Document Processing System - Technical Report

Website Link: https://document-processing-app-ashen.vercel.app/

1. Introduction

The Document Processing System is a full-stack web application designed to extract text and information from uploaded documents (PDFs and images) using both standard extraction methods and AI-powered analysis. The system includes:

- **Document Upload**: Secure file upload with form data collection
- **Text Extraction**: Standard extraction using Tesseract.js and pdf-parse
- **AI Analysis**: Advanced information extraction using Google's Gemini AI
- **Results Display**: Clean, organized presentation of extracted data
- **Document Management**: History and tracking of processed documents

This report covers the technical implementation, architecture decisions, deployment process, and challenges encountered during development.

2. Technical Implementation

2.1 System Architecture

```
Frontend (Vercel) → Backend (Fly.io) → Supabase (Database & Storage)
```

2.2 Frontend Architecture

- **Framework**: Next.js 14 with TypeScript
- **Styling**: Tailwind CSS with custom components
- **State Management**: Redux Toolkit with slices for theme, upload, jobs, and results
- **UI Components**: shadcn/ui with Heroicons
- **API Client**: Native fetch with error handling

2.3 Backend Architecture

- **Framework**: Express.js with Node.js
- **File Processing**: Multer for uploads, pdf-parse and Tesseract.js for extraction
- **AI Integration**: Google Gemini AI for advanced data extraction
- **Database**: Supabase PostgreSQL for metadata and file storage
- **Authentication**: JWT tokens for future user management

2.4 API Endpoints

Endpoint	Method	Purpose
`/api/upload`	POST	Handle document upload and processing

Endpoint	Method	Purpose
`/api/results/:jobId`	GET	Retrieve processing results
`/api/results`	GET	Get all processing jobs
`/api/health`	GET	System health check
`/api/results/:jobId/document`	GET	Get the original document
`/api/results/:jobId`	DELETE	Delete a seleted document

3. Core Features

3.1 Document Upload

- **File Type Support**: PDF, JPG, JPEG, PNG
- **Size Limit**: 10MB maximum file size
- **Form Data**: First name, last name, date of birth
- **Processing Method**: Standard extraction or AI analysis

3.2 Text Extraction

- **PDF Processing**: Dual approach (direct text extraction + OCR fallback using pdftoppm)
- **Image Processing**: Tesseract.js OCR with optimization
- **Text Cleaning**: Removal of artifacts and formatting issues

3.3 AI Information Extraction

- **Gemini AI Integration**: Structured data extraction from text
- **Data Categories**:
  - Important Information (names, dates)
  - Contact Information (emails, phone numbers)
  - Addresses and locations
  - Identification numbers
  - Document summary

3.4 Results Display

- **Real-time Updates**: Polling for processing status
- **Side-by-Side Comparison**: Standard vs AI extraction results
- **Responsive Design**: Mobile-friendly interface
- **Dark/Light Mode**: Theme switching with persistence

4. Technical Challenges & Solutions

4.1 PDF Processing Challenges

Challenge: PDF text extraction inconsistency across different PDF types (text-based vs scanned)

Solution: Implemented dual extraction approach:

1. Primary: Direct text extraction using pdf-parse
2. Fallback: OCR conversion using pdftoppm + Tesseract.js

```
// Implementation for Windows compatibility
async function extractTextWithOCR(pdfBuffer) {
  try {
    // Windows-compatible PDF to image conversion
    const images = await convertPdfToImages(pdfBuffer);
    let fullText = '';

    for (const imagePath of images) {
      const result = await Tesseract.recognize(imagePath, 'eng');
      fullText += result.data.text + '\n\n';
    }

    return cleanExtractedText(fullText);
  } catch (error) {
    throw new Error(`OCR failed: ${error.message}`);
  }
}
```

4.2 Docker Compatibility Issues

Challenge: pdftoppm not available in all Docker base images, especially Windows containers

Solution: Multi-platform Docker setup with conditional installation:

```
# Backend Dockerfile with multi-platform support
FROM node:18-alpine

# Install Poppler utilities based on platform
RUN if [ "$(uname -m)" = "x86_64" ]; then \
        apk add --no-cache poppler-utils; \
    elif [ "$(uname -m)" = "aarch64" ]; then \
        apk add --no-cache poppler-utils; \
    else \
        echo "Platform not supported"; exit 1; \
    fi

# Alternative for Windows containers
# RUN choco install poppler -y
```

4.3 AI Data Consistency

Challenge: Gemini AI sometimes returning inconsistent JSON structures

Solution: Enhanced parsing with fallbacks and validation:

```
function parseAIResponse(aiText) {
  try {
    // Multiple JSON extraction strategies
    const jsonMatch = aiText.match(/```json\n([\s\S]*?)\n```/)
                   || aiText.match(/({[\s\S]*})/);

    if (jsonMatch) {
      const jsonString = jsonMatch[1] || jsonMatch[0];
      const parsedData = JSON.parse(jsonString);

      // Validate and normalize structure
      return normalizeAIStructure(parsedData);
    }
    return { rawResponse: aiText };
  } catch (error) {
    return { error: "Failed to parse AI response", rawResponse: aiText };
  }
}
```

5. Deployment Architecture

5.1 Frontend (Vercel)

- **Platform**: Vercel for Next.js optimization
- **Environment Variables**:
  - `NEXT_PUBLIC_API_URL`: Backend API endpoint
  - `NEXT_PUBLIC_SUPABASE_URL`: Supabase instance URL
  - `NEXT_PUBLIC_SUPABASE_ANON_KEY`: Supabase authentication key

5.2 Backend (Fly.io)

- **Platform**: Fly.io with Docker deployment
- **Dependencies**: System libraries for PDF processing (poppler-utils)
- **Environment Variables**:
  - `SUPABASE_URL`: Database connection string
  - `SUPABASE_KEY`: Database authentication key
  - `GEMINI_API_KEY`: Google AI API key

5.3 Database & Storage (Supabase)

- **PostgreSQL**: Structured data storage for jobs and metadata
- **Storage**: File storage for uploaded documents
- **Realtime**: Potential for real-time updates (future enhancement) Use **Web Sockets** currently using **Short Polling**. Web Sockets were not used because they are almost impossible to setup on free Supabase Tier.

6. Docker Implementation

6.1 Multi-Platform Docker Support

The application includes Docker configuration that works across different platforms:

```
# backend/Dockerfile
FROM node:18-alpine

# Install platform-specific dependencies
RUN case "$(uname -m)" in \
    x86_64|aarch64) \
        apk add --no-cache poppler-utils tesseract-ocr tesseract-ocr-data-eng ;; \
    *) \
        echo "Unsupported architecture: $(uname -m)" && exit 1 ;; \
    esac

# Application setup
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY . .
EXPOSE 3001
CMD ["npm", "start"]
```

6.2 Docker Compose for Development

```
version: '3.8'
services:
  backend:
    build: ./backend
    ports: ["3001:3001"]
    environment:
      - NODE_ENV=development
      - SUPABASE_URL=${SUPABASE_URL}
      - SUPABASE_KEY=${SUPABASE_KEY}
    volumes:
      - ./backend:/app
      - /app/node_modules

  frontend:
    build: ./frontend
    ports: ["3000:3000"]
    environment:
      - NEXT_PUBLIC_API_URL=http://localhost:3001
    depends_on:
      - backend
```

7. UI/UX Decisions

7.1 Design System

- **Color Scheme**:

    - Red gradient: Primary actions and buttons
    - Blue: Information and results
    - Gray/Black: Neutral backgrounds and text
- **Typography**: Inter font family for readability
- **Icons**: Heroicons for consistent iconography

7.2 User Experience

- **Drag-and-Drop Upload**: Intuitive file selection
- **Real-time Progress**: Status updates during processing
- **Responsive Layout**: Mobile-first design approach, followed by laptops.
- **Theme Support**: Dark/light mode with system preference detection

7.3 Error Handling

- **Graceful Failures**: User-friendly error messages
- **Retry Mechanisms**: Automatic retry for failed operations
- **Validation**: Client and server-side form validation

8. Future Improvements

1. **Real-time Updates**: WebSocket integration for live progress updates, currently using Short Polling.
2. **User Authentication**: JWT-based user accounts and document privacy
3. **Batch Processing**: Support for multiple document uploads
4. **Advanced AI Models**: Custom-trained models for specific document types
5. **Export Functionality**: PDF/CSV export of extracted data
6. **API Rate Limiting**: Protection against abuse
7. **Admin Dashboard**: Management interface for system monitoring

9. Conclusion

The Document Processing System successfully demonstrates a modern full-stack application with complex file processing capabilities. Key achievements include:

- **Robust PDF Handling**: Support for both text-based and scanned PDFs through dual extraction methods
- **AI Integration**: Effective use of Gemini AI for structured data extraction
- **Cross-Platform Deployment**: Docker configuration that works on macOS, Linux, and Windows
- **Production Ready**: Deployment pipelines for Vercel and Fly.io
- **User-Friendly Interface**: Intuitive design with responsive layout

The system provides a solid foundation for document processing applications and can be extended with additional features like user management, advanced AI models, and real-time collaboration.

Deployment Links:

- Frontend GitHub Repository: https://github.com/kimocks-netizen/document-processing-app-frontend
- Backend GitHub Repository: https://github.com/kimocks-netizen/backend-document-processing-app

Useful Links:

- Website Link: https://document-processing-app-ashen.vercel.app/
- Backend APIs Link: https://backend-document-processing-app.fly.dev

Developed by: Bryne (Full Stack Developer)