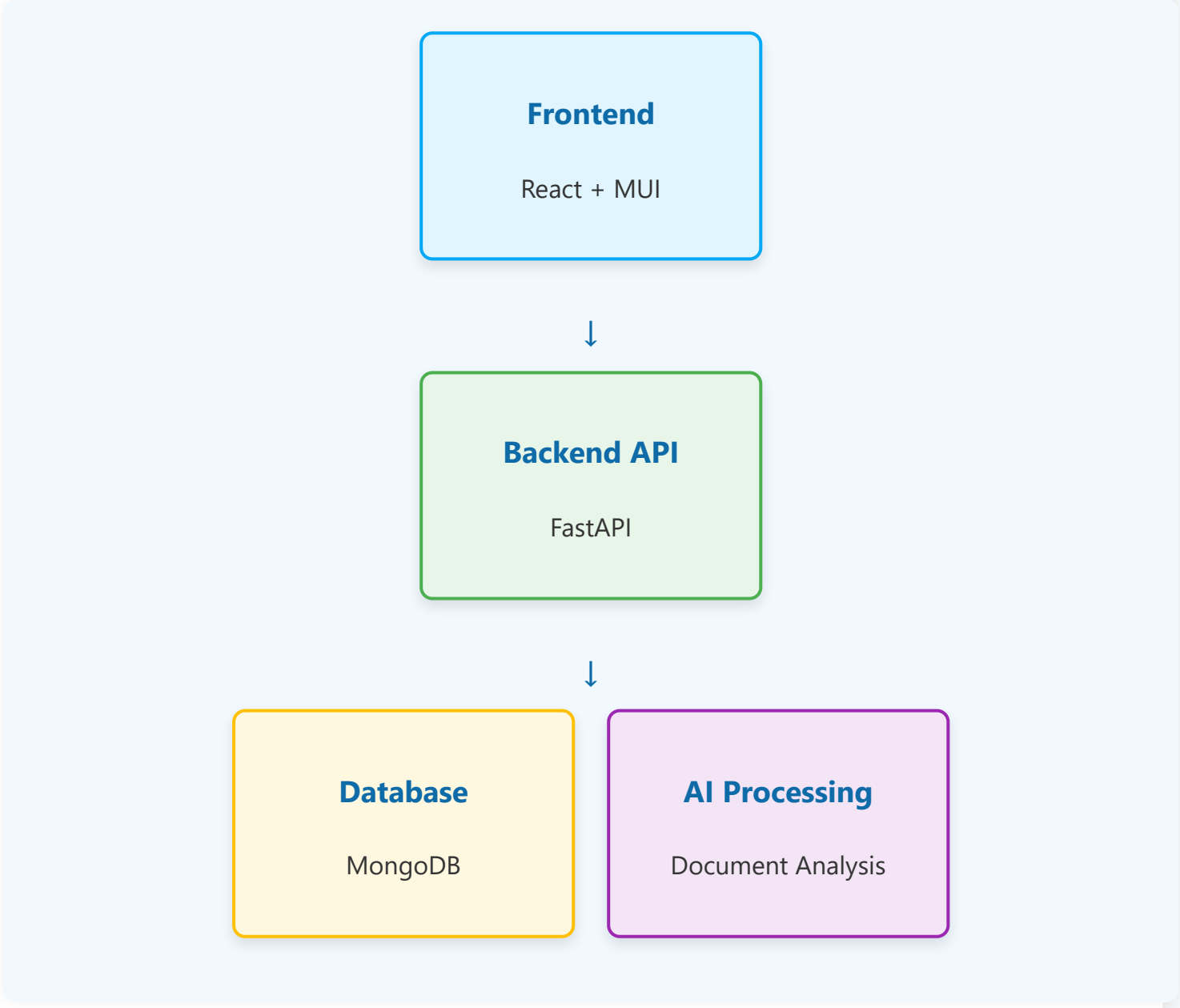
DocuMorph AI - Project Techniques & Architecture

System Architecture

DocuMorph AI follows a modern three-tier architecture with dedicated AI components to provide document transformation capabilities.



Key Components

Frontend

Backend

- React 19 for UI components
- Material UI 7 for design system
- React Router 7 for navigation
- Axios for API requests
- Context API for state management

- FastAPI for RESTful API endpoints
- Uvicorn ASGI server
- JWT authentication
- Motor for async MongoDB operations
- Pydantic for data validation

Database

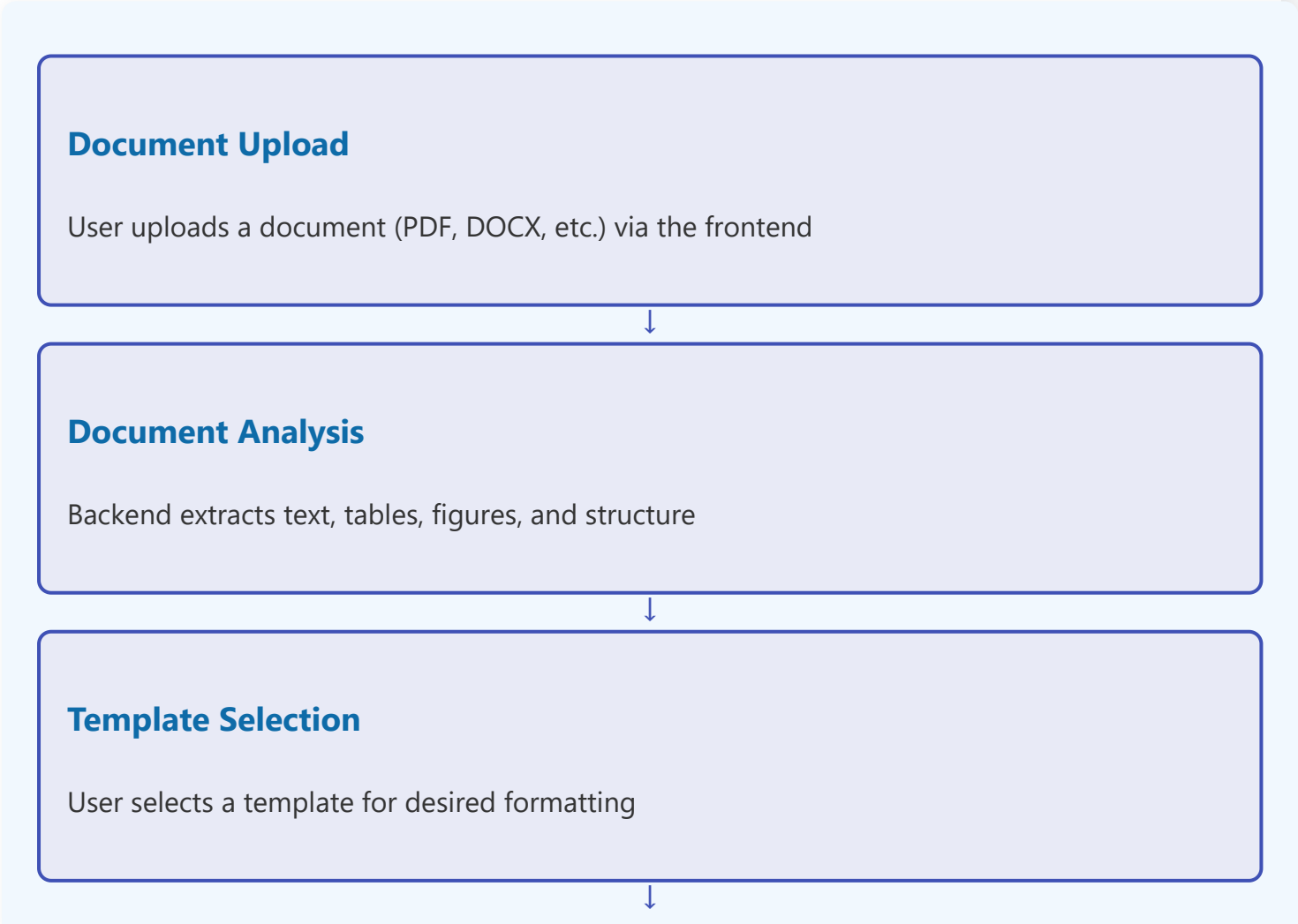
- MongoDB for document storage
- AsyncIO for non-blocking database operations
- Atlas for cloud hosting (optional)

Document Processing

- pdfplumber for PDF text extraction
- OpenPyXL for Excel generation
- python-docx for Word document manipulation
- Computer Vision for layout detection

Document Processing Workflow

The document processing workflow involves several steps from upload to formatted output:



Content Transformation

Backend applies template rules to document content

↓

Preview & Download

User previews and downloads the transformed document

API Authentication Flow

The system uses JWT (JSON Web Tokens) for stateless authentication:

Step	Process	Implementation
1. Registration	User creates account with email/password	Hashed using bcrypt before storage
2. Login	User provides credentials, server validates	OAuth2PasswordRequestForm from FastAPI
3. JWT Generation	Server creates signed token with user data	PyJWT with HS256 algorithm
4. Authentication	Token sent in Authorization header	Bearer token pattern
5. Token Validation	Server validates and extracts user info	FastAPI Dependency Injection

Table Extraction Implementation

A key feature of DocuMorph AI is the ability to extract and manipulate tables from PDF documents. The implementation uses pdfplumber and pandas:

```
def extract_clean_tables_from_pdf(pdf_file):
    """
    Extract tables from a PDF and return a list of cleaned DataFrames.
    """
    tables = []
    with pdfplumber.open(pdf_file) as pdf:
        for page in pdf.pages:
            extracted = page.extract_tables()
            if extracted:
                for table in extracted:
                    df = pd.DataFrame(table[1:], columns=table[0])
                    df = clean_column_names(df)
                    tables.append(df)
    return tables
```

This approach allows for:

- Extraction of tabular data from PDF documents
- Conversion to structured DataFrame objects
- Data cleaning and normalization
- Export to various formats (Excel, CSV, etc.)

Frontend-Backend Communication

Communication between frontend and backend is handled through RESTful API calls:

```
// Frontend Axios Request
export const uploadDocument = async (file) => {
  try {
    const formData = new FormData();
    formData.append('file', file);

    const response = await axios.post('/api/upload', formData, {
      headers: {
        'Content-Type': 'multipart/form-data',
      },
    });

    return response.data;
  } catch (error) {
    console.error('Error uploading document:', error);
    throw error;
  }
};

# Backend FastAPI Endpoint
```

```
@app.post("/api/upload")
async def upload_document(
    file: UploadFile = File(...),
    background_tasks: BackgroundTasks = None,
    current_user = Depends(get_optional_user)
):
    # Implementation details...
```

Database Schema

MongoDB collections used in the project:

Collection	Purpose	Key Fields
users	Store user information	id, email, hashed_password, name, tier
documents	Track uploaded and processed documents	id, original_name, upload_path, status, user_id
templates	Store document templates	id, name, description, category, formatting

Deployment Architecture

The application is designed for flexible deployment:

Development Environment

- Local development server
- MongoDB running locally or in Atlas
- Hot-reloading enabled
- Debug logging

Production Environment

- Containerized using Docker
- Frontend served as static files
- Backend running with Gunicorn/Uvicorn
- MongoDB Atlas for database