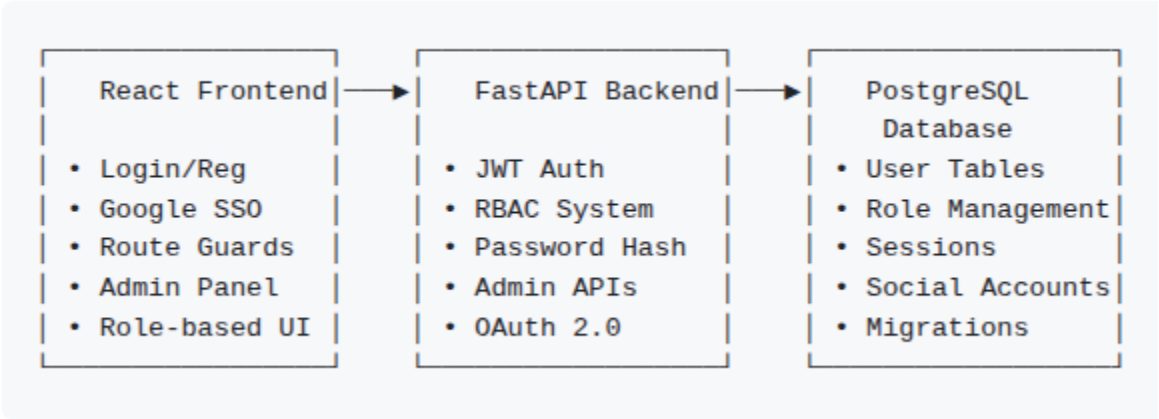
Project 4 - Complete Authentication & Authorization System - Technical Implementation Guide

1. Project Overview & Learning Objectives

Business Context

Transition from Streamlit-based applications to modern full-stack development with secure user authentication and comprehensive role-based access control (RBAC). This project establishes the foundation for multi-user applications with both authentication and authorization capabilities.

Enhanced Architecture Overview



Core Learning Goals

- Authentication & Security: JWT tokens, bcrypt hashing, Google SSO, protected routes
- Authorization Systems: Role-Based Access Control (RBAC) with user/admin roles
- Full-Stack Architecture: Clean separation between FastAPI backend and React frontend
- Database Management: PostgreSQL with user data, sessions, roles, and social accounts
- Modern Frontend: React + Vite + Tailwind CSS development patterns
- Social Authentication: OAuth 2.0 integration with major providers

2. Implementation Strategy & Copilot Integration

Development Approach

This guide provides implementation roadmaps that align with the 6-milestone structure from the Project Assignment:

1. Milestone 1: Core Authentication Foundation
2. Milestone 2: Authentication System & JWT Implementation
3. Milestone 3: Role-Based Access Control (RBAC) Implementation
4. Milestone 4: Frontend Authorization & Admin Features
5. Milestone 5: Social Authentication Integration
6. Milestone 6: Production Readiness & Advanced Features

Copilot Optimization Tips

- Use specific, context-rich prompts that include both authentication AND authorization requirements
- Reference the role-based data models and API patterns provided in this guide
- Ask Copilot to explain security implications of both authentication and authorization code
- Request validation and error handling patterns for role-based access control

3. Milestone 1: Core Authentication Foundation

3.1 Enhanced Project Structure Setup

Recommended Directory Structure with Role Support and Future Chat Integration

```
None
project-4-auth-system/
├── backend/
│   ├── app/
```

```
│   │   ├── __init__.py
│   │   ├── main.py
│   │   ├── models/
│   │   │   ├── __init__.py
│   │   │   ├── user.py            # User model with roles
│   │   │   ├── social_account.py  # Social authentication
│   │   │   ├── chat_thread.py     # Chat threads (Project 5 foundation)
│   │   │   ├── message.py         # Chat messages (Project 5 foundation)
│   │   │   └── uploaded_file.py   # File attachments (Project 5 foundation)
│   │   ├── schemas/
│   │   │   ├── __init__.py
│   │   │   ├── user.py            # User schemas with role validation
│   │   │   ├── token.py           # JWT schemas with role claims
│   │   │   ├── chat.py            # Chat schemas (Project 5 foundation)
│   │   │   └── file.py            # File schemas (Project 5 foundation)
│   │   ├── services/
│   │   │   ├── __init__.py
│   │   │   ├── auth.py            # Authentication service
│   │   │   ├── authorization.py   # RBAC service
│   │   │   ├── chat.py            # Chat service (Project 5 foundation)
│   │   │   ├── file_service.py    # File handling service (Project 5 foundation)
│   │   │   └── database.py
│   │   ├── api/
│   │   │   ├── __init__.py
│   │   │   └── endpoints/
│   │   │       ├── __init__.py
│   │   │       ├── auth.py        # Auth endpoints
│   │   │       ├── admin.py       # Admin-only endpoints
│   │   │       ├── chat.py        # Chat endpoints (Project 5 foundation)
│   │   │       ├── files.py       # File endpoints (Project 5 foundation)
│   │   │       └── oauth.py       # Social auth endpoints
│   │   └── core/
│   │       ├── __init__.py
│   │       ├── config.py
│   │       ├── security.py
│   │       └── rbac.py            # Role-based access control
```

```
│   ├── requirements.txt
│   └── .env
├── frontend/
│   ├── src/
│   │   ├── components/
│   │   │   ├── auth/            # Authentication components
│   │   │   ├── admin/           # Admin-only components
│   │   │   ├── chat/            # Chat components (Project 5 foundation)
│   │   │   ├── files/           # File components (Project 5 foundation)
│   │   │   └── common/          # Shared components
│   │   ├── pages/
│   │   │   ├── Login.tsx
│   │   │   ├── Register.tsx
│   │   │   ├── Dashboard.tsx    # Enhanced with chat preview
│   │   │   ├── ChatPage.tsx     # Chat interface (Project 5 foundation)
│   │   │   └── AdminPanel.tsx   # Admin-only page
│   │   ├── contexts/
│   │   │   ├── AuthContext.tsx  # Auth context with roles
│   │   │   └── ChatContext.tsx  # Chat context (Project 5 foundation)
│   │   ├── hooks/
│   │   │   ├── useAuth.ts       # Authentication hook
│   │   │   ├── useRBAC.ts       # Role-based access hook
│   │   │   ├── useChat.ts       # Chat hook (Project 5 foundation)
│   │   │   └── useFileUpload.ts # File upload hook (Project 5 foundation)
│   │   ├── routes/
│   │   │   ├── ProtectedRoute.tsx # General protected routes
│   │   │   └── AdminRoute.tsx   # Admin-only routes
│   │   └── utils/
│   │       ├── api.ts
│   │       ├── roles.ts         # Role utilities
│   │       └── fileUtils.ts     # File utilities (Project 5 foundation)
│   ├── package.json
│   └── vite.config.js
└── README.md
```

3.2 Enhanced Database Design & Models with RBAC and Chat Foundation

Complete Database Schema (Project 4 + Project 5 Foundation)

Copilot Prompt: *"Create a comprehensive SQLAlchemy schema for authentication with RBAC and chat system foundation including users with roles, chat threads, messages, and file attachments. Include proper relationships, indexes, and constraints for scalable multi-user chat system."*

```sql
-- Users table with roles (Project 4 - Authentication & Authorization)
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    role VARCHAR(20) DEFAULT 'user' NOT NULL CHECK (role IN ('user', 'admin')),
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- User sessions table for token management (Project 4)
CREATE TABLE user_sessions (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    token_jti VARCHAR(255) UNIQUE NOT NULL,
    expires_at TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Social accounts table with role inheritance (Project 4)
CREATE TABLE social_accounts (
```

```
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) NOT NULL ON DELETE CASCADE,
    provider VARCHAR(20) NOT NULL,
    provider_user_id VARCHAR(255) NOT NULL,
    provider_email VARCHAR(255),
    access_token TEXT,
    refresh_token TEXT,
    expires_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(provider, provider_user_id)
);

-- Chat threads table (Project 5 foundation - prepared in Project 4)
CREATE TABLE chat_threads (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) NOT NULL ON DELETE CASCADE,
    title VARCHAR(200),
    thread_type VARCHAR(20) DEFAULT 'general' CHECK (thread_type IN ('general', 'document',
'data_analysis')),
    is_active BOOLEAN DEFAULT TRUE,
    metadata JSONB DEFAULT '{}',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Messages table (Project 5 foundation - prepared in Project 4)
CREATE TABLE messages (
    id SERIAL PRIMARY KEY,
    thread_id INTEGER REFERENCES chat_threads(id) NOT NULL ON DELETE CASCADE,
    user_id INTEGER REFERENCES users(id) ON DELETE SET NULL,
    content TEXT NOT NULL,
    message_type VARCHAR(20) DEFAULT 'text' CHECK (message_type IN ('text', 'system', 'file',
'image', 'analysis')),
    role VARCHAR(20) DEFAULT 'user' CHECK (role IN ('user', 'assistant', 'system')),
    metadata JSONB DEFAULT '{}',
```

```
    parent_message_id INTEGER REFERENCES messages(id),
    is_edited BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- File attachments table (Project 5 foundation - prepared in Project 4)
CREATE TABLE uploaded_files (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) NOT NULL ON DELETE CASCADE,
    thread_id INTEGER REFERENCES chat_threads(id) ON DELETE SET NULL,
    message_id INTEGER REFERENCES messages(id) ON DELETE SET NULL,
    filename VARCHAR(255) NOT NULL,
    original_filename VARCHAR(255) NOT NULL,
    file_path VARCHAR(500),
    file_size BIGINT NOT NULL,
    mime_type VARCHAR(100),
    file_type VARCHAR(50) NOT NULL CHECK (file_type IN ('document', 'image', 'data', 'other')),
    processing_status VARCHAR(20) DEFAULT 'pending' CHECK (processing_status IN
('pending', 'processing', 'completed', 'failed')),
    metadata JSONB DEFAULT '{}',
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Document embeddings table (Project 5 foundation - RAG system)
CREATE TABLE document_embeddings (
    id SERIAL PRIMARY KEY,
    file_id INTEGER REFERENCES uploaded_files(id) NOT NULL ON DELETE CASCADE,
    chunk_text TEXT NOT NULL,
    chunk_index INTEGER NOT NULL,
    embedding_vector FLOAT8[] DEFAULT '{}',  -- For vector similarity search
    token_count INTEGER DEFAULT 0,
    metadata JSONB DEFAULT '{}',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
);

-- Analysis results table (Project 5 foundation - data analysis results)
CREATE TABLE analysis_results (
    id SERIAL PRIMARY KEY,
    thread_id INTEGER REFERENCES chat_threads(id) NOT NULL ON DELETE CASCADE,
    message_id INTEGER REFERENCES messages(id) ON DELETE SET NULL,
    user_id INTEGER REFERENCES users(id) NOT NULL ON DELETE CASCADE,
    query_text TEXT NOT NULL,
    result_type VARCHAR(50) NOT NULL CHECK (result_type IN ('chart', 'table', 'summary',
'calculation')),
    result_data JSONB NOT NULL DEFAULT '{}',
    generated_code TEXT,
    execution_time_ms INTEGER,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Indexes for optimal performance
-- Authentication & Authorization indexes (Project 4)
CREATE INDEX idx_users_role ON users(role);
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_username ON users(username);
CREATE INDEX idx_social_provider ON social_accounts(provider, provider_user_id);
CREATE INDEX idx_sessions_user_id ON user_sessions(user_id);
CREATE INDEX idx_sessions_expires ON user_sessions(expires_at);

-- Chat system indexes (Project 5 foundation)
CREATE INDEX idx_threads_user_id ON chat_threads(user_id);
CREATE INDEX idx_threads_created_at ON chat_threads(created_at);
CREATE INDEX idx_threads_type ON chat_threads(thread_type);
CREATE INDEX idx_messages_thread_id ON messages(thread_id);
CREATE INDEX idx_messages_user_id ON messages(user_id);
CREATE INDEX idx_messages_created_at ON messages(created_at);
CREATE INDEX idx_messages_type ON messages(message_type);

-- File system indexes (Project 5 foundation)
```

```
CREATE INDEX idx_files_user_id ON uploaded_files(user_id);
CREATE INDEX idx_files_thread_id ON uploaded_files(thread_id);
CREATE INDEX idx_files_type ON uploaded_files(file_type);
CREATE INDEX idx_files_status ON uploaded_files(processing_status);
CREATE INDEX idx_embeddings_file_id ON document_embeddings(file_id);
CREATE INDEX idx_analysis_thread_id ON analysis_results(thread_id);
CREATE INDEX idx_analysis_user_id ON analysis_results(user_id);
```

Enhanced SQLAlchemy Models with Complete Relationships

Copilot Prompt: *"Create comprehensive SQLAlchemy models for the complete schema including User model with roles, ChatThread, Message, UploadedFile, and all supporting tables with proper relationships, cascade deletes, and role-based access methods."*

```python
# models/user.py - Enhanced User model with chat relationships
from sqlalchemy import Boolean, Column, Integer, String, DateTime, Text
from sqlalchemy.sql import func
from sqlalchemy.orm import relationship
from app.db.base import Base
from passlib.context import CryptContext
from enum import Enum
import enum

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

class UserRole(str, enum.Enum):
    USER = "user"
    ADMIN = "admin"

class User(Base):
    __tablename__ = "users"
```

```
    id = Column(Integer, primary_key=True, index=True)
    email = Column(String(100), unique=True, index=True, nullable=False)
    username = Column(String(50), unique=True, index=True, nullable=False)
    hashed_password = Column(String(255), nullable=False)
    first_name = Column(String(50))
    last_name = Column(String(50))
    role = Column(String(20), default=UserRole.USER.value, nullable=False)
    is_active = Column(Boolean, default=True)
    created_at = Column(DateTime(timezone=True), server_default=func.now())
    updated_at = Column(DateTime(timezone=True), onupdate=func.now())

    # Authentication relationships (Project 4)
    sessions = relationship("UserSession", back_populates="user", cascade="all,
delete-orphan")
    social_accounts = relationship("SocialAccount", back_populates="user", cascade="all,
delete-orphan")

    # Chat system relationships (Project 5 foundation)
    chat_threads = relationship("ChatThread", back_populates="user", cascade="all,
delete-orphan")
    messages = relationship("Message", back_populates="user")
    uploaded_files = relationship("UploadedFile", back_populates="user", cascade="all,
delete-orphan")
    analysis_results = relationship("AnalysisResult", back_populates="user", cascade="all,
delete-orphan")

    # Authentication methods (Project 4)
    @classmethod
    def create(cls, db, *, email: str, username: str, password: str, role: str =
UserRole.USER.value):
        hashed_password = pwd_context.hash(password)
        user = cls(
            email=email,
            username=username,
            hashed_password=hashed_password,
            role=role
```

```
        )
        db.add(user)
        db.commit()
        db.refresh(user)
        return user

    def verify_password(self, plain_password: str) -> bool:
        return pwd_context.verify(plain_password, self.hashed_password)

    def has_role(self, role: UserRole) -> bool:
        return self.role == role.value

    def is_admin(self) -> bool:
        return self.role == UserRole.ADMIN.value

    # Chat system methods (Project 5 foundation)
    def can_access_thread(self, thread_id: int) -> bool:
        """Check if user can access a specific chat thread"""
        if self.is_admin():
            return True
        return any(thread.id == thread_id for thread in self.chat_threads)

    def get_active_threads(self):
        """Get user's active chat threads"""
        return [thread for thread in self.chat_threads if thread.is_active]

# models/chat_thread.py - Chat thread model (Project 5 foundation)
class ThreadType(str, enum.Enum):
    GENERAL = "general"
    DOCUMENT = "document"
    DATA_ANALYSIS = "data_analysis"

class ChatThread(Base):
    __tablename__ = "chat_threads"

    id = Column(Integer, primary_key=True, index=True)
```

```
    user_id = Column(Integer, ForeignKey("users.id", ondelete="CASCADE"), nullable=False)
    title = Column(String(200))
    thread_type = Column(String(20), default=ThreadType.GENERAL.value, nullable=False)
    is_active = Column(Boolean, default=True)
    metadata = Column(JSON, default={})
    created_at = Column(DateTime(timezone=True), server_default=func.now())
    updated_at = Column(DateTime(timezone=True), onupdate=func.now())

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

    def get_recent_messages(self, limit: int = 10):
        """Get recent messages in thread"""
        return sorted(self.messages, key=lambda x: x.created_at, reverse=True)[:limit]

    def get_file_count(self) -> int:
        """Get count of files in thread"""
        return len([f for f in self.uploaded_files if f.is_active])

# models/message.py - Message model (Project 5 foundation)
class MessageType(str, enum.Enum):
    TEXT = "text"
    SYSTEM = "system"
    FILE = "file"
    IMAGE = "image"
    ANALYSIS = "analysis"

class MessageRole(str, enum.Enum):
    USER = "user"
    ASSISTANT = "assistant"
    SYSTEM = "system"

class Message(Base):
```

```
    __tablename__ = "messages"

    id = Column(Integer, primary_key=True, index=True)
    thread_id = Column(Integer, ForeignKey("chat_threads.id", ondelete="CASCADE"),
nullable=False)
    user_id = Column(Integer, ForeignKey("users.id", ondelete="SET NULL"))
    content = Column(Text, nullable=False)
    message_type = Column(String(20), default=MessageType.TEXT.value, nullable=False)
    role = Column(String(20), default=MessageRole.USER.value, nullable=False)
    metadata = Column(JSON, default={})
    parent_message_id = Column(Integer, ForeignKey("messages.id"))
    is_edited = Column(Boolean, default=False)
    created_at = Column(DateTime(timezone=True), server_default=func.now())
    updated_at = Column(DateTime(timezone=True), onupdate=func.now())

    # Relationships
    thread = relationship("ChatThread", back_populates="messages")
    user = relationship("User", back_populates="messages")
    uploaded_files = relationship("UploadedFile", back_populates="message")
    parent_message = relationship("Message", remote_side=[id])

    def is_from_user(self) -> bool:
        """Check if message is from user (not assistant)"""
        return self.role == MessageRole.USER.value

# models/uploaded_file.py - File model (Project 5 foundation)
class FileType(str, enum.Enum):
    DOCUMENT = "document"  # PDF, DOCX, TXT
    IMAGE = "image"        # PNG, JPG, etc
    DATA = "data"          # XLSX, CSV
    OTHER = "other"

class ProcessingStatus(str, enum.Enum):
    PENDING = "pending"
    PROCESSING = "processing"
    COMPLETED = "completed"
```

```
    FAILED = "failed"

class UploadedFile(Base):
    __tablename__ = "uploaded_files"

    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id", ondelete="CASCADE"), nullable=False)
    thread_id = Column(Integer, ForeignKey("chat_threads.id", ondelete="SET NULL"))
    message_id = Column(Integer, ForeignKey("messages.id", ondelete="SET NULL"))
    filename = Column(String(255), nullable=False)
    original_filename = Column(String(255), nullable=False)
    file_path = Column(String(500))
    file_size = Column(BigInteger, nullable=False)
    mime_type = Column(String(100))
    file_type = Column(String(50), nullable=False)
    processing_status = Column(String(20), default=ProcessingStatus.PENDING.value,
nullable=False)
    metadata = Column(JSON, default={})
    is_active = Column(Boolean, default=True)
    created_at = Column(DateTime(timezone=True), server_default=func.now())
    updated_at = Column(DateTime(timezone=True), onupdate=func.now())

    # Relationships
    user = relationship("User", back_populates="uploaded_files")
    thread = relationship("ChatThread", back_populates="uploaded_files")
    message = relationship("Message", back_populates="uploaded_files")
    embeddings = relationship("DocumentEmbedding", back_populates="file", cascade="all,
delete-orphan")

    def is_processed(self) -> bool:
        """Check if file processing is complete"""
        return self.processing_status == ProcessingStatus.COMPLETED.value

    def can_be_analyzed(self) -> bool:
        """Check if file can be analyzed (documents/data)"""
        return self.file_type in [FileType.DOCUMENT.value, FileType.DATA.value]
```

4. Milestone 2: Authentication System & JWT Implementation

4.1 Enhanced JWT System with Role Claims

JWT Configuration with Role Support (Updated Requirements)

- Secret Key: Secure random string stored in environment variables
- Token Expiration: 30 minutes for access tokens (as per assignment), 7 days for refresh
- Role Claims: Include user role in JWT payload for RBAC authorization
- Algorithm: HS256 for simplicity
- Role Validation: Verify role claims match current user role

Implementation Reference: Follow the patterns in JWT Authentication and Authorization for secure implementation with roles.

Copilot Prompt: *"Create JWT service with token generation including role claims, validation, refresh functionality, and role-aware middleware using python-jose. Include proper error handling for expired tokens and role validation."*

Enhanced JWT Token Structure with Role Claims

```python
# Enhanced token payload with roles (matches assignment requirements)
{
    "sub": "user-id",           # user_id
    "username": "johndoe",      # username
    "email": "john@example.com",  # user email
    "role": "admin",            # user role for RBAC (key requirement)
    "exp": 1642694400,          # expiration timestamp (30 min)
    "iat": 1642608000,          # issued at
    "jti": "unique-token-id"    # JWT ID for session tracking
}
```

4.2 Enhanced API Endpoints with RBAC and Chat Foundation

Complete API Structure (Project 4 + Project 5 Foundation)

Copilot Prompt: *"Create comprehensive FastAPI endpoint structure including authentication/authorization endpoints with RBAC, basic chat thread management, message handling, and file upload capabilities as foundation for Project 5."*

Python

```
# Authentication endpoints (Project 4)
POST /auth/register            # User registration (default 'user' role)
POST /auth/login               # Login with JWT containing role claims
GET  /auth/me                  # Current user profile with role info
PUT  /auth/profile             # Update user profile (protected)
POST /auth/logout              # Logout and invalidate session

# Role-based authorization endpoints (Project 4 - RBAC)
GET  /auth/users               # List all users (Admin only)
PUT  /auth/users/{user_id}/role  # Change user roles (Admin only)
DELETE /auth/users/{user_id}     # Delete users (Admin only)

# OAuth endpoints (Project 4 - Social authentication)
GET  /auth/login/{provider}      # Initiate OAuth (Google, Facebook, LinkedIn)
GET  /auth/callback/{provider}   # Handle OAuth callback with role assignment
POST /auth/link-social           # Link social account to existing user

# Admin-specific endpoints (Project 4)
GET  /admin/dashboard          # Admin dashboard data (Admin only)
GET  /admin/analytics          # System analytics (Admin only)
POST /admin/users              # Create users as admin (Admin only)

# Chat system endpoints (Project 5 foundation - prepared in Project 4)
GET    /chat/threads           # Get user's chat threads (protected)
POST   /chat/threads           # Create new chat thread (protected)
GET    /chat/threads/{thread_id} # Get specific thread with messages (protected)
PUT    /chat/threads/{thread_id} # Update thread (title, metadata) (protected)
DELETE /chat/threads/{thread_id} # Delete thread (protected)

POST   /chat/threads/{thread_id}/messages  # Send message to thread (protected)
```

```
GET    /chat/threads/{thread_id}/messages  # Get thread messages (protected)
PUT    /chat/messages/{message_id}         # Edit message (protected)
DELETE /chat/messages/{message_id}         # Delete message (protected)

# File handling endpoints (Project 5 foundation - prepared in Project 4)
POST   /files/upload            # Upload file to thread (protected)
GET    /files/{file_id}         # Get file info (protected)
GET    /files/{file_id}/download # Download file (protected)
DELETE /files/{file_id}         # Delete file (protected)
GET    /files/thread/{thread_id} # Get files in thread (protected)

# Admin file management (Project 4 + 5 integration)
GET    /admin/files             # List all files (Admin only)
DELETE /admin/files/{file_id}    # Admin delete file (Admin only)
GET    /admin/files/stats       # File usage statistics (Admin only)
```

Enhanced FastAPI Route Implementation with Chat Foundation

Copilot Prompt: *"Create FastAPI route implementations for chat thread management including RBAC authorization, user ownership validation, and basic CRUD operations for threads and messages as Project 5 foundation."*

```python
# api/endpoints/chat.py - Chat endpoints (Project 5 foundation)
from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session
from app.api.dependencies import get_current_user, get_db
from app.models.user import User
from app.models.chat_thread import ChatThread, Message
from app.schemas.chat import ThreadCreate, ThreadResponse, MessageCreate,
MessageResponse

router = APIRouter(prefix="/chat", tags=["chat"])

@router.get("/threads", response_model=List[ThreadResponse])
```

```python
async def get_user_threads(
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db)
):
    """Get current user's chat threads"""
    threads = db.query(ChatThread).filter(
        ChatThread.user_id == current_user.id,
        ChatThread.is_active == True
    ).order_by(ChatThread.updated_at.desc()).all()

    return threads

@router.post("/threads", response_model=ThreadResponse)
async def create_thread(
    thread_data: ThreadCreate,
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db)
):
    """Create new chat thread for current user"""
    new_thread = ChatThread(
        user_id=current_user.id,
        title=thread_data.title,
        thread_type=thread_data.thread_type,
        metadata=thread_data.metadata or {}
    )

    db.add(new_thread)
    db.commit()
    db.refresh(new_thread)

    return new_thread

@router.get("/threads/{thread_id}", response_model=ThreadResponse)
async def get_thread(
    thread_id: int,
    current_user: User = Depends(get_current_user),
```

```
    db: Session = Depends(get_db)
):
    """Get specific thread with access control"""
    thread = db.query(ChatThread).filter(ChatThread.id == thread_id).first()

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

    # Check access permissions
    if thread.user_id != current_user.id and not current_user.is_admin():
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Access denied to this thread"
        )

    return thread

@router.post("/threads/{thread_id}/messages", response_model=MessageResponse)
async def send_message(
    thread_id: int,
    message_data: MessageCreate,
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db)
):
    """Send message to chat thread"""
    # Verify thread access
    thread = db.query(ChatThread).filter(ChatThread.id == thread_id).first()
    if not thread or (thread.user_id != current_user.id and not current_user.is_admin()):
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Access denied to this thread"
        )

    new_message = Message(
        thread_id=thread_id,
        user_id=current_user.id,
```

```
        content=message_data.content,
        message_type=message_data.message_type,
        role=message_data.role,
        metadata=message_data.metadata or {}
    )

    db.add(new_message)

    # Update thread's updated_at timestamp
    thread.updated_at = func.now()

    db.commit()
    db.refresh(new_message)

    return new_message

# api/endpoints/files.py - File endpoints (Project 5 foundation)
from fastapi import APIRouter, Depends, HTTPException, UploadFile, File
from app.models.uploaded_file import UploadedFile, FileType, ProcessingStatus
from app.services.file_service import FileService

router = APIRouter(prefix="/files", tags=["files"])

@router.post("/upload")
async def upload_file(
    thread_id: int,
    file: UploadFile = File(...),
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db),
    file_service: FileService = Depends()
):
    """Upload file to chat thread (Project 5 foundation)"""
    # Verify thread access
    thread = db.query(ChatThread).filter(ChatThread.id == thread_id).first()
    if not thread or (thread.user_id != current_user.id and not current_user.is_admin()):
        raise HTTPException(
```

```
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Access denied to this thread"
        )

    # Process file upload
    uploaded_file = await file_service.save_file(
        file=file,
        user_id=current_user.id,
        thread_id=thread_id,
        db=db
    )

    return uploaded_file

@router.get("/thread/{thread_id}")
async def get_thread_files(
    thread_id: int,
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db)
):
    """Get files in specific thread"""
    # Verify thread access
    thread = db.query(ChatThread).filter(ChatThread.id == thread_id).first()
    if not thread or (thread.user_id != current_user.id and not current_user.is_admin()):
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Access denied to this thread"
        )

    files = db.query(UploadedFile).filter(
        UploadedFile.thread_id == thread_id,
        UploadedFile.is_active == True
    ).order_by(UploadedFile.created_at.desc()).all()

    return files
```

5. Milestone 3: Role-Based Access Control (RBAC) Implementation

5.1 Backend RBAC System Implementation

Role-Based Authorization Middleware

Copilot Prompt: *"Create FastAPI RBAC system with role-checking decorators, middleware for admin-only endpoints, proper 403 Forbidden responses, and JWT role validation. Include @require_admin and @require_role decorators."*

```python
# RBAC Implementation Pattern (matches assignment requirements)
from functools import wraps
from fastapi import HTTPException, status, Depends

# Role validation dependency
async def get_current_admin_user(current_user: User = Depends(get_current_user)):
    """Dependency that requires admin role"""
    if current_user.role != 'admin':
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Admin access required"
        )
    return current_user

# Role-based endpoint protection
@router.get("/admin/users", dependencies=[Depends(get_current_admin_user)])
async def get_all_users(db = Depends(get_db)):
    """Admin-only endpoint to list all users"""
    users = db.query(User).all()
    return users
```

Database Migration for Role Support

Copilot Prompt: *"Create Alembic migration to add role column to existing users table with default 'user' value, admin role option, and proper constraints as specified in the database schema."*

5.2 Admin User Management System

Admin-Only User Management APIs

Copilot Prompt: *"Create admin-only FastAPI endpoints for user management including listing users, updating user roles, deactivating users, and creating new users with proper RBAC authorization checks."*

Admin User Seeding Mechanism

Copilot Prompt: *"Create admin user seeding system for testing that allows creation of initial admin users through environment variables or CLI commands, with secure password handling."*

6. Milestone 4: Frontend Authorization & Admin Features

6.1 Enhanced React Authentication Context with RBAC

Enhanced Auth Context with Role Management

Copilot Prompt: *"Create React authentication context with TypeScript that manages user state including role information ('user'/'admin'), login/logout functions, role-based access checking functions (hasRole, isAdmin), token storage with role claims, and automatic role validation."*

```
None
// Enhanced Auth Context Pattern (matches assignment requirements)
```

```
interface AuthContextType {
  user: User | null;
  login: (credentials: LoginData) => Promise<void>;
  logout: () => void;
  hasRole: (role: string) => boolean;
  isAdmin: () => boolean;
  loading: boolean;
}

// Usage pattern for role-based UI
const { user, hasRole, isAdmin } = useAuth();

return (
  <div>
    <UserDashboard />
    {isAdmin() && <AdminPanel />}
    {hasRole('admin') && <UserManagement />}
  </div>
);
```

6.2 Role-Based Route Protection

Enhanced Protected Routes with RBAC

Copilot Prompt: *"Create React route protection components including ProtectedRoute for authenticated users and AdminRoute for admin-only pages, with proper redirects to login for unauthenticated users and 403 error pages for insufficient permissions."*

Navigation Structure with Role-Based Access (From Assignment)

```
None
/ (public)                - Landing page
/login (public)             - Login form
/register (public)          - Registration form
```

```
/dashboard (protected)       - Main user dashboard (all authenticated)
/profile (protected)         - User profile management (all authenticated)
/admin (admin only)          - Admin dashboard (admin role only)
/admin/users (admin only)    - User management (admin role only)
```

6.3 Enhanced Dashboard Implementation with Chat Preview

Admin Dashboard with Chat System Overview

Copilot Prompt: *"Create admin dashboard that includes user management, role assignment, chat thread statistics, file upload monitoring, and system analytics including chat activity metrics."*

```
None
// Enhanced Admin Dashboard with Chat System Integration
const AdminDashboard: React.FC = () => {
  const { user, isAdmin } = useAuth();
  const [stats, setStats] = useState<AdminStats | null>(null);

  useEffect(() => {
    if (isAdmin()) {
      fetchAdminStats();
    }
  }, []);

  const fetchAdminStats = async () => {
    const response = await api.get('/admin/analytics');
    setStats(response.data);
  };

  if (!isAdmin()) {
    return <Navigate to="/dashboard" />;
  }

  return (
```

```
<div className="admin-dashboard">
  <h1>Admin Dashboard</h1>

  {/* User Management Section (Project 4) */}
  <div className="grid grid-cols-1 md:grid-cols-3 gap-6">
    <StatCard
      title="Total Users"
      value={stats?.totalUsers}
      icon="👥"
    />
    <StatCard
      title="Admin Users"
      value={stats?.adminUsers}
      icon="👑"
    />
    <StatCard
      title="Active Sessions"
      value={stats?.activeSessions}
      icon="🔒"
    />
  </div>

  {/* Chat System Statistics (Project 5 foundation) */}
  <div className="grid grid-cols-1 md:grid-cols-4 gap-4 mt-6">
    <StatCard
      title="Chat Threads"
      value={stats?.totalThreads}
      icon="💬"
    />
    <StatCard
      title="Messages Today"
      value={stats?.messagesToday}
      icon="📝"
    />
    <StatCard
      title="Files Uploaded"
```

```
        value={stats?.totalFiles}
        icon="📎"
       />
       <StatCard
        title="Storage Used"
        value={stats?.storageUsed}
        icon="💾"
       />
      </div>

      {/* User Management Table */}
      <UserManagementTable />

      {/* Recent Activity (including chat) */}
      <RecentActivityFeed />
    </div>
  );
};
```

Enhanced User Dashboard with Chat System

Copilot Prompt: *"Create user dashboard that shows authentication status, user profile management, recent chat threads preview, and role-based features with proper RBAC conditional rendering."*

```
None
// Enhanced User Dashboard with Chat Foundation
const Dashboard: React.FC = () => {
  const { user, hasRole } = useAuth();
  const [recentThreads, setRecentThreads] = useState<ChatThread[]>([]);

  useEffect(() => {
    fetchRecentThreads();
  }, []);
```

```
const fetchRecentThreads = async () => {
  try {
    const response = await api.get('/chat/threads?limit=5');
    setRecentThreads(response.data);
  } catch (error) {
    console.error('Failed to fetch recent threads:', error);
  }
};

return (
  <div className="dashboard">
    <div className="welcome-section">
      <h1>Welcome back, {user?.first_name || user?.username}!</h1>
      <p>Role: <span className="role-badge">{user?.role}</span></p>
    </div>

    {/* Quick Actions */}
    <div className="quick-actions">
      <button
        className="action-btn primary"
        onClick={() => navigate('/chat/new')}
      >
        Start New Chat
      </button>

      {hasRole('admin') && (
        <button
          className="action-btn admin"
          onClick={() => navigate('/admin')}
        >
          Admin Panel
        </button>
      )}
    </div>

    {/* Recent Chat Threads Preview (Project 5 foundation) */}
```

```
      <div className="recent-threads">
        <h2>Recent Conversations</h2>
        {recentThreads.length > 0 ? (
          <div className="threads-grid">
            {recentThreads.map(thread => (
              <ThreadPreviewCard
                key={thread.id}
                thread={thread}
                onClick={() => navigate(`/chat/${thread.id}`)}
              />
            ))}
          </div>
        ) : (
          <div className="empty-state">
            <p>No conversations yet. Start your first chat!</p>
            <button onClick={() => navigate('/chat/new')}>
              Create First Chat
            </button>
          </div>
        )}
      </div>

      {/* User Statistics */}
      <div className="user-stats grid grid-cols-2 md:grid-cols-4 gap-4">
        <StatCard title="Chat Threads" value={user?.threadCount} />
        <StatCard title="Messages Sent" value={user?.messageCount} />
        <StatCard title="Files Uploaded" value={user?.fileCount} />
        <StatCard title="Member Since" value={formatDate(user?.created_at)} />
      </div>
    </div>
  );
};
```

7. Milestone 5: Social Authentication Integration

7.1 OAuth Provider Setup (Enhanced with Role Assignment)

Provider Configuration with Role Handling

Copilot Prompt: *"Create OAuth configuration for Google Sign-In (primary requirement), Facebook, and LinkedIn providers including client IDs, secrets, scopes, redirect URIs, and automatic 'user' role assignment for new social users."*

7.2 Backend OAuth Implementation with RBAC

Social Authentication with Role Assignment

Copilot Prompt: *"Create FastAPI OAuth endpoints for Google, Facebook, and LinkedIn authentication with proper state validation, token exchange, automatic 'user' role assignment for new users, and role preservation for existing users linking social accounts."*

Social Account Management with Roles

Copilot Prompt: *"Create service for handling social account creation and linking, ensuring new users get default 'user' role and existing users maintain their current role when linking social accounts."*

7.3 Frontend Social Authentication

Social Login UI Components

Copilot Prompt: *"Create React social login components with Google Sign-In button (primary), Facebook and LinkedIn login buttons, proper OAuth flow handling, error management, and role-aware user experience after social login."*

8. Milestone 6: Production Readiness & Advanced Features

8.1 Production Features

Claude-like Dashboard with Role Features

Copilot Prompt: *"Create Claude-like dashboard interface with role-appropriate features, different UI experiences for admin vs regular users, and seamless role-based functionality throughout the interface."*

Advanced User Management for Admins

Copilot Prompt: *"Create comprehensive admin user management interface with user listing, role editing, user search/filtering, bulk operations, and proper confirmation dialogs for role changes."*

8.2 Session Management & Security

Enhanced Session Management

Copilot Prompt: *"Create session management system with secure logout, token invalidation, role-aware session tracking, and proper cleanup of expired sessions with role information."*

9. Enhanced Success Criteria & Quality Gates

Authentication Success Criteria

- User Registration: Complete signup with validation and default 'user' role assignment
- User Login: Secure authentication returning JWT with role claims
- Google SSO: OAuth 2.0 integration working with role assignment
- Password Security: bcrypt hashing with 12+ rounds implemented
- Session Management: Secure token handling and logout functionality

Authorization Success Criteria

- Role-Based Access Control: Complete RBAC with user/admin roles implemented
- Admin Features: Admin dashboard and user management accessible only to admins

- Role Persistence: Roles maintained across login sessions and token refresh
- Access Control: Proper 403 responses for unauthorized role access attempts
- Conditional UI: Frontend renders different interfaces based on user roles