Document-Based Question Answering System

Team 70 - Contributors

Name	Email Address	Contributions %
NEERAJ BHATT	2024aa05020@wilp.bits-pilani.ac.in	100%
V. S. BALAKRISHNAN	2024aa05017@wilp.bits-pilani.ac.in	100%
KURUVELLA VENKATA SAI UPENDRA	2024aa05016@wilp.bits-pilani.ac.in	100%
SAJAL CHAUDHARY	2024aa05026@wilp.bits-pilani.ac.in	100%
SACHIN KUMAR	2024aa05024@wilp.bits-pilani.ac.in	100%

Overview

A web-based Question Answering application that allows users to upload documents (PDF, DOCX, TXT) and ask questions based on the document content. The application provides accurate answers extracted from the uploaded documents with confidence scores and source references.

Features

- **Document Upload**: Support for PDF, DOCX, and TXT formats
- **Multiple Query Methods**:
- Ask questions on uploaded documents
- Ask questions on directly pasted text
- **AI-Powered QA**: Uses pre-trained transformer models for accurate answer extraction
- **Source References**: Every answer includes the source passage and document reference
- **Confidence Scores**: Visual indicators showing the confidence level of each answer
- **Responsive UI**: Modern, user-friendly web interface
- **Real-time Processing**: Fast document processing and query response

Technology Stack

Backend

- **Framework**: FastAPI
- **NLP**: Hugging Face Transformers (RoBERTa-SQuAD2)
- **Document Parsing**: PyPDF2, python-docx
- **Text Processing**: NLTK, scikit-learn

Frontend

- **HTML5**, **CSS3**, **JavaScript (Vanilla)**
- **Bootstrap 5** for responsive design
- **RESTful API** integration

Project Structure

```
doc-qa-system/
├── backend/
│   ├── app/
│   │   ├── models/
│   │   │   └── schemas.py           # Pydantic models
│   │   ├── routes/
│   │   │   ├── documents.py         # Document upload endpoints
│   │   │   └── qa.py                # QA query endpoints
│   │   ├── services/
│   │   │   ├── document_indexer.py  # Document storage & indexing
│   │   │   └── qa_engine.py         # QA processing engine
│   │   ├── utils/
│   │   │   └── document_processor.py # Document parsing
│   │   └── main.py                  # FastAPI app initialization
│   ├── requirements.txt
│   └── run.py                        # Server startup script
│
├── frontend/
│   ├── index.html                   # Main HTML page
│   └── static/
│       ├── css/
│       │   └── style.css            # Custom styles
│       └── js/
│           └── app.js               # Frontend logic
│
└── docs/
    ├── INSTALLATION.md
    ├── API_DOCUMENTATION.md
    ├── DESIGN_CHOICES.md
    └── ENHANCEMENT_PLAN.md
```

Installation

Prerequisites

- Python 3.8 or higher
- pip (Python package manager)
- Modern web browser

Step 1: Clone/Setup Project

```
cd /Users/bvs/Documents/assignments/NLP/doc-qa-system
```

Step 2: Create Virtual Environment (Recommended)

```
# Create virtual environment
python3 -m venv venv

# Activate virtual environment
# On macOS/Linux:
source venv/bin/activate
# On Windows:
# venv\Scripts\activate
```

Step 3: Install Backend Dependencies

```
cd backend
pip install -r requirements.txt
```

Note: First-time model download might take a few minutes as it downloads pre-trained transformers.

Step 4: Run the Backend Server

```
python run.py
```

The API will be available at `http://localhost:8000`

- API Docs: `http://localhost:8000/docs`
- ReDoc: `http://localhost:8000/redoc`

Step 5: Access the Frontend

Open your web browser and navigate to:

```
file:///Users/bvs/Documents/assignments/NLP/doc-qa-system/frontend/index.html
```

Or alternatively, serve it with a simple HTTP server:

```
# In a new terminal, from frontend directory
cd ../frontend
python3 -m http.server 8001
```

Then open: `http://localhost:8001`

Usage

Basic Workflow

1. **Upload Documents**
   - Click the upload area or drag-and-drop files
   - Supported formats: PDF, DOCX, TXT
   - Upload one or multiple documents
2. **Ask Questions**
   - Click on the "Uploaded Docs" tab
   - Type your question in the question field
   - Select number of answers needed (1-5)
   - Click "Ask Question"
3. **View Results**
   - Answers appear with confidence scores
   - See highlighted source passages
   - Reference document names for each answer
4. **Direct Text Queries**
   - Switch to "Direct Text" tab

- Paste text directly
- Ask questions without uploading documents

API Endpoints

Document Management

- **POST** `/api/documents/upload` - Upload a document
- **GET** `/api/documents/list` - List all uploaded documents
- **DELETE** `/api/documents/{doc_id}` - Delete a document
- **GET** `/api/documents/stats` - Get indexing statistics
- **POST** `/api/documents/clear` - Clear all documents

Question Answering

- **POST** `/api/qa/ask` - Ask question on uploaded documents
- **POST** `/api/qa/ask-direct` - Ask question on provided text
- **GET** `/api/qa/health` - Health check

API Examples

Upload Document

```
curl -X POST "http://localhost:8000/api/documents/upload" \
  -F "file=@document.pdf"
```

Ask Question

```
curl -X POST "http://localhost:8000/api/qa/ask" \
  -H "Content-Type: application/json" \
  -d '{
    "question": "What is the main topic?",
    "top_k": 3
  }'
```

Ask Direct Question

```
curl -X POST "http://localhost:8000/api/qa/ask-direct" \
  -H "Content-Type: application/json" \
  -d '{
    "text": "Your document text here...",
    "question": "What is the main topic?",
    "top_k": 3
  }'
```

Design Choices

1. **QA Model Selection**

- **Choice**: RoBERTa-base fine-tuned on SQuAD 2.0
- **Reasoning**:
- Excellent performance on extractive QA tasks
- Faster inference than larger models
- Good balance between accuracy and speed
- Handles out-of-context questions better than BERT

2. **Passage Retrieval**

- **Choice**: TF-IDF similarity with sentence windowing
- **Reasoning**:
- Fast retrieval without additional indexing overhead
- Works well for diverse document types

- Provides good baseline for relevance matching
- Can be upgraded to semantic search in future

3. **Storage Strategy**

- **Choice**: In-memory document storage (v1)
- **Reasoning**:
- Simple implementation for MVP
- Fast access and processing
- Sufficient for course assignment scope
- Can be upgraded to database for production

4. **Frontend Technology**

- **Choice**: Vanilla JavaScript with Bootstrap
- **Reasoning**:
- Lightweight and responsive
- No complex build pipeline needed
- Easy to understand and modify
- Can run directly without server

5. **Confidence Scoring**

- **Choice**: Combined TF-IDF similarity + QA model confidence
- **Formula**: $0.3 \times similarity_{score} + 0.7 \times qa_{score}$
- **Reasoning**:
- Balances passage relevance with answer confidence
- Gives more weight to QA model's confidence
- Provides meaningful scoring for user feedback

Performance Characteristics

- **Model Loading Time**: ~5-10 seconds (first run)

- **Document Processing**: ~100-200ms per page
- **Query Processing**: ~500ms-2s (depends on document size)
- **Typical Response Time**: 1-3 seconds end-to-end

Limitations & Future Improvements

Current Limitations

1. In-memory storage (documents lost on restart)
2. Single-hop reasoning only
3. Fixed context window for QA model
4. No user authentication
5. No database persistence

Planned Enhancements (Task B)

1. **Multi-document Querying**: Database integration, pagination
2. **Real-time Indexing**: Queue-based processing, incremental updates
3. **Complex Reasoning**: Multi-hop QA, entity linking, graph-based retrieval
4. **Vector Search**: Semantic embeddings, similarity search
5. **Production Ready**: Authentication, caching, monitoring

Troubleshooting

Issue: Model Download Fails

Solution: Ensure internet connection. Models are downloaded to `~/.cache/huggingface/`

Issue: CORS Errors

Solution: The API has CORS enabled for all origins. Check browser console for specific errors.

Issue: File Upload Fails

Solution: Ensure file format is PDF, DOCX, or TXT and file size is reasonable (<50MB)

Issue: No Answers Found

Solution: Try different documents or rephrased questions. Check that documents contain relevant information.

Testing

Manual Testing Steps

1. Upload a PDF file with technical documentation
2. Ask a factual question about the content
3. Verify answer is extracted from the document
4. Check confidence scores are reasonable
5. Test with multiple documents
6. Try direct text input without uploading

Example Test Documents

Prepare sample documents covering:

- Technical documentation (API specs, user manuals)
- News articles (factual extraction)
- Academic papers (complex reasoning)

Environment Variables

Create `.env` file in backend directory (optional):

```
# Model configuration
QA_MODEL_NAME=deepset/roberta-base-squad2
PASSAGE_WINDOW_SIZE=3
TOP_K_DEFAULT=3

# Server configuration
API_HOST=0.0.0.0
API_PORT=8000
DEBUG=False
```

Dependencies

See `backend/requirements.txt` for complete list. Key dependencies:

- fastapi==0.104.1
- torch==2.0.0
- transformers==4.34.0
- PyPDF2==3.0.1
- python-docx==0.8.11
- scikit-learn==1.3.2

License

Educational use - BITS WILP Assignment

Contact

For issues or questions, contact: Balakrishnan V S (2024aa05017@wilp.bits-pilani.ac.in)

Last Updated: December 2025