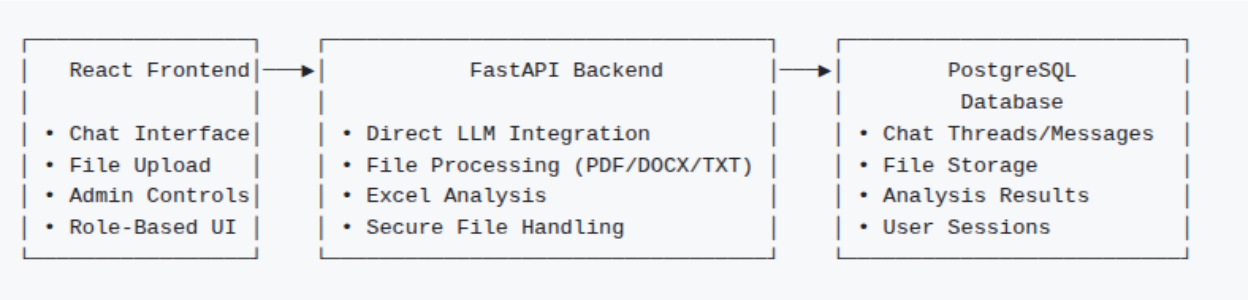
Project 5 - Intelligent Chat System with File Analysis - Technical Implementation Guide

1. Project Overview & Learning Objectives

Business Context

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 through direct LLM integration.

Architecture Overview



Core Learning Goals

- File Processing: Handle multiple file formats securely
- Direct LLM Integration: Send file content + prompts to OpenAI/Claude
- Chat System Extension: Build upon Project 4's authentication
- Role-Based File Access: Admin-only file upload permissions
- Simple AI Analysis: Direct file content analysis without complex RAG

2. Implementation Strategy

Development Approach

This guide provides step-by-step implementation building on your completed Project 4:

1. Extend existing Project 4 codebase - don't start from scratch
2. Add chat functionality to existing authentication system
3. Implement secure file upload with role-based permissions
4. Integrate direct LLM analysis for uploaded files
5. Build intuitive chat interface with file attachment support

3. Milestone 1: Chat System Integration

3.1 Database Schema Extension

Add Chat Tables to Existing Project 4 Schema

Copilot Prompt: *"Create Alembic migration to add chat functionality to existing Project 4 database. Add tables for chat_threads, messages, and uploaded_files that reference the existing users table."*

```SQL
-- filepath: /home/sivaj/projects/AI-ML/AmzurAI-BasedCodeReview/AI-Curriculum/Project-Assignments/Plan-1/V2-Plan-1-Phase-1/V2-Plan-1-Phase-1-Project-Implementations/Project-05-Technical-Implementation-Guide.md
-- Migration: Add chat tables to existing Project 4 schema
-- (users, user_sessions, social_accounts tables already exist from Project 4)

-- Chat threads for organizing conversations
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
);

-- Messages within chat threads
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
```

```
);

-- Simple file storage for uploaded files
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, -- Extracted text content
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Indexes for performance
CREATE INDEX idx_messages_thread_id ON messages(thread_id);
CREATE INDEX idx_files_user_id ON uploaded_files(user_id);
CREATE INDEX idx_files_thread_id ON uploaded_files(thread_id);
```

3.2 SQLAlchemy Models Extension

Extend Project 4 Models

Copilot Prompt: *"Add SQLAlchemy models for ChatThread, Message, and UploadedFile that work with existing Project 4 User model. Include proper relationships and validation."*

Python

```
# ...existing Project 4 models (User, UserSession, SocialAccount)...

# New chat models
class ChatThread(Base):
    __tablename__ = "chat_threads"

    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), nullable=False)
    title = Column(String(200))
    created_at = Column(DateTime, default=datetime.utcnow)
    updated_at = Column(DateTime, default=datetime.utcnow,
      onupdate=datetime.utcnow)

    # Relationships
    user = relationship("User", back_populates="chat_threads")
    messages = relationship("Message", back_populates="thread", cascade="all,
      delete-orphan")
    files = relationship("UploadedFile", back_populates="thread")

class Message(Base):
    __tablename__ = "messages"

    id = Column(Integer, primary_key=True, index=True)
    thread_id = Column(Integer, ForeignKey("chat_threads.id"), nullable=False)
    content = Column(Text, nullable=False)
    role = Column(String(20), nullable=False)  # 'user' or 'assistant'
    has_file = Column(Boolean, default=False)
    created_at = Column(DateTime, default=datetime.utcnow)

    # Relationships
    thread = relationship("ChatThread", back_populates="messages")
```

```
    files = relationship("UploadedFile", back_populates="message")

class UploadedFile(Base):
    __tablename__ = "uploaded_files"

    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), nullable=False)
    thread_id = Column(Integer, ForeignKey("chat_threads.id"))
    message_id = Column(Integer, ForeignKey("messages.id"))
    filename = Column(String(255), nullable=False)
    file_type = Column(String(20), nullable=False)
    file_size = Column(Integer, nullable=False)
    file_path = Column(String(500), nullable=False)
    processed_content = Column(Text)  # Extracted text
    created_at = Column(DateTime, default=datetime.utcnow)

    # Relationships
    user = relationship("User", back_populates="uploaded_files")
    thread = relationship("ChatThread", back_populates="files")
    message = relationship("Message", back_populates="files")

# Update existing User model to include new relationships
# Add to existing User class:
# chat_threads = relationship("ChatThread", back_populates="user")
# uploaded_files = relationship("UploadedFile", back_populates="user")
```

3.3 Basic Chat API Endpoints

Chat Management Endpoints

Copilot Prompt: *"Create FastAPI endpoints for chat management that use existing Project 4 authentication middleware. Include create thread, get threads, send message, and get messages endpoints."*

Python

```
# ...existing Project 4 auth imports and dependencies...

# New chat router
@router.post("/chat/threads")
async def create_chat_thread(
    title: str = "New Chat",
    current_user: User = Depends(get_current_user)  # Use existing Project 4 auth
):
    """Create a new chat thread for authenticated user"""
    thread = ChatThread(
        user_id=current_user.id,
        title=title
    )
    db.add(thread)
    db.commit()
    db.refresh(thread)

    return {
        "id": thread.id,
        "title": thread.title,
        "created_at": thread.created_at
    }

@router.get("/chat/threads")
async def get_user_threads(
    current_user: User = Depends(get_current_user)
):
```

```
    """Get all chat threads for authenticated user"""
    threads = db.query(ChatThread).filter(
        ChatThread.user_id == current_user.id
    ).order_by(ChatThread.updated_at.desc()).all()

    return [
        {
            "id": thread.id,
            "title": thread.title,
            "created_at": thread.created_at,
            "updated_at": thread.updated_at,
            "message_count": len(thread.messages)
        }
        for thread in threads
    ]

@router.post("/chat/threads/{thread_id}/messages")
async def send_message(
    thread_id: int,
    content: str,
    current_user: User = Depends(get_current_user)
):
    """Send a message in a chat thread"""
    # Verify thread ownership
    thread = db.query(ChatThread).filter(
        ChatThread.id == thread_id,
        ChatThread.user_id == current_user.id
    ).first()

    if not thread:
```

```
    raise HTTPException(status_code=404, detail="Thread not found")

# Save user message
user_message = Message(
    thread_id=thread_id,
    content=content,
    role="user"
)
db.add(user_message)
db.commit()

# For now, return simple echo response
# Will be replaced with LLM integration in Milestone 2
assistant_message = Message(
    thread_id=thread_id,
    content=f"Echo: {content}",
    role="assistant"
)
db.add(assistant_message)
db.commit()

return {
    "user_message": {
        "id": user_message.id,
        "content": user_message.content,
        "role": user_message.role,
        "created_at": user_message.created_at
    },
    "assistant_message": {
        "id": assistant_message.id,
```

```
            "content": assistant_message.content,
            "role": assistant_message.role,
            "created_at": assistant_message.created_at
        }
    }

@router.get("/chat/threads/{thread_id}/messages")
async def get_thread_messages(
    thread_id: int,
    current_user: User = Depends(get_current_user)
):
    """Get all messages in a chat thread"""
    # Verify thread ownership
    thread = db.query(ChatThread).filter(
        ChatThread.id == thread_id,
        ChatThread.user_id == current_user.id
    ).first()

    if not thread:
        raise HTTPException(status_code=404, detail="Thread not found")

    messages = db.query(Message).filter(
        Message.thread_id == thread_id
    ).order_by(Message.created_at.asc()).all()

    return [
        {
            "id": msg.id,
            "content": msg.content,
            "role": msg.role,
```

```
            "has_file": msg.has_file,
            "created_at": msg.created_at
        }
        for msg in messages
    ]
```

4. Milestone 2: File Processing & LLM Integration

4.1 Simple File Processing

Multi-Format File Processors

Copilot Prompt: *"Create simple file processing classes for PDF, DOCX, TXT, and Excel files. Extract text content cleanly without complex chunking. Handle errors gracefully and return structured results."*

Python

```
import PyPDF2
import docx
import pandas as pd
from pathlib import Path

class SimpleFileProcessor:
    """Simple file processor for basic text extraction"""

    def __init__(self):
        self.supported_types = {
            'application/pdf': self.process_pdf,
            'application/vnd.openxmlformats-officedocument.wordprocessingml.document':
        self.process_docx,
```

```
        'text/plain': self.process_txt,
        'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet':
    self.process_excel,
        'application/vnd.ms-excel': self.process_excel
    }

def process_file(self, file_path: str, content_type: str) -> dict:
    """Process file based on content type"""
    processor = self.supported_types.get(content_type)

    if not processor:
        raise ValueError(f"Unsupported file type: {content_type}")

    try:
        return processor(file_path)
    except Exception as e:
        return {
            "success": False,
            "error": f"Failed to process file: {str(e)}",
            "content": ""
        }

def process_pdf(self, file_path: str) -> dict:
    """Extract text from PDF file"""
    try:
        with open(file_path, 'rb') as file:
            reader = PyPDF2.PdfReader(file)
            text = ""

            for page in reader.pages:
```

```
            text += page.extract_text() + "\n"

        return {
            "success": True,
            "content": text.strip(),
            "metadata": {
                "pages": len(reader.pages),
                "type": "pdf"
            }
        }
    except Exception as e:
        return {
            "success": False,
            "error": str(e),
            "content": ""
        }

def process_docx(self, file_path: str) -> dict:
    """Extract text from DOCX file"""
    try:
        doc = docx.Document(file_path)
        text = ""

        for paragraph in doc.paragraphs:
            text += paragraph.text + "\n"

        return {
            "success": True,
            "content": text.strip(),
            "metadata": {
```

```
                "paragraphs": len(doc.paragraphs),
                "type": "docx"
            }
        }
    except Exception as e:
        return {
            "success": False,
            "error": str(e),
            "content": ""
        }

def process_txt(self, file_path: str) -> dict:
    """Read text file content"""
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            content = file.read()

            return {
                "success": True,
                "content": content,
                "metadata": {
                    "size": len(content),
                    "type": "txt"
                }
            }
    except Exception as e:
        return {
            "success": False,
            "error": str(e),
            "content": ""
```

```
        }

    def process_excel(self, file_path: str) -> dict:
        """Process Excel file and convert to text representation"""
        try:
            # Read all sheets
            excel_data = pd.read_excel(file_path, sheet_name=None)

            text_content = ""
            sheets_info = {}

            for sheet_name, df in excel_data.items():
                # Convert DataFrame to readable text
                sheet_text = f"Sheet: {sheet_name}\n"
                sheet_text += f"Columns: {', '.join(df.columns)}\n"
                sheet_text += f"Rows: {len(df)}\n\n"

                # Add sample data (first 10 rows)
                sample_data = df.head(10).to_string(index=False)
                sheet_text += f"Sample Data:\n{sample_data}\n\n"

                # Add summary statistics for numeric columns
                numeric_cols = df.select_dtypes(include=[float, int]).columns
                if len(numeric_cols) > 0:
                    stats = df[numeric_cols].describe()
                    sheet_text += f"Summary Statistics:\n{stats.to_string()}\n\n"

                text_content += sheet_text
                sheets_info[sheet_name] = {
                    "shape": df.shape,
```

```
            "columns": df.columns.tolist()
        }

    return {
        "success": True,
        "content": text_content,
        "metadata": {
            "sheets": list(excel_data.keys()),
            "sheets_info": sheets_info,
            "type": "excel"
        }
    }
except Exception as e:
    return {
        "success": False,
        "error": str(e),
        "content": ""
    }
```

4.2 Admin-Only File Upload

Secure File Upload Endpoint

Copilot Prompt: *"Create FastAPI file upload endpoint restricted to admin users using existing Project 4 RBAC. Include file validation, secure storage, and processing integration."*

Python

```
from fastapi import File, UploadFile
import shutil
```

```
import uuid
from pathlib import Path

# ...existing imports...

UPLOAD_DIR = Path("./uploads")
UPLOAD_DIR.mkdir(exist_ok=True)

MAX_FILE_SIZE = 10 * 1024 * 1024  # 10MB
ALLOWED_TYPES = {
    'application/pdf',
    'application/vnd.openxmlformats-officedocument.wordprocessingml.document',
    'text/plain',
    'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet',
    'application/vnd.ms-excel'
}

@router.post("/files/upload")
async def upload_file(
    thread_id: int,
    file: UploadFile = File(...),
    current_user: User = Depends(get_current_admin_user)  # Admin only from Project 4
):
    """Upload file - Admin users only"""

    # Validate file type
    if file.content_type not in ALLOWED_TYPES:
        raise HTTPException(
            status_code=400,
            detail=f"File type {file.content_type} not supported"
```

```
    )

# Validate file size
if file.size > MAX_FILE_SIZE:
    raise HTTPException(
        status_code=400,
        detail="File size exceeds 10MB limit"
    )

# Verify thread exists and belongs to user
thread = db.query(ChatThread).filter(
    ChatThread.id == thread_id,
    ChatThread.user_id == current_user.id
).first()

if not thread:
    raise HTTPException(status_code=404, detail="Thread not found")

try:
    # Generate unique filename
    file_extension = Path(file.filename).suffix
    unique_filename = f"{uuid.uuid4()}{file_extension}"
    file_path = UPLOAD_DIR / unique_filename

    # Save file to disk
    with open(file_path, "wb") as buffer:
        shutil.copyfileobj(file.file, buffer)

    # Process file content
    processor = SimpleFileProcessor()
```

```
    result = processor.process_file(str(file_path), file.content_type)

    # Save file record to database
    db_file = UploadedFile(
        user_id=current_user.id,
        thread_id=thread_id,
        filename=file.filename,
        file_type=file.content_type,
        file_size=file.size,
        file_path=str(file_path),
        processed_content=result.get("content", "")
    )

    db.add(db_file)
    db.commit()
    db.refresh(db_file)

    return {
        "message": "File uploaded successfully",
        "file_id": db_file.id,
        "filename": file.filename,
        "processed": result.get("success", False),
        "processing_error": result.get("error")
    }

except Exception as e:
    # Clean up file if database save fails
    if file_path.exists():
        file_path.unlink()
```

```
        raise HTTPException(
            status_code=500,
            detail=f"Failed to upload file: {str(e)}"
        )
```

4.3 Direct LLM Integration

Simple LLM Analysis Service

Copilot Prompt: *"Create a service that combines file content with user prompts and sends them directly to OpenAI GPT-4. Include prompt engineering for file analysis and proper error handling."*

Python

```
import openai
from typing import Optional

class SimpleLLMAnalyzer:
    """Direct LLM integration for file analysis"""

    def __init__(self, api_key: str):
        self.client = openai.OpenAI(api_key=api_key)

    def analyze_with_file(self, user_prompt: str, file_content: str, filename: str, file_type: str)
        -> dict:
        """Analyze file content with user prompt"""

        try:
            # Create enhanced prompt
            enhanced_prompt = self.create_enhanced_prompt(
                user_prompt, file_content, filename, file_type
```

```
        )

        # Send to OpenAI
        response = self.client.chat.completions.create(
            model="gpt-4",
            messages=[
                {
                    "role": "system",
                    "content": "You are a helpful assistant that analyzes documents and data
files. Provide clear, accurate responses based on the file content provided."
                },
                {
                    "role": "user",
                    "content": enhanced_prompt
                }
            ],
            max_tokens=1500,
            temperature=0.1
        )

        return {
            "success": True,
            "response": response.choices[0].message.content,
            "model_used": "gpt-4"
        }

    except Exception as e:
        return {
            "success": False,
            "error": f"LLM analysis failed: {str(e)}",
```

```
            "response": "I'm sorry, I couldn't analyze the file due to a technical error."
        }

    def create_enhanced_prompt(self, user_prompt: str, file_content: str, filename: str,
      file_type: str) -> str:
        """Create enhanced prompt combining user query with file content"""

        # Truncate content if too long (keep within token limits)
        max_content_length = 8000  # Conservative limit for GPT-4
        if len(file_content) > max_content_length:
            file_content = file_content[:max_content_length] + "\n...[Content truncated]"

        prompt = f"""I have uploaded a file that I need help analyzing.

File Information:
- Filename: {filename}
- File Type: {file_type}

File Content:
===== START FILE CONTENT =====
{file_content}
===== END FILE CONTENT =====

User Question: {user_prompt}

Please analyze the file content and answer my question. If the file contains data, provide
      insights and summaries as appropriate. Be specific and reference the actual content
      from the file."""

        return prompt
```

```
def analyze_without_file(self, user_prompt: str) -> dict:
    """Handle regular chat without file content"""

    try:
        response = self.client.chat.completions.create(
            model="gpt-4",
            messages=[
                {
                    "role": "system",
                    "content": "You are a helpful assistant. Provide clear and accurate
  responses."
                },
                {
                    "role": "user",
                    "content": user_prompt
                }
            ],
            max_tokens=1000,
            temperature=0.7
        )

        return {
            "success": True,
            "response": response.choices[0].message.content,
            "model_used": "gpt-4"
        }

    except Exception as e:
        return {
            "success": False,
```

```
            "error": f"LLM analysis failed: {str(e)}",
            "response": "I'm sorry, I couldn't process your request due to a technical error."
        }
```

4.4 Enhanced Chat Endpoint with File Analysis

Update Send Message Endpoint

Copilot Prompt: *"Update the send_message endpoint to integrate LLM analysis. Check for uploaded files in the thread, combine them with user prompts, and return AI-generated responses."*

Python

```
# ...existing imports...
from app.services.llm_analyzer import SimpleLLMAnalyzer

# Initialize LLM service
llm_analyzer = SimpleLLMAnalyzer(api_key=os.getenv("OPENAI_API_KEY"))

@router.post("/chat/threads/{thread_id}/messages")
async def send_message(
    thread_id: int,
    content: str,
    current_user: User = Depends(get_current_user)
):
    """Send a message and get AI response"""

    # Verify thread ownership
    thread = db.query(ChatThread).filter(
        ChatThread.id == thread_id,
```

```
    ChatThread.user_id == current_user.id
).first()

if not thread:
    raise HTTPException(status_code=404, detail="Thread not found")

# Save user message
user_message = Message(
    thread_id=thread_id,
    content=content,
    role="user"
)
db.add(user_message)
db.commit()

# Get uploaded files for this thread
thread_files = db.query(UploadedFile).filter(
    UploadedFile.thread_id == thread_id
).all()

# Generate AI response
if thread_files:
    # Analyze with file content
    # For simplicity, use the most recent file
    latest_file = max(thread_files, key=lambda f: f.created_at)

    result = llm_analyzer.analyze_with_file(
        user_prompt=content,
        file_content=latest_file.processed_content or "",
        filename=latest_file.filename,
```

```
            file_type=latest_file.file_type
        )
    else:
        # Regular chat without files
        result = llm_analyzer.analyze_without_file(content)

    # Save assistant response
    assistant_message = Message(
        thread_id=thread_id,
        content=result["response"],
        role="assistant"
    )
    db.add(assistant_message)
    db.commit()

    # Update thread timestamp
    thread.updated_at = datetime.utcnow()
    db.commit()

    return {
        "user_message": {
            "id": user_message.id,
            "content": user_message.content,
            "role": user_message.role,
            "created_at": user_message.created_at
        },
        "assistant_message": {
            "id": assistant_message.id,
            "content": assistant_message.content,
            "role": assistant_message.role,
```

```
            "created_at": assistant_message.created_at
        },
        "analysis_info": {
            "files_used": len(thread_files),
            "llm_success": result["success"]
        }
    }
```

5. Milestone 3: Frontend Integration

5.1 Chat Interface Components

Main Chat Component

Copilot Prompt: *"Create React chat interface that integrates with existing Project 4 authentication. Include message display, input handling, file upload for admins, and real-time message updates."*

```
None
// ...existing Project 4 imports and AuthContext...

const ChatInterface = () => {
    const { user } = useAuth(); // Use existing Project 4 auth context
    const [threads, setThreads] = useState([]);
    const [activeThread, setActiveThread] = useState(null);
    const [messages, setMessages] = useState([]);
    const [newMessage, setNewMessage] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [uploadedFiles, setUploadedFiles] = useState([]);
```

```
// Load user's chat threads
useEffect(() => {
  loadThreads();
}, []);

// Load messages when thread changes
useEffect(() => {
  if (activeThread) {
    loadMessages(activeThread.id);
    loadThreadFiles(activeThread.id);
  }
}, [activeThread]);

const loadThreads = async () => {
  try {
    const response = await api.get('/chat/threads');
    setThreads(response.data);

    // Select first thread if available
    if (response.data.length > 0 && !activeThread) {
      setActiveThread(response.data[0]);
    }
  } catch (error) {
    console.error('Failed to load threads:', error);
  }
};

const loadMessages = async (threadId) => {
  try {
    const response = await api.get(`/chat/threads/${threadId}/messages`);
```

```
      setMessages(response.data);
    } catch (error) {
      console.error('Failed to load messages:', error);
    }
};

const loadThreadFiles = async (threadId) => {
    try {
      const response = await api.get(`/files/thread/${threadId}`);
      setUploadedFiles(response.data);
    } catch (error) {
      console.error('Failed to load files:', error);
    }
};

const createNewThread = async () => {
    try {
      const response = await api.post('/chat/threads', {
        title: 'New Chat'
      });

      const newThread = response.data;
      setThreads(prev => [newThread, ...prev]);
      setActiveThread(newThread);
    } catch (error) {
      console.error('Failed to create thread:', error);
    }
};

const sendMessage = async () => {
```

```
    if (!newMessage.trim() || !activeThread || isLoading) return;

    setIsLoading(true);
    const messageContent = newMessage;
    setNewMessage('');

    try {
      const response = await api.post(`/chat/threads/${activeThread.id}/messages`, {
        content: messageContent
      });

      // Add both user and assistant messages
      setMessages(prev => [
        ...prev,
        response.data.user_message,
        response.data.assistant_message
      ]);

    } catch (error) {
      console.error('Failed to send message:', error);
      setNewMessage(messageContent); // Restore message on error
    } finally {
      setIsLoading(false);
    }
  };

  const handleKeyPress = (e) => {
    if (e.key === 'Enter' && !e.shiftKey) {
      e.preventDefault();
      sendMessage();
```

```
  }
};

return (
  <div className="flex h-screen bg-gray-100">
    {/* Sidebar - Thread List */}
    <div className="w-1/4 bg-white border-r border-gray-200 flex flex-col">
      <div className="p-4 border-b border-gray-200">
        <button
          onClick={createNewThread}
          className="w-full bg-blue-600 text-white py-2 px-4 rounded-lg
hover:bg-blue-700"
        >
          New Chat
        </button>
      </div>

      <div className="flex-1 overflow-y-auto">
        {threads.map(thread => (
          <div
            key={thread.id}
            onClick={() => setActiveThread(thread)}
            className={`p-4 cursor-pointer border-b border-gray-100
hover:bg-gray-50 ${
              activeThread?.id === thread.id ? 'bg-blue-50 border-blue-200' : ''
            }`}
          >
            <div className="font-medium text-gray-900 truncate">
              {thread.title}
            </div>
            <div className="text-sm text-gray-500">
```

```
                {thread.message_count} messages
              </div>
            </div>
          ))}
        </div>
      </div>

      {/* Main Chat Area */}
      <div className="flex-1 flex flex-col">
        {activeThread ? (
          <>
            {/* Chat Header */}
            <div className="bg-white border-b border-gray-200 p-4">
              <h2 className="text-lg font-semibold text-gray-900">
                {activeThread.title}
              </h2>
              {uploadedFiles.length > 0 && (
                <div className="text-sm text-gray-600 mt-1">
                  Files: {uploadedFiles.map(f => f.filename).join(', ')}
                </div>
              )}
            </div>

            {/* Messages */}
            <div className="flex-1 overflow-y-auto p-4 space-y-4">
              {messages.map(message => (
                <MessageBubble key={message.id} message={message} />
              ))}
              {isLoading && (
                <div className="flex justify-start">
```

```
                <div className="bg-gray-200 rounded-lg p-3">
                  <div className="animate-pulse">Thinking...</div>
                </div>
              </div>
            )}
          </div>

          {/* File Upload Area (Admin Only) */}
          {user?.role === 'admin' && (
            <FileUploadArea
              threadId={activeThread.id}
              onFileUploaded={() => loadThreadFiles(activeThread.id)}
            />
          )}

          {/* Message Input */}
          <div className="bg-white border-t border-gray-200 p-4">
            <div className="flex space-x-2">
              <textarea
                value={newMessage}
                onChange={(e) => setNewMessage(e.target.value)}
                onKeyPress={handleKeyPress}
                placeholder="Ask about your files or chat..."
                className="flex-1 resize-none rounded-lg border border-gray-300
p-3 focus:ring-2 focus:ring-blue-500 focus:border-transparent"
                rows="2"
              />
              <button
                onClick={sendMessage}
                disabled={!newMessage.trim() || isLoading}
```

```
                        className="bg-blue-600 text-white px-6 py-2 rounded-lg
    hover:bg-blue-700 disabled:opacity-50 disabled:cursor-not-allowed"
                      >
                        Send
                      </button>
                    </div>
                  </div>
                </>
              ) : (
                <div className="flex-1 flex items-center justify-center text-gray-500">
                  Select a chat thread to start messaging
                </div>
              )}
            </div>
          </div>
        );
};

const MessageBubble = ({ message }) => {
  const isUser = message.role === 'user';

  return (
    <div className={`flex ${isUser ? 'justify-end' : 'justify-start'}`}>
      <div className={`max-w-2xl rounded-lg p-3 ${
        isUser
          ? 'bg-blue-600 text-white'
          : 'bg-gray-200 text-gray-900'
      }`}>
        <div className="whitespace-pre-wrap">{message.content}</div>
        <div className={`text-xs mt-2 ${
```

```
                isUser ? 'text-blue-100' : 'text-gray-500'
              }`}>
                {new Date(message.created_at).toLocaleTimeString()}
              </div>
            </div>
          </div>
        );
      };
```

5.2 File Upload Component

Admin File Upload Interface

Copilot Prompt: *"Create file upload component for admin users with drag-and-drop, file validation, upload progress, and file management. Show uploaded files and allow deletion."*

```
None
    const FileUploadArea = ({ threadId, onFileUploaded }) => {
      const [isDragOver, setIsDragOver] = useState(false);
      const [uploading, setUploading] = useState(false);
      const [uploadProgress, setUploadProgress] = useState(0);
      const fileInputRef = useRef();

      const allowedTypes = [
        'application/pdf',
        'application/vnd.openxmlformats-officedocument.wordprocessingml.document',
        'text/plain',
        'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet',
        'application/vnd.ms-excel'
```

```
];

const validateFile = (file) => {
  if (!allowedTypes.includes(file.type)) {
    alert(`File type ${file.type} is not supported`);
    return false;
  }

  if (file.size > 10 * 1024 * 1024) { // 10MB
    alert('File size must be less than 10MB');
    return false;
  }

  return true;
};

const uploadFile = async (file) => {
  if (!validateFile(file)) return;

  setUploading(true);
  setUploadProgress(0);

  const formData = new FormData();
  formData.append('file', file);
  formData.append('thread_id', threadId);

  try {
    const response = await api.post('/files/upload', formData, {
      headers: {
        'Content-Type': 'multipart/form-data',
```

```
            },
            onUploadProgress: (progressEvent) => {
                const progress = Math.round(
                    (progressEvent.loaded * 100) / progressEvent.total
                );
                setUploadProgress(progress);
            }
        });

        if (response.data.processed) {
            alert(`File "${file.name}" uploaded and processed successfully!`);
        } else {
            alert(`File uploaded but processing failed: ${response.data.processing_error}`);
        }

        onFileUploaded();
    } catch (error) {
        console.error('Upload failed:', error);
        alert('Failed to upload file. Please try again.');
    } finally {
        setUploading(false);
        setUploadProgress(0);
    }
};

const handleDrop = (e) => {
    e.preventDefault();
    setIsDragOver(false);

    const files = Array.from(e.dataTransfer.files);
```

```
    files.forEach(uploadFile);
};

const handleDragOver = (e) => {
    e.preventDefault();
    setIsDragOver(true);
};

const handleDragLeave = () => {
    setIsDragOver(false);
};

const handleFileSelect = (e) => {
    const files = Array.from(e.target.files);
    files.forEach(uploadFile);
    e.target.value = ''; // Reset file input
};

return (
    <div className="bg-gray-50 border-t border-gray-200 p-4">
        <div
            className={`border-2 border-dashed rounded-lg p-6 text-center cursor-pointer
  transition-colors ${
                isDragOver ? 'border-blue-500 bg-blue-50' : 'border-gray-300'
            } ${uploading ? 'pointer-events-none opacity-50' : ''}`}
            onDrop={handleDrop}
            onDragOver={handleDragOver}
            onDragLeave={handleDragLeave}
            onClick={() => !uploading && fileInputRef.current?.click()}
        >
```

```
<input
  ref={fileInputRef}
  type="file"
  multiple
  accept=".pdf,.docx,.txt,.xlsx,.xls"
  onChange={handleFileSelect}
  className="hidden"
/>

{uploading ? (
  <div className="space-y-2">
    <div className="text-blue-600">Uploading... {uploadProgress}%</div>
    <div className="w-full bg-gray-200 rounded-full h-2">
      <div
        className="bg-blue-600 h-2 rounded-full transition-all"
        style={{ width: `${uploadProgress}%` }}
      />
    </div>
  </div>
) : (
  <div>
    <svg className="mx-auto h-12 w-12 text-gray-400" stroke="currentColor" fill="none" viewBox="0 0 48 48">
      <path d="M28 8H12a4 4 0 00-4 4v20m32-12v8m0 0v8a4 4 0 01-4 4H12a4 4 0 01-4-4v-4m32-4l-3.172-3.172a4 4 0 00-5.656 0L28 28M8 32l9.172-9.172a4 4 0 015.656 0L28 28m0 0l4 4m4-24h8m-4-4v8m-12 4h.02" strokeWidth="2" strokeLinecap="round" strokeLinejoin="round" />
    </svg>
    <div className="mt-2 text-sm text-gray-600">
      <span className="font-medium text-blue-600">Click to upload</span> or drag and drop
```

```
                    </div>
                    <div className="text-xs text-gray-500 mt-1">
                      PDF, DOCX, TXT, XLSX files (max 10MB)
                    </div>
                  </div>
                )}
              </div>
            </div>
          );
        };
```

6. Stretch Goals Implementation

6.1 Basic RAG System (Level 3 Stretch Goal)

Vector Embeddings and Search

Copilot Prompt: *"Implement basic RAG system with document chunking, OpenAI embeddings, pgvector storage, and semantic search for more accurate document analysis."*

Python

```
# Only implement if core functionality is complete
class BasicRAGSystem:
    """Simple RAG implementation for document analysis"""

    def __init__(self):
        self.embedding_model = "text-embedding-ada-002"
        self.chunk_size = 1000
        self.chunk_overlap = 200
```

```
def chunk_document(self, text: str) -> List[str]:
    """Split document into chunks for embedding"""
    # Simple chunking by sentences
    sentences = text.split('. ')
    chunks = []
    current_chunk = ""

    for sentence in sentences:
        if len(current_chunk + sentence) < self.chunk_size:
            current_chunk += sentence + ". "
        else:
            if current_chunk:
                chunks.append(current_chunk.strip())
            current_chunk = sentence + ". "

    if current_chunk:
        chunks.append(current_chunk.strip())

    return chunks

def create_embeddings(self, file_id: int, text: str):
    """Create and store embeddings for document chunks"""
    chunks = self.chunk_document(text)

    for i, chunk in enumerate(chunks):
        # Generate embedding
        response = openai.Embedding.create(
            model=self.embedding_model,
            input=chunk
```

```
        )

        embedding = response['data'][0]['embedding']

        # Store in database (requires pgvector extension)
        # This would require additional database setup
        pass

    def semantic_search(self, query: str, file_ids: List[int], top_k: int = 3):
        """Find relevant document chunks"""
        # Generate query embedding
        query_response = openai.Embedding.create(
            model=self.embedding_model,
            input=query
        )

        query_embedding = query_response['data'][0]['embedding']

        # Search similar chunks (requires pgvector)
        # This would require vector similarity search
        pass
```

7. Success Criteria & Testing

Core Requirements Checklist

- Project 4 Integration: Successfully extends existing auth system
- Chat Functionality: Create threads, send messages, view history
- File Upload: Admin-only upload with validation and security
- File Processing: Extract text from PDF, DOCX, TXT, Excel files

- LLM Integration: Generate relevant responses using file content
- Role-Based Access: Proper permission enforcement
- Error Handling: Graceful handling of failures

8. Deployment Considerations

Environment Variables

None

```
# Add to existing Project 4 .env file
# Keep all existing variables from Project 4

# New Project 5 variables
OPENAI_API_KEY=your_openai_api_key_here
UPLOAD_DIRECTORY=./uploads
MAX_FILE_SIZE_MB=10
```

Dependencies

None

```
# Add to existing Project 4 requirements.txt
PyPDF2==3.0.1
python-docx==0.8.11
pandas==2.1.3
openpyxl==3.1.2
python-multipart==0.0.6
```

File Storage Setup

- Create secure uploads directory
- Set proper file permissions
- Consider cloud storage for production

- Implement file cleanup policies

