/route.js
import { createRouteHandlerClient } from '@supabase/auth-helpers-nextjs';
import { cookies } from 'next/headers';
import { NextResponse } from 'next/server';
import { processPDF } from '@/lib/processors/pdfProcessor';
import { processDocx } from '@/lib/processors/docxProcessor';
import { processImage } from '@/lib/processors/imageProcessor';

export async function POST(request) {
  const supabase = createRouteHandlerClient({ cookies });

  try {
    // Get user
    const { data: { user }, error: authError } = await supabase.auth.getUser();
    if (authError || !user) {
      return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
    }

    // Parse form data
    const formData = await request.formData();
    const file = formData.get('file');
    const chatbotId = formData.get('chatbotId');

    if (!file) {
      return NextResponse.json({ error: 'No file provided' }, { status: 400 });
    }

    // Validate file
    const buffer = Buffer.from(await file.arrayBuffer());
    const fileType = file.type;
    const fileName = file.name;

    // Upload to Supabase Storage
    const fileKey = `${user.id}/${chatbotId}/${Date.now()}-${fileName}`;
    const { data: uploadData, error: uploadError } = await supabase.storage
      .from('training-files')
      .upload(fileKey, buffer, {
        contentType: fileType,
        cacheControl: '3600'
      });

    if (uploadError) {
      return NextResponse.json({ error: 'Upload failed' }, { status: 500 });
```

```
}

// Save to database
const { data: trainingData, error: dbError } = await supabase
  .from('training_data')
  .insert({
    chatbot_id: chatbotId,
    type: 'file',
    title: fileName,
    source_url: fileKey,
    file_size: buffer.length,
    processing_status: 'pending'
  })
  .select()
  .single();

if (dbError) {
  return NextResponse.json({ error: 'Database error' }, { status: 500 });
}

// Process file based on type
let processedData;
if (fileType === 'application/pdf') {
  processedData = await processPDF(buffer);
} else if (fileType.includes('wordprocessing')) {
  processedData = await processDocx(buffer);
} else if (fileType.includes('image')) {
  processedData = await processImage(buffer);
}

// Update with processed content
if (processedData.success) {
  await supabase
    .from('training_data')
    .update({
      content: processedData.text,
      processing_status: 'completed'
    })
    .eq('id', trainingData.id);

  // Store chunks
  const chunkInserts = processedData.chunks.map((chunk, index) => ({
    training_data_id: trainingData.id,
    content: chunk,
        chunk_index: index,
        metadata: { page: Math.floor(index / 3) + 1 }
      }));

      await supabase.from('content_chunks').insert(chunkInserts);
    }

    return NextResponse.json({
      id: trainingData.id,
      status: processedData.success ? 'completed' : 'failed',
      message: processedData.success ? 'File processed successfully' : processedData.error
    });

  } catch (error) {
    console.error('Upload error:', error);
    return NextResponse.json({ error: 'Internal server error' }, { status: 500 });
  }
}
```

Week 2 Success Criteria ✅

By the end of week 2, you should have:

1. **Complete File Upload System**
   - Drag & drop file interface
   - Support for PDF, DOCX, XLSX, and images
   - File validation and size limits
   - Progress tracking and error handling
2. **Text Extraction Pipeline**
   - PDF text extraction with pdf-parse
   - Word document processing with mammoth
   - Excel data conversion to readable text
   - OCR for images using Tesseract.js
3. **File Management Dashboard**
   - View all uploaded files
   - Processing status tracking
   - Delete and re-process capabilities
   - File content preview

4. **Robust Processing System**
   - Background processing queue
   - Error handling and retry logic
   - Text chunking for AI training
   - Database storage of processed content

Environment Variables to Add

```env
# File processing limits
MAX_FILE_SIZE=10485760  # 10MB
MAX_FILES_PER_USER=100
PROCESSING_TIMEOUT=300000  # 5 minutes

# OCR settings
TESSERACT_LANG=eng
OCR_TIMEOUT=120000  # 2 minutes
```

Daily Time Allocation

- **Morning (3-4 hours)**: Core file processing development
- **Afternoon (3-4 hours)**: API routes and integration
- **Evening (2-3 hours)**: Testing, optimization, bug fixes

Total: 7-11 hours per day

Tips for Week 2

1. **Test with real files** - Use actual PDFs, Word docs, and images
2. **Handle errors gracefully** - File processing can fail in many ways
3. **Optimize for performance** - Large files can slow down processing
4. **Monitor memory usage** - File processing is memory-intensive
5. **Implement proper cleanup** - Delete temporary files and failed uploads
6. **User feedback is crucial** - Show clear status and error messages

This completes the file upload and processing foundation. Next week, you'll add website scraping capabilities!