Synapse AI - System Architecture Guide

Overview

Synapse AI is a comprehensive document processing and translation system that combines OCR, translation, and AI-powered chat capabilities. The system is built using a modern tech stack with a clear separation between frontend and backend components.

System Architecture

1. Frontend (React + Vite)

Location: `/frontend`

The frontend is built using React with Vite as the build tool. Key components include:

- **User Interface Components** (`/frontend/src/components/`)
  - Document upload and management (`MultilingualOCR.jsx`)
  - OCR results display (`OCRResult.jsx`)
  - Translation interface (`TranslatedText.jsx`)
  - Chat interface (`GeminiChat.jsx`)
  - Text-to-speech controls (integrated in `GeminiChat.jsx`)
- **State Management**
  - React Context for global state
  - Local state for component-specific data
  - LocalStorage for chat history persistence
- **API Integration**
  - RESTful API calls to backend services
  - Google Gemini API for chat functionality
  - Web Speech API for text-to-speech

2. Backend (Python + FastAPI)

Location: `/backend/app/main.py`

The backend is implemented as a single FastAPI application with the following key features:

- **OCR Processing**
  - Tesseract OCR integration with support for multiple languages (English, Spanish, Malayalam, French)
  - PDF and image file support
  - Text cleaning and paragraph preservation
  - Endpoint: `/api/ocr`
- **Translation Service**
  - Google Translate API integration
  - Language code mapping between OCR and translation services
  - Text cleaning and formatting preservation
  - Endpoint: `/api/translate`
- **Core Utilities**
  - Text cleaning and formatting
  - File handling and processing
  - Error handling and logging
  - CORS middleware for frontend communication

Feature Implementation Details

1. Multilingual OCR

- **Implementation**: Direct integration with Tesseract OCR in `main.py`
- **Key Functions**:
  - `ocr()`: Handles file upload and OCR processing
  - `clean_text()`: Preserves paragraph structure while cleaning text
- **Supported Languages**: English (eng), Spanish (spa), Malayalam (mal), French (fra)

2. Text Translation

- **Implementation**: Google Translate API integration in `main.py`
- **Key Functions**:
  - `translate_text()`: Handles translation requests
  - Language mapping between OCR and translation services
- **Features**:
  - Automatic language detection
  - Text cleaning and formatting preservation
  - Error handling and validation

3. AI Chat Interface

- **Implementation**: Google Gemini API integration in `GeminiChat.jsx`
- **Key Features**:
  - Document context-aware responses
  - Markdown support for formatted responses
  - Chat history persistence using LocalStorage
  - Real-time message updates
  - Loading indicators
- **Key Functions**:
  - `handleSendMessage()`: Processes user queries and fetches AI responses
  - `handleInputChange()`: Manages text input with auto-resizing
  - `handleKeyDown()`: Handles keyboard shortcuts

4. Text-to-Speech

- **Implementation**: Web Speech API integration in `GeminiChat.jsx`
- **Key Features**:
  - Real-time speech synthesis
  - Chunk-based text processing for long messages
  - Voice selection (prefers Google UK English Female)

  - Speech rate control
  - Toggle functionality for starting/stopping speech
- **Key Functions**:
  - `handleSpeak()`: Manages speech synthesis
  - `speakText()`: Processes text into speech chunks
  - `speakNextChunk()`: Handles sequential speech synthesis

5. File Processing

- **Implementation**: Integrated file handling in `main.py`
- **Features**:
  - PDF and image file support
  - File validation
  - Error handling
  - Memory-efficient processing

Technical Stack

Frontend

- React
- Vite
- Material-UI
- Axios
- WebSocket
- Web Speech API
- Google Gemini API
- ReactMarkdown

Backend

- Python
- FastAPI

- Tesseract OCR
- Google Translate API
- PIL (Python Imaging Library)
- pdf2image

Development Setup

1. **Frontend Setup**

```
cd frontend
npm install
npm run dev
```

1. **Backend Setup**

```
cd backend
python -m venv .venv
source .venv/bin/activate  # or .venv\\Scripts\\activate on Windows
pip install -r requirements.txt
uvicorn app.main:app --reload
```

API Documentation

The backend API is documented using FastAPI's automatic OpenAPI documentation, accessible at:

- Swagger UI: `http://localhost:8000/docs`
- ReDoc: `http://localhost:8000/redoc`

Security Considerations

- File upload validation
- CORS configuration
- Input sanitization
- Error handling
- API key management for Gemini API

Future Improvements

- Enhanced language support
- Advanced OCR capabilities
- User authentication
- Cloud storage integration
- Role-based access control
- Enhanced voice options for text-to-speech
- Improved chat context management