Project 5: Intelligent Chat System with File Analysis

Objective (Why?)

Build an intelligent chat system that extends your Project 4 authentication system with file upload and AI-powered analysis capabilities. This project transforms your secure user management platform into an interactive AI assistant that can analyze documents and data files. You will practice:

- Full-Stack Extension: Building upon your existing FastAPI + React + PostgreSQL architecture
- File Processing: Handle multiple file formats (PDF, DOCX, TXT, Excel) with secure upload
- Direct LLM Integration: Send file content + user prompts to OpenAI/Claude for analysis
- Role-Based File Access: Extend your RBAC system to include file upload permissions
- Chat Interface: Build modern chat UI similar to ChatGPT/Claude with file attachment support

Project Progression from Project 4

What You're Building Upon:

- Authentication System: User registration, login, JWT tokens, social auth
- Authorization System: Role-based access control (user/admin roles)
- Database Foundation: PostgreSQL with users, sessions, social accounts
- Frontend Foundation: React + Vite + Tailwind CSS with protected routes
- Backend Foundation: FastAPI with secure endpoints and middleware

What You're Adding in Project 5:

- Chat System: Multi-threaded conversations with message history
- File Upload: Secure file handling with role-based permissions
- AI Integration: Direct LLM analysis of uploaded files
- Document Processing: Text extraction from multiple file formats

- Enhanced UI: ChatGPT-like interface with file attachments

Core Requirements (Must-have)

Part 1: Extended Chat System (Building on Project 4)

Component	Requirement
User Authentication	Use existing Project 4 auth system - JWT tokens, role validation
Chat Management	Create new chat threads, manage conversations, persistent chat history
File Upload Permissions	Admin users can upload files (extending existing RBAC)
Database Extension	Add chat/message tables to existing Project 4 schema
Protected Chat Routes	Extend existing route protection to include chat features

Part 2: File Analysis Capabilities

Component	Requirement
Secure File Upload	Admin-only file upload with validation (extending Project 4 permissions)
Document Processing	Extract text from PDF, DOCX, TXT files
Excel Analysis	Parse Excel files and convert data to readable format
LLM Integration	Send file content + user prompt directly to OpenAI/Claude

Role-Based Responses	All users can chat, only admins can upload files

Milestone 1: Chat System Integration

Deliverables:

- Extend Project 4 database schema with chat tables
- Create chat API endpoints using existing auth middleware
- Build chat UI components in existing React app
- Implement role-based chat permissions (users can chat, admins can upload)

Database Migration (Extending Project 4):

```
SQL
    -- Add to existing users table from Project 4
    -- (users, user_sessions, social_accounts tables already exist)

    -- New chat tables
    CREATE TABLE chat_threads (
      id SERIAL PRIMARY KEY,
      user_id INTEGER REFERENCES users(id) NOT NULL,
      title VARCHAR(200),
      created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    );

    CREATE TABLE messages (
      id SERIAL PRIMARY KEY,
      thread_id INTEGER REFERENCES chat_threads(id) NOT NULL,
      content TEXT NOT NULL,
      role VARCHAR(20) CHECK (role IN ('user', 'assistant')) NOT NULL,
      has_file BOOLEAN DEFAULT FALSE,
```

```
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE uploaded_files (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) NOT NULL,
    thread_id INTEGER REFERENCES chat_threads(id),
    message_id INTEGER REFERENCES messages(id),
    filename VARCHAR(255) NOT NULL,
    file_type VARCHAR(20) NOT NULL,
    file_size INTEGER NOT NULL,
    file_path VARCHAR(500) NOT NULL,
    processed_content TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Milestone 2: File Processing & LLM Integration

Deliverables:

- File upload endpoint (admin-only, using existing RBAC)
- Text extraction from PDF, DOCX, TXT files
- Excel data parsing and formatting
- OpenAI/Claude API integration
- Enhanced chat interface with file attachment support

Milestone 3: Production Features

Deliverables:

- File management interface for admins
- Error handling and user feedback

- Performance optimization
- Testing with existing authentication
- Deployment alongside Project 4

Technical Implementation (Extending Project 4)

API Endpoints (Adding to Project 4)

Python

```
# Existing Project 4 endpoints remain unchanged:
# POST /auth/register, POST /auth/login, GET /auth/me, etc.

# New chat endpoints (using existing auth middleware):
@app.get("/api/chat/threads")
@require_auth  # Use existing Project 4 auth decorator
async def get_user_threads(current_user: User = Depends(get_current_user)):
    # Get chat threads for authenticated user

@app.post("/api/chat/threads")
@require_auth
async def create_thread(current_user: User = Depends(get_current_user)):
    # Create new chat thread for authenticated user

@app.post("/api/chat/threads/{thread_id}/messages")
@require_auth
async def send_message(
    thread_id: int,
    message: str,
    current_user: User = Depends(get_current_user)
):
    # Send message in thread (all authenticated users)
```

```
@app.post("/api/files/upload")
@require_admin  # Use existing Project 4 admin decorator
async def upload_file(
    file: UploadFile,
    current_user: User = Depends(get_current_admin)
):
    # Admin-only file upload (extending existing RBAC)
```

Frontend Integration (Extending Project 4 React App)

None

```
// Extend existing Project 4 AuthContext
const AuthContext = createContext();

// Add chat functionality to existing app
function ChatInterface() {
    const { user, isAdmin } = useAuth(); // Use existing auth context

    return (
        <div className="chat-container">
            <MessageList />
            <MessageInput />
            {isAdmin && <FileUploadButton />} {/* Only show to admins */}
        </div>
    );
}

// Add to existing router in App.jsx
function App() {
```

```
  return (
    <Routes>
      {/* Existing Project 4 routes */}
      <Route path="/login" element={<Login />} />
      <Route path="/dashboard" element={<Dashboard />} />

      {/* New Project 5 routes */}
      <Route
        path="/chat"
        element={
          <ProtectedRoute> {/* Use existing route protection */}
            <ChatInterface />
          </ProtectedRoute>
        }
      />
    </Routes>
  );
}
```

File Processing Implementation

Python

```
# Simple file processors (new for Project 5)
class FileProcessor:
  def process_file(self, file_path, file_type):
    if file_type == 'pdf':
      return self.extract_pdf_text(file_path)
    elif file_type == 'docx':
      return self.extract_docx_text(file_path)
    elif file_type == 'txt':
```

```
            return self.extract_txt_content(file_path)
        elif file_type in ['xlsx', 'xls']:
            return self.extract_excel_data(file_path)

    def extract_pdf_text(self, file_path):
        # PyPDF2 implementation
        pass

    def extract_docx_text(self, file_path):
        # python-docx implementation
        pass

# LLM integration (new for Project 5)
class LLMAnalyzer:
    def analyze_with_file(self, user_prompt, file_content, file_type):
        enhanced_prompt = f"""
        User has uploaded a {file_type} file with the following content:

        === FILE CONTENT START ===
        {file_content}
        === FILE CONTENT END ===

        User Question: {user_prompt}

        Please analyze the file content and answer the user's question.
        """
        return self.call_openai(enhanced_prompt)
```

Sample User Workflows

Admin User Workflow (Can Upload Files)

None

1. Admin logs in using Project 4 auth system
2. Navigates to new chat interface
3. Uploads contract.pdf (admin-only permission)
4. Asks: "What are the key terms in this contract?"
5. System: Extracts PDF text + sends to LLM
6. Receives AI analysis of contract terms

Regular User Workflow (Can Chat Only)

None

1. User logs in using Project 4 auth system
2. Navigates to chat interface
3. Can view and chat in existing threads
4. Cannot see file upload button (role-based UI)
5. Can ask questions about files uploaded by admins

Role-Based Permissions (Extending Project 4 RBAC)

File Upload Permissions

- Admin Users: Can upload all supported file types (PDF, DOCX, TXT, Excel)
- Regular Users: Cannot upload files (UI hidden, API returns 403)
- All Users: Can participate in chat conversations
- All Users: Can view AI responses to file analysis

Dependencies (Adding to Project 4)

New Backend Dependencies

```
None
# Add to existing Project 4 requirements.txt:
PyPDF2==3.0.1          # PDF text extraction
python-docx==0.8.11     # DOCX text extraction
pandas==2.1.3          # Excel processing
openpyxl==3.1.2         # Excel file reading
openai==1.3.5          # OpenAI API integration
python-multipart==0.0.6 # File upload handling
```

New Frontend Dependencies

```
JSON
// Add to existing Project 4 package.json:
{
  "dependencies": {
    // ...existing Project 4 dependencies...
    "react-dropzone": "^14.2.3",
    "lucide-react": "^0.294.0"
  }
}
```

Stretch Goals (Optional - For Advanced Students)

Level 1: Basic RAG Implementation

- Vector Embeddings: Store document chunks in vector database
- Semantic Search: Find relevant document sections for user queries
- Source Attribution: Show which parts of documents were used in responses

Level 2: Advanced Features

- Multi-Document Chat: Handle multiple files in single conversation
- File Management: Admin interface for organizing uploaded files
- Export Functionality: Download chat conversations and analysis results

Level 3: Intelligence Enhancements

- Context Memory: Remember previous conversations and file content
- Smart Summarization: Automatic document summarization
- Data Visualization: Generate charts from Excel data analysis

File Analysis Success

- Admin users can upload PDF, DOCX, TXT, Excel files
- Regular users cannot access file upload (proper 403 handling)
- File content extracted accurately from all supported formats
- LLM provides relevant analysis based on file content and user questions

Role-Based Access Success

- File upload permissions properly enforced (admin-only)
- UI conditionally shows/hides features based on user role
- All users can participate in chat regardless of upload permissions
- Existing Project 4 RBAC system extended seamlessly

Advanced Features (Stretch Goals)

Document Intelligence Enhancements

- Multi-Language Support: Process documents in various languages
- Visual Document Processing: Handle images and charts within documents
- Document Comparison: Side-by-side analysis of multiple files
- Version Tracking: Track document changes and revisions

Data Analysis Enhancements

- Real-Time Data Connection: Connect to databases and APIs

- Advanced Statistical Analysis: Regression, correlation, forecasting
- Custom Visualization Types: Industry-specific chart types
- Automated Report Generation: Scheduled analysis and reporting

Intelligence Enhancements

- Learning from Interactions: Improve query classification based on feedback
- Advanced Context Management: Better handling of long conversations
- Multi-User Collaboration: Shared document and data analysis workspaces
- Template Queries: Pre-built query templates for common business questions

Deliverables

1. GitHub Repository with complete integrated platform
2. Live Demo showing document and data analysis capabilities
3. PLATFORM_DEMO.md - Include:
    - Screenshots of document analysis workflows
    - Screenshots of Excel data analysis with visualizations
    - Screenshots of cross-modal query handling
    - Sample files and analysis examples
    - Performance metrics and benchmarks
4. INTEGRATION_ARCHITECTURE.md - Include:
    - System architecture diagrams
    - Database schema documentation
    - API endpoint documentation
    - RAG and data processing architecture
5. Technical_Learnings.md - Include:
    - RAG implementation insights
    - Natural language to code generation learnings
    - Cross-modal query handling experience
    - Multi-format processing challenges

Performance Requirements

Document Processing Performance

- Document upload and parsing: < 30 seconds for files up to 10MB
- Vector embedding generation: < 15 seconds for typical documents
- Semantic search response: < 2 seconds for document queries

Data Analysis Performance

- Excel file processing: < 20 seconds for typical business spreadsheets
- NL-to-code generation: < 5 seconds for standard business queries
- Chart generation and rendering: < 3 seconds for interactive visualizations

Cross-Modal Performance

- Task classification: < 1 second for query type detection
- Cross-modal response generation: < 5 seconds for complex queries
- Context switching: < 2 seconds between document and data modes

Success Criteria Checklist

Document Intelligence Success

- Multi-format document processing (PDF, DOCX, TXT) working
- RAG system providing accurate context-aware responses
- Source attribution and citation functionality
- Multi-document conversation support
- Vector search performance meets requirements

Data Analysis Success

- Excel files upload and parse correctly with multi-sheet support
- Natural language questions convert to accurate Pandas operations
- Query results display in appropriate text and visual formats
- Charts automatically select correct visualization types
- Export functionality works for multiple formats

Cross-Modal Intelligence Success

- System correctly detects document vs. data analysis tasks
- Cross-modal queries handled with unified responses
- Context maintained across different query types
- Seamless user experience across document and data analysis
- Performance requirements met for all operations

Integration Success

- Role-based access control properly implemented
- Security measures effective for file upload and code execution
- Production-ready code quality and comprehensive testing
- Comprehensive documentation and user guides

Quick Start Resources

Document Processing & RAG

- LangChain RAG: https://python.langchain.com/docs/use_cases/question_answering
- Vector Databases: https://python.langchain.com/docs/integrations/vectorstores
- Document Loaders: https://python.langchain.com/docs/integrations/document_loaders

Data Analysis & NL-to-Code

- LangChain Pandas: https://python.langchain.com/docs/integrations/toolkits/pandas
- Plotly Python: https://plotly.com/python/
- Pandas Documentation: https://pandas.pydata.org/docs/

Task Detection & Routing

- LangChain Text Classification: https://python.langchain.com/docs/use_cases/classification
- OpenAI Function Calling: https://platform.openai.com/docs/guides/function-calling

Document Processing & RAG

- LangChain RAG: https://python.langchain.com/docs/use_cases/question_answering
- Vector Databases: https://python.langchain.com/docs/integrations/vectorstores
- Document Loaders: https://python.langchain.com/docs/integrations/document_loaders

Data Analysis & NL-to-Code

- LangChain Pandas: https://python.langchain.com/docs/integrations/toolkits/pandas
- Plotly Python: https://plotly.com/python/
- Pandas Documentation: https://pandas.pydata.org/docs/

AI Agents & Tools

- LangChain Agents: https://python.langchain.com/docs/modules/agents
- Custom Tools: https://python.langchain.com/docs/modules/agents/tools
- Function Calling: https://python.langchain.com/docs/modules/chains/additional/openai_functions