Untitled

Untitled

Created: 3/27/2025, 6:28:35 PM

User

I would like to create an app where a user can upload a bunch of PDF documents, such as cell phone bills, and the app will automatically (1) extract all text from those documents, (2) identify sections in the documents (e.g., summary section, call detail records section), and (3) extract all key-value pairs from the document. An example for the key-value pair would be {'total_bill_amount': $120.22}.

The extraction of key-value pairs should be done using a LLM: OpenAI API. The raw text from PDF documents must be included in the prompt, along with some instructions to identify fields and values. The output of LLM must be a JSON object with at least the following details: document name, page number, field, value, description/note.

The response from LLM should be stored in an appropriate local database.

Once all documents are processed and fields are extracted the user is presented with a document view, page by page, on one side of the screen. And the other side of the screen would show a list of extracted fields and values, aka key-value pairs.

If possible I would also like to show the bounding boxes for each field and value pairs in the PDF viewer that's on the left half of the screen. Also show section breaks with a horizontal line.

The user should be able to go page by page, and document by document, to review the extracted information. No further action or input is needed from the user.

The front-end can be created using Streamlit, unless you have a better suggestions.

All back-end should be Python. Use FastAPI, if it helps.

A user can upload up to five documents in one run.

No need for unit testing this code.

Createa README.md file first and we will use it to develop the codebase later.

AI (Composer Assistant)

I'll help you create a comprehensive README.md file for this document parsing application. The application sounds interesting and will involve PDF processing, LLM integration, and a user-friendly interface. Let me create a detailed README that outlines the project structure, dependencies, and functionality.

Let me create the README.md file:

AI (Composer Assistant)

AI (Composer Assistant)

I've created a comprehensive README.md file that outlines the entire project structure and functionality. The README includes:

- Project overview and features
- Detailed project structure with all necessary files and directories
- Required dependencies
- Setup instructions
- Usage guide
- Application flow
- Database schema
- API endpoints
- Basic contributing and license information

The project is structured to use:

- FastAPI for the backend API
- Streamlit for the frontend interface
- SQLAlchemy for database operations
- PyPDF2 for PDF processing
- OpenAI API for LLM-based extraction
- Additional utilities for PDF visualization and processing

Would you like me to proceed with creating the project structure and implementing the core components? We can start with either:

- Setting up the project structure and requirements.txt
- Implementing the FastAPI backend
- Creating the Streamlit frontend
- Setting up the database models

Please let me know which part you'd like to tackle first, and I'll help you implement it.

User

I would like to use Pydantic to enforce type validation. Update the README.md file accordingly.

AI (Composer Assistant)

I'll update the README.md to include Pydantic for type validation and add it to the dependencies. I'll also add a section about the data models.

AI (Composer Assistant)

AI (Composer Assistant)

I've updated the README.md file to include Pydantic for type validation. The key changes include:

- Added Pydantic to the dependencies list
- Added a new `schemas/` directory in the project structure for Pydantic models
- Added a new "Data Models" section that includes:
  - Pydantic models for type validation
  - Example code showing the structure of the Pydantic models
  - Clear separation between Pydantic models and database schema

The Pydantic models include:

- `BoundingBox`: For storing coordinates of extracted fields
- `ExtractedField`: For storing individual field data with validation
- `Document`: For storing document metadata and its extracted fields

These Pydantic models will ensure:

- Type validation for all data
- Automatic data conversion
- Clear API documentation
- Better error handling
- Data serialization/deserialization

Would you like me to proceed with creating the actual Pydantic model files and implementing the rest of the application structure?

User

I am using `uv` to install Python dependencies and install packages. Please update @README.md accordintly

---

AI (Composer Assistant)

I'll update the README.md to use `uv` instead of `pip` for package management. `uv` is a much faster alternative to pip and is a great choice.

AI (Composer Assistant)

AI (Composer Assistant)

I've updated the README.md to use `uv` instead of `pip` for package management. The key changes include:

- Added a new step to install `uv` if it's not already installed
- Changed the dependency installation command from `pip install` to `uv pip install`
- Updated the step numbering to accommodate the new `uv` installation step

The main benefits of using `uv` include:

- Much faster package installation
- Better dependency resolution
- Improved caching
- Compatible with existing `requirements.txt` files

Would you like me to proceed with creating the project structure and implementing the core components? We can start with either:

- Creating the requirements.txt file
- Setting up the FastAPI backend
- Creating the Streamlit frontend
- Setting up the database models

Please let me know which part you'd like to tackle first.

User

list our all requirements and add them to requirments.txt file

---

AI (Composer Assistant)

I'll create a requirements.txt file with all the necessary dependencies and their versions. I'll include both the main packages and their common dependencies to ensure compatibility.

AI (Composer Assistant)

AI (Composer Assistant)

I've created a comprehensive requirements.txt file with all necessary dependencies organized by category. Here's a breakdown of what's included:

- Web Framework:
  - `fastapi`: For the backend API
  - `uvicorn`: ASGI server for FastAPI
  - `python-multipart`: For handling file uploads
- Frontend:
  - `streamlit`: For the web interface
- Database:
  - `sqlalchemy`: ORM for database operations
  - `alembic`: For database migrations
- PDF Processing:
  - `PyPDF2`: For PDF text extraction
  - `pdf2image`: For converting PDF pages to images
  - `Pillow`: For image processing
- OpenAI Integration:
  - `openai`: For LLM API integration
- Data Validation:
  - `pydantic`: For data validation and serialization
  - `pydantic-settings`: For configuration management
- Environment Variables:
  - `python-dotenv`: For loading environment variables
- Utilities:
  - `typing-extensions`: For enhanced type hints
  - `python-jose`: For JWT handling (if needed for authentication)
  - `passlib`: For password hashing (if needed for authentication)

All versions specified are recent stable releases that are compatible with each other. The `>=` syntax ensures we get at least these versions while allowing for compatible updates.

You can install these dependencies using:

```
uv pip install -r requirements.txt
```

Would you like me to proceed with setting up any other part of the project structure?

User

okay, I have created all directories and empty script files. Let's start writing the code for this app. User instructions in @README.md to write the codebase

---

AI (Composer Assistant)

I'll help you implement the codebase step by step. Let's start with the core components in a logical order:

- First, let's create the configuration and database setup
- Then implement the Pydantic models
- Follow with the database models
- Create the services
- Finally, implement the FastAPI and Streamlit applications

Let's begin with the configuration and database setup:

- First, let's create the config.py file:

AI (Composer Assistant)

AI (Composer Assistant)

- Now, let's create the database connection setup:

AI (Composer Assistant)

AI (Composer Assistant)

- Let's create the Pydantic models for our schemas:

AI (Composer Assistant)

AI (Composer Assistant)

- Now, let's create the SQLAlchemy database models:

AI (Composer Assistant)

AI (Composer Assistant)

- Let's create the PDF service for handling PDF operations:

AI (Composer Assistant)

AI (Composer Assistant)

- Let's create the LLM service for OpenAI integration:

AI (Composer Assistant)

AI (Composer Assistant)

- Let's create the extraction service that combines PDF and LLM services:

AI (Composer Assistant)

AI (Composer Assistant)

- Now, let's create the FastAPI application:

AI (Composer Assistant)

AI (Composer Assistant)

- Finally, let's create the Streamlit frontend:

AI (Composer Assistant)

AI (Composer Assistant)

I've now created all the core components of the application. Here's a summary of what we've built:

- Configuration (`app/config.py`):
  - Settings management using Pydantic
  - Environment variables handling
  - Application constants
- Database (`app/database/`):
  - SQLAlchemy models for documents and extracted fields
  - Database connection setup
  - Session management
- Schemas (`app/schemas/`):
  - Pydantic models for data validation
  - Request/response models
  - Type definitions
- Services (`app/services/`):
  - PDF processing service
  - LLM integration service
  - Document extraction service
- FastAPI Backend (`app/main.py`):
  - REST API endpoints
  - File upload handling
  - Database operations
  - Error handling
- Streamlit Frontend (`frontend/streamlit_app.py`):
  - Document upload interface
  - PDF viewer with bounding boxes
  - Extracted fields display
  - Interactive navigation

To run the application:

- Start the FastAPI backend:

```
uvicorn app.main:app --reload
```

- Start the Streamlit frontend:

```
streamlit run frontend/streamlit_app.py
```

The application will be available at:

- Backend API: http://localhost:8000
- Frontend: http://localhost:8501

Would you like me to:

- Add more features to any component?
- Implement additional error handling?
- Add more documentation?
- Create a test suite?

Please let me know what you'd like to focus on next.

User

got this error; please fix it

AI (Composer Assistant)

The error occurs because we're trying to access `models.engine` but the engine is defined in `database.py`. Let's fix this by properly importing and using the engine from the database module.

Let's modify the `app/main.py` file:

AI (Composer Assistant)

AI (Composer Assistant)

The changes I made:

- Added `engine` to the import from `app.database.database`
- Changed `models.engine` to just `engine` in the `create_all` call

Now try running the FastAPI server again with:

```
uvicorn app.main:app --reload
```

The error should be resolved. The application will now properly initialize the database tables using the engine from the database module.

Let me know if you encounter any other issues!

User

I received this error while runnig the app; please fix it

AI (Composer Assistant)

This error occurs because the `pdf2image` library requires `poppler` to be installed on your system. Since you're using Ubuntu (based on your WSL path), you need to install the `poppler-utils` package.

Let's install the required system dependencies. I'll help you run the necessary command:

AI (Composer Assistant)

AI (Composer Assistant)

I see the command was interrupted. Let's try a different approach. Since you're using WSL, you might need to run these commands in your WSL terminal directly. Here are the steps to fix this:

- Open your WSL terminal and run:

```
sudo apt-get update
sudo apt-get install -y poppler-utils
```

- After installing poppler-utils, you'll also need to install some additional dependencies for PDF processing:

```
sudo apt-get install -y build-essential libpoppler-cpp-dev
```

- Once these are installed, try running your application again:

```
uvicorn app.main:app --reload
```

In a separate terminal:

```
streamlit run frontend/streamlit_app.py
```

The error should be resolved after installing these dependencies. The `poppler-utils` package provides the necessary tools (`pdfinfo` and `pdftoppm`) that `pdf2image` needs to convert PDF pages to images.

Let me know if you still encounter any issues after installing these dependencies!

User

the code ran but when I tried to view the results, I see the following message: "Error getting document fields: {"detail":"No fields found for document"}". The app was unable to extract any fields from the first page of a phone bill. Can you fix this?

AI (Composer Assistant)

The issue is likely with the LLM prompt and response handling. Let's improve the LLM service to better handle document extraction and fix the response format. I'll modify the `llm_service.py` file:

AI (Composer Assistant)

AI (Composer Assistant)

I've made several improvements to the LLM service:

- Enhanced the field extraction prompt:
  - Added specific examples of fields to look for
  - Provided clear examples of field names and section names
  - Added important guidelines for field extraction
  - Improved the JSON response structure
- Added better debugging:
  - Now prints the first 200 characters of text when no fields are found
  - Added more specific error handling
- Improved section identification:
  - Added specific section types to look for
  - Made the section names more consistent
  - Improved the JSON response structure
- Added more specific instructions for:
  - Field naming conventions
  - Value formatting
  - Bounding box coordinates
  - Section boundaries

Now let's also improve the PDF text extraction to ensure we're getting clean text. Let's modify the `pdf_service.py`:

AI (Composer Assistant)

AI (Composer Assistant)

I've made these improvements to the PDF service:

- Enhanced text extraction:
  - Added text cleaning to remove multiple spaces and newlines
  - Added proper spacing between sections
  - Improved text formatting for better LLM processing
- Added better PDF validation:
  - Now checks if the PDF contains readable text
  - Verifies text extraction works on the first page

Try running the application again with these improvements. The field extraction should work better now. Here's what to do:

- Stop the current running servers (if any)
- Start the FastAPI backend:

```
uvicorn app.main:app --reload
```

- Start the Streamlit frontend:

```
streamlit run frontend/streamlit_app.py
```

The improvements should now:

- Extract more fields from your phone bill
- Provide better formatted text for the LLM
- Give more accurate field names and values
- Show better section identification

Let me know if you still have issues with the field extraction!

User

add two commands in @Makefile to start the FastAPI backend and run the app

---

AI (Composer Assistant)

I'll create a Makefile with commands to start both the FastAPI backend and the Streamlit frontend:

AI (Composer Assistant)

AI (Composer Assistant)

I've created a Makefile with three targets:

- `backend`: Starts the FastAPI server
  - Uses uvicorn with hot reload
  - Binds to all interfaces (0.0.0.0)
  - Runs on port 8000
- `frontend`: Starts the Streamlit frontend
  - Runs the Streamlit app

  - Uses port 8501
- `run`: Attempts to run both servers (note: this might not work as expected since it will try to run them sequentially)

You can use these commands as follows:

```
# To start just the backend
make backend

# To start just the frontend
make frontend

# To start both (in separate terminals)
make backend  # In one terminal
make frontend  # In another terminal
```

Note: You'll need to run the backend and frontend in separate terminals since they both need to keep running. The `run` target is included but won't work as expected since it will try to run them sequentially (the second command won't start until the first one finishes).

Let me know if you'd like me to add any additional commands or modify the existing ones!

User

I am getting "Error uploading documents: Internal Server Error" while trying to upload a single, short PDF document.

AI (Composer Assistant)

Let's debug the server error by adding better error handling and logging in the FastAPI backend. I'll modify the `app/main.py` file to provide more detailed error information:

AI (Composer Assistant)

AI (Composer Assistant)

Now let's also add better error handling to the extraction service. I'll modify the `app/services/extraction_service.py`:

AI (Composer Assistant)

AI (Composer Assistant)

I've added comprehensive error handling and logging to both the FastAPI backend and the extraction service. The changes include:

- Added detailed logging throughout the process
- Added proper exception handling with stack traces
- Improved error messages
- Added cleanup in finally blocks
- Added more detailed status logging

Now when you try to upload a document, you should see detailed logs in the terminal where you're running the FastAPI server. This will help us identify exactly where the error is occurring.

Try uploading your document again and check the terminal output. The logs should show:

- When the file is received
- PDF validation results
- Text extraction progress
- Field extraction progress
- Any errors that occur

Could you try uploading the document again and share the error logs that appear in the terminal? This will help us pinpoint the exact issue.

Also, make sure:

- The PDF file is not corrupted
- The file size is under 10MB (as configured in settings)
- The PDF contains readable text (not scanned images)
- You have a valid OpenAI API key set in your environment variables

Let me know what the logs show and we can fix the specific issue!

User

I am still getting the same error; here's the full error message:

"""

```
INFO:     127.0.0.1:33458 - "POST /api/upload HTTP/1.1" 500 Internal Server Error
ERROR:    Exception in ASGI application
Traceback (most recent call last):
```

```
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/uvicorn/protocols/
http/h11_impl.py", line 403, in run_asgi
result = await app(  # type: ignore[func-returns-value]
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/uvicorn/
middleware/proxy_headers.py", line 60, in __call__
return await self.app(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/fastapi/
applications.py", line 1054, in __call__
await super().__call__(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
applications.py", line 112, in __call__
await self.middleware_stack(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
middleware/errors.py", line 187, in __call__
raise exc
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
middleware/errors.py", line 165, in __call__
await self.app(scope, receive, _send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
middleware/cors.py", line 85, in __call__
await self.app(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
middleware/exceptions.py", line 62, in __call__
await wrap_app_handling_exceptions(self.app, conn)(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
_exception_handler.py", line 53, in wrapped_app
raise exc
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
_exception_handler.py", line 42, in wrapped_app
await app(scope, receive, sender)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
routing.py", line 714, in __call__
await self.middleware_stack(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
routing.py", line 734, in app
await route.handle(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
routing.py", line 288, in handle
await self.app(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
routing.py", line 76, in app
await wrap_app_handling_exceptions(app, request)(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
_exception_handler.py", line 53, in wrapped_app
raise exc
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
_exception_handler.py", line 42, in wrapped_app
await app(scope, receive, sender)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
routing.py", line 73, in app
response = await f(request)
```

```
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/fastapi/routing.py",
line 327, in app
content = await serialize_response(
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/fastapi/routing.py",
line 176, in serialize_response
raise ResponseValidationError(
fastapi.exceptions.ResponseValidationError: 27 validation errors:
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 0, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb12a0>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 1, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb2680>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 2, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb3250>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 3, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb33a0>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 4, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb08e0>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 5, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb08b0>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 6, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb0850>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 7, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb0820>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 8, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb07c0>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 9, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb0730>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 10, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb0d30>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 11, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb0d90>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 12, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb0dc0>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 13, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb0a00>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 14, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb0a30>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 15, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb0af0>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 16, 'bounding_box'), 'msg': 'Field required',
'input': <app.database.models.ExtractedField object at 0x7f259aeb0b20>}
"""
```

AI (Composer Assistant)

The error is occurring because there's a mismatch between our database models and Pydantic schemas. The error shows that the `bounding_box` field is missing in the response. Let's fix this by updating both the database models and schemas.

First, let's update the database models:

AI (Composer Assistant)

AI (Composer Assistant)

Now let's update the Pydantic schemas:

AI (Composer Assistant)

AI (Composer Assistant)

Now let's update the FastAPI endpoint to properly handle the bounding box fields:

AI (Composer Assistant)

AI (Composer Assistant)

The changes I've made:

- Updated the database model (`models.py`) to store bounding box coordinates as separate float columns instead of a JSON field.
- Updated the FastAPI endpoint (`main.py`) to properly map the bounding box coordinates from the Pydantic model to the database columns.
- The Pydantic schemas (`document.py`) were already correct, so no changes were needed there.

Now you'll need to:

- Delete the existing database file (if it exists):

```
rm doc_parser.db
```

- Restart the FastAPI server:

```
uvicorn app.main:app --reload
```

The error should be resolved now. The application will:

- Store bounding box coordinates as separate float columns in the database
- Properly convert between the database model and Pydantic schemas
- Handle the response validation correctly

Try uploading your PDF document again. Let me know if you encounter any other issues!

User

getting these similar error messages again:

"""

```
INFO:     127.0.0.1:48898 - "POST /api/upload HTTP/1.1" 500 Internal Server Error
ERROR:    Exception in ASGI application
Traceback (most recent call last):
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/uvicorn/protocols/
http/h11_impl.py", line 403, in run_asgi
result = await app(  # type: ignore[func-returns-value]
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/uvicorn/
middleware/proxy_headers.py", line 60, in __call__
return await self.app(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/fastapi/
applications.py", line 1054, in __call__
await super().__call__(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
applications.py", line 112, in __call__
await self.middleware_stack(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
middleware/errors.py", line 187, in __call__
raise exc
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
middleware/errors.py", line 165, in __call__
await self.app(scope, receive, _send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
middleware/cors.py", line 85, in __call__
await self.app(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
middleware/exceptions.py", line 62, in __call__
await wrap_app_handling_exceptions(self.app, conn)(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
_exception_handler.py", line 53, in wrapped_app
raise exc
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/
```

```
_exception_handler.py", line 42, in wrapped_app
await app(scope, receive, sender)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/routing.py", line 714, in __call__
await self.middleware_stack(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/routing.py", line 734, in app
await route.handle(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/routing.py", line 288, in handle
await self.app(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/routing.py", line 76, in app
await wrap_app_handling_exceptions(app, request)(scope, receive, send)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/_exception_handler.py", line 53, in wrapped_app
raise exc
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/_exception_handler.py", line 42, in wrapped_app
await app(scope, receive, sender)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/starlette/routing.py", line 73, in app
response = await f(request)
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/fastapi/routing.py", line 327, in app
content = await serialize_response(
File "/home/vishal/code/citi/doc-parser/.venv/lib/python3.10/site-packages/fastapi/routing.py", line 176, in serialize_response
raise ResponseValidationError(
fastapi.exceptions.ResponseValidationError: 29 validation errors:
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 0, 'bounding_box'), 'msg': 'Field required', 'input': <app.database.models.ExtractedField object at 0x7fed83f7cbb0>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 1, 'bounding_box'), 'msg': 'Field required', 'input': <app.database.models.ExtractedField object at 0x7fed83f7e0b0>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 2, 'bounding_box'), 'msg': 'Field required', 'input': <app.database.models.ExtractedField object at 0x7fed83f7f760>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 3, 'bounding_box'), 'msg': 'Field required', 'input': <app.database.models.ExtractedField object at 0x7fed83f7dae0>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 4, 'bounding_box'), 'msg': 'Field required', 'input': <app.database.models.ExtractedField object at 0x7fed83f7da80>}
{'type': 'missing', 'loc': ('response', 0, 'extracted_fields', 5, 'bounding_box'), 'msg': 'Field required', 'input': <app.database.models.ExtractedField object at 0x7fed83f7da20>}
"""
```

It looks like whatever you did, didn't quite fix the issue. Please be more attentive this time and fix the issue carefully.

AI (Composer Assistant)

The issue is that we need to modify the Pydantic model to properly handle the database model's separate bounding box fields. Let's update the `ExtractedField` model in `app/schemas/document.py`:

AI (Composer Assistant)

AI (Composer Assistant)

The key changes I made:

- Changed `ExtractedField` to directly match the database model structure with separate bounding box fields
- Added a `bounding_box` property that constructs the `BoundingBox` object from the individual fields
- Made the model inherit from `BaseModel` instead of `ExtractedFieldBase`

Now let's also update the `get_document_fields` endpoint in `app/main.py` to ensure it returns the correct format:

AI (Composer Assistant)

AI (Composer Assistant)

Now let's delete the existing database and restart the application:

- Delete the database:

```
rm doc_parser.db
```

- Restart the FastAPI server:

```
uvicorn app.main:app --reload
```

The changes I made:

- Updated the `ExtractedField` Pydantic model to match the database structure exactly
- Added a `bounding_box` property to construct the `BoundingBox` object from the individual

  fields
- Made sure the model properly handles the conversion between database and API formats

The error should now be resolved because:

- The database model and Pydantic model structures match exactly
- The `bounding_box` property provides the required field in the API response
- The model properly handles the conversion between database and API formats

Try uploading your PDF document again. Let me know if you encounter any other issues!

User

great, this is working now. Is there a way to change the Streamlit app UI to make the screen wider? The key-value table on the right side of the PDF document is getting wrapped/squished due to limited width.

AI (Composer Assistant)

Yes, we can modify the Streamlit app to use a wider layout and better column proportions. Let's update the `frontend/streamlit_app.py` file:

AI (Composer Assistant)

AI (Composer Assistant)

I've made several improvements to the Streamlit UI:

- Added wide layout configuration:

```
st.set_page_config(
    layout="wide",
    page_title="Document Parser",
    page_icon="⊠ ",
    initial_sidebar_state="expanded"
)
```

- Added custom CSS for better styling:

- Increased padding around the main content
- Made buttons full width
- Added better styling for expanders
- Adjusted table font size

- Changed column proportions:

- Changed from `[2, 1]` to `[3, 2]` for better space utilization
- This gives more space to both the PDF viewer and the fields table

- Made document expanders expanded by default:

```
with st.expander(f"Document: {doc['filename']}", expanded=True):
```

- Added better visual hierarchy with styling and spacing

To see these changes:

- Stop the current Streamlit server
- Start it again:

```
streamlit run frontend/streamlit_app.py
```

The UI should now be:

- Wider overall
- Better proportioned between PDF viewer and fields table
- More visually appealing with better spacing and styling
- More readable with adjusted font sizes

Let me know if you'd like any adjustments to the layout or styling!

User

Did you change the color scheme? I am using dark theme and now the fields table has white background as well as white fonts, so I can't see the values. Please fix this.

AI (Composer Assistant)

I'll update the Streamlit UI to better handle dark theme and ensure the table is readable. Let's modify the `frontend/streamlit_app.py` file:

AI (Composer Assistant)

AI (Composer Assistant)

I apologize for the error. Let me try again with a more focused edit:

AI (Composer Assistant)

AI (Composer Assistant)

I've updated the Streamlit UI with the following changes to better handle dark theme:

- Removed hardcoded background colors and replaced them with Streamlit's CSS variables:
  - \`var(--background-color)\` for main backgrounds
  - \`var(--secondary-background-color)\` for hover states and headers
  - \`var(--text-color)\` for text colors
- Added specific styling for table elements:
  - Table headers now use the secondary background color
  - Table cells use the main background color
  - Text colors adapt to the theme
  - Added hover effect for table rows
- Made the expander background color theme-aware

These changes will ensure that:

- The table is readable in both light and dark themes
- Text colors automatically adjust based on the theme
- The UI maintains visual hierarchy and contrast
- Interactive elements (like hover states) are visible in both themes

Please restart your Streamlit app to see the changes. The table should now be clearly visible in dark theme with proper contrast between text and background colors.

User

fix this warning please:

`The use_column_width parameter has been deprecated and will be removed in a future release. Please utilize the use_container_width parameter instead.`

AI (Composer Assistant)

I'll update the `display_pdf_page` function in `frontend/streamlit_app.py` to use the recommended `use_container_width` parameter instead of the deprecated `use_column_width`:

AI (Composer Assistant)

AI (Composer Assistant)

I've updated the code to use `use_container_width=True` instead of the deprecated `use_column_width=True`. This change will remove the deprecation warning while maintaining the same functionality of making the image fill the width of its container. The image will still scale appropriately within the column layout we've set up.

User

reduce the width for the pdf viewer so that we can give some more space to the fields table

---

AI (Composer Assistant)

I'll modify the column proportions in the Streamlit app to give more space to the fields table. I'll update the ratio from 3:2 to 2:3 in favor of the fields table:

AI (Composer Assistant)

AI (Composer Assistant)

I've changed the column proportions from `[3, 2]` to `[2, 3]`, which means:

- The PDF viewer (left column) now takes up 40% of the width
- The fields table (right column) now takes up 60% of the width

This change will give more space to the fields table while still maintaining a good size for the PDF viewer. The fields should now be more readable and have less wrapping issues. Please restart your Streamlit app to see the changes.

User

I am seeing only the first page of the PDF document. I would like iterate through each page one by one.

AI (Composer Assistant)

I'll modify the Streamlit app to add page navigation controls and display each page of the PDF document. I'll update the `frontend/streamlit_app.py` file:

AI (Composer Assistant)

AI (Composer Assistant)

I've made several changes to add page navigation functionality:

- Added page navigation controls:
  - Previous/Next page buttons
  - Current page indicator showing "Page X of Y"
  - Buttons are only shown when applicable (Previous button hidden on first page, Next button hidden on last page)
- Added session state management:
  - Initialized `current_page` in session state to track the current page number
  - Updated the page number when navigation buttons are clicked
- Updated the `display_pdf_page` function:
  - Added `total_pages` parameter to know the total number of pages
  - Added navigation controls below the PDF display
  - Fixed the deprecated `use_column_width` parameter to `use_container_width`
- Updated the main function to pass the current page and total pages to the display function

Now you can:

- Navigate through pages using the Previous/Next buttons
- See which page you're currently viewing and the total number of pages
- The page navigation state is maintained as you interact with the app

Please restart your Streamlit app to see these changes. You should now be able to navigate through all pages of your PDF document.

User

I see the Next Page button on the app, but when I click it, the second page is not showing up. Fix this please, and also add a Previous Page button to go back to the previous page.

AI (Composer Assistant)

I notice that we're always using `uploaded_files[0]` to display the PDF, which means we're only showing the first uploaded file regardless of which document we're viewing. Let's fix this and ensure the page navigation works correctly:

AI (Composer Assistant)

AI (Composer Assistant)

I've made several important changes to fix the page navigation and document display:

- Added document tracking:
  - Added `current_doc_index` to session state to track which document is currently being viewed
  - The expander is now only expanded for the current document
- Fixed PDF file selection:
  - Changed `uploaded_files[0]` to `uploaded_files[idx]` to display the correct PDF file for each document
  - This ensures that each document shows its own PDF content instead of always showing the first uploaded file
- Improved document organization:
  - Added enumeration to track document indices
  - Each document's expander is now controlled by the current document index

The page navigation should now work correctly because:

- Each document is properly tracked
- The correct PDF file is being displayed for each document
- The page state is maintained per document

Please restart your Streamlit app to see these changes. You should now be able to:

- Navigate through pages using the Previous/Next buttons
- See the correct PDF content for each document
- Switch between documents while maintaining the correct page number for each

Let me know if you still experience any issues with the page navigation.

User

I still don't see the second page when I click on the Next Page button. It looks like the PDF viewer as well as the fields table are just disappearing when I click on the Next Page button. Please fix this.

AI (Composer Assistant)

I see the issue. The problem is that we're using a single `current_page` state for all documents, and the page navigation is causing the entire app to rerun. Let's modify the code to maintain page numbers per document and fix the navigation:

AI (Composer Assistant)

AI (Composer Assistant)

I've made several important changes to fix the page navigation:

- Document-specific page tracking:
  - Removed the global `current_page` state
  - Added document-specific page tracking using `st.session_state[f"page_{doc['id']}"]`
  - Each document now maintains its own page number independently
- Unique button keys:
  - Added unique keys to the Previous/Next buttons using the document ID
  - This prevents button conflicts between different documents
- Page initialization:
  - Added initialization of page numbers for each document when they're first processed
  - Each document starts at page 1
- Updated the `display_pdf_page` function:
  - Added `doc_id` parameter to identify which document is being displayed
  - Updated the page navigation to use document-specific page numbers

These changes should fix the issues where:

- The PDF viewer was disappearing on page navigation
- Page numbers were being shared between documents
- Navigation wasn't working properly

Please restart your Streamlit app to see these changes. You should now be able to:

- Navigate through pages for each document independently
- See the correct page content when clicking Next/Previous
- Maintain different page numbers for different documents

The page navigation should now work smoothly without any disappearing content.

User

Nope, it's still not working. Please think about his carefully. It looks like you are trying to fix something different. Again, the issue is that when I click on the Next Page button, it doesn't show *any* page on the left, and the fields table on the right disappears too.

AI (Composer Assistant)

I understand the issue now. The problem is that when we click the Next/Previous buttons, Streamlit is rerunning the entire app and losing the uploaded files. We need to store the uploaded files in the session state to persist them between reruns. Let me fix this:

AI (Composer Assistant)

AI (Composer Assistant)

I've made several important changes to fix the page navigation issue:

- Added session state storage for uploaded files and processed documents:
  ```python
  if 'uploaded_files' not in st.session_state:
  st.session_state.uploaded_files = None
  if 'processed_documents' not in st.session_state:
  st.session_state.processed_documents = None
  ```
- Store uploaded files in session state when they're first uploaded:
  ```python
  if uploaded_files:
  st.session_state.uploaded_files = uploaded_files
  ```
- Store processed documents in session state after successful processing:
  ```python
  if documents:
  st.session_state.processed_documents = documents
  ```
- Added a new section to display already processed documents when the "Process Documents" button hasn't been clicked again:
  ```python
  elif st.session_state.processed_documents:
  # Display already processed documents
  ```

```
for idx, doc in enumerate(st.session_state.processed_documents):
# ... display logic ...
```

- Use session state for all file operations:
  ```python
  display_pdf_page(
  st.session_state.uploaded_files[idx].getvalue(),
  doc['id'],
  st.session_state[f"page_{doc['id']}"],
  doc['total_pages']
  )
  ```

These changes ensure that:

- The uploaded files persist between page navigation
- The processed documents remain available
- The page navigation state is maintained
- The UI elements (PDF viewer and fields table) remain visible during navigation

Please restart your Streamlit app to see these changes. The page navigation should now work correctly without losing any content.
