Project 4: User Authentication System

Objective (Why?)

Build a complete user authentication system with registration, login, and protected routes in 4 days. This is your transition from Streamlit to modern full-stack development with React and FastAPI. You will practice:

- Full-Stack Architecture: FastAPI backend + React frontend separation
- Database Management: PostgreSQL with user data and sessions
- Authentication & Security: Password hashing, JWT tokens, protected routes
- Modern Frontend: React + Vite + Tailwind CSS development

Note: For detailed explanations of these concepts, refer to the Project 4 Key Concepts document.

Core Requirements (Must-have)

Layer	Requirement
Backend	FastAPI + PostgreSQL User registration endpoint with validation Login endpoint with JWT token generation Password hashing using bcrypt Protected routes requiring authentication User profile management Database models with SQLAlchemy
Frontend	React + Vite + Tailwind CSS Registration form with validation Login form with error handling Protected dashboard with chat-like layout User profile management page JWT token management and route protection Responsive design with Tailwind CSS
Database	PostgreSQL Users table with proper schema Password storage (hashed) User sessions/token management Database migrations Connection pooling
Security	Password hashing (bcrypt) JWT token authentication Input validation and sanitization Protected API endpoints Secure cookie handling

Development Approach: Milestone-Based Progression

Philosophy: Focus on deliverable quality and comprehensive review compliance rather than rigid timelines. Each milestone must pass all relevant review templates from our Templates folder before proceeding.

Milestone 1: Full-Stack Foundation & Database Setup

Estimated Time: 6-8 hours (flexible based on learning pace)

Deliverables:

- PostgreSQL database setup with proper schema design
- FastAPI backend with SQLAlchemy models and migrations
- React + Vite frontend with Tailwind CSS styling
- Basic API endpoints for user registration and authentication
- Environment configuration and security setup

Review Requirements (Must Pass to Proceed):

- Security Review: Database security, environment setup, input validation
- Architecture Review: Full-stack architecture and component separation
- Code Quality Review: Clean code organization and documentation

Milestone 2: Authentication System & JWT Implementation

Estimated Time: 6-8 hours (flexible based on Milestone 1 completion)

Deliverables:

- Complete JWT authentication system with token management
- Password hashing with bcrypt and security best practices
- Protected API routes with middleware implementation
- React authentication context and route protection
- Login/registration forms with comprehensive validation

Review Requirements (Must Pass to Proceed):

- Security Review: Authentication security, JWT implementation, password handling
- Code Quality Review: Clean authentication patterns and error handling
- Performance Review: Efficient authentication flows and token management

Milestone 3: Dashboard & Production Features

Estimated Time: 4-6 hours (flexible based on previous milestones)

Deliverables:

- Claude-like dashboard interface with responsive design
- User profile management and settings functionality

- Session management and logout functionality
- Comprehensive error handling and user feedback
- Production deployment preparation and documentation

Review Requirements (Must Pass for Project Completion):

- Architecture Review: Complete full-stack architecture assessment
- Security Review: Production security assessment and penetration testing
- Code Quality Review: Production-ready code quality standards
- Performance Review: Frontend/backend performance optimization

Milestone 4: Social Authentication Integration

Estimated Time: 5-7 hours (flexible based on previous milestones)

Deliverables:

- OAuth 2.0 implementation for third-party authentication
- Integration with specific providers:
  - Google Sign-In implementation
  - Facebook Login integration
  - LinkedIn OAuth authentication
- Social account database models and migrations
- Account linking and profile data synchronization
- Social login buttons in frontend UI
- User profile merging strategy

Review Requirements (Must Pass to Proceed):

- Security Review: OAuth implementation, state validation, CSRF protection
- Code Quality Review: Clean provider integration patterns
- User Experience Review: Seamless social login flows and error handling

Milestone Progression Rules:

- Cannot advance to next milestone without passing all review requirements
- Flexible timing allows for learning at individual pace
- Quality gates ensure each milestone meets professional standards

- Mentor support available for concept clarification and review failures

Measurable Goals & Review Template Compliance

Primary Objectives (Must Complete for Project Advancement)

- Security Excellence: Pass Security Review with 9.0/10+ score (critical for authentication)
- Architecture Quality: Pass Architecture Review with 8.5/10+ score
- Full-Stack Integration: Seamless frontend-backend authentication flow
- Performance Standards: Sub-200ms authentication response times
- Code Quality Standards: Pass Code Quality Review with 8.5/10+ score

Review Template Integration (All Must Pass)

Security Review Requirements (Critical for Authentication Systems)

```python
security_criteria = {
    "authentication_security": {"target": 95, "weight": 35},     # JWT, bcrypt, secure flows
    "input_validation": {"target": 90, "weight": 20},          # Comprehensive validation
    "session_management": {"target": 90, "weight": 20},        # Secure session handling
    "data_protection": {"target": 85, "weight": 15},           # Password security, PII
    "vulnerability_prevention": {"target": 85, "weight": 10}     # OWASP compliance
}
```

Architecture Review Requirements

```python
architecture_criteria = {
    "fullstack_design": {"target": 90, "weight": 30},         # Clean frontend-backend separation
```

```
    "api_architecture": {"target": 85, "weight": 25},          # RESTful API design
    "database_design": {"target": 85, "weight": 20},           # Normalized user schema
    "scalability_design": {"target": 80, "weight": 15},        # Multi-user capability
    "component_separation": {"target": 80, "weight": 10}       # Clean component architecture
}
```

Performance Review Requirements

```python
performance_criteria = {
    "auth_response_time": {"target": 95, "weight": 30},        # Fast authentication
    "database_efficiency": {"target": 85, "weight": 25},       # Optimized queries
    "frontend_performance": {"target": 80, "weight": 20},      # Responsive UI
    "concurrent_sessions": {"target": 75, "weight": 15},       # Multi-user support
    "resource_optimization": {"target": 80, "weight": 10}      # Memory/CPU efficiency
}
```

Performance Standards

- Authentication Response: < 200ms for login/registration
- Password Security: Minimum 12 rounds bcrypt hashing
- JWT Security: Secure secret management and expiration
- Database Performance: Indexed queries with sub-50ms response times

Task Tracking & Project Management Integration

Epic: Project 4 - User Authentication System

Epic ID: P4-AUTH
Priority: Critical
Estimated Effort: 16-22 hours

Assignee: [Candidate Name]
Reviewer: [Mentor Name]
Dependencies: Project 3 completion

Milestone 1: Backend Authentication Foundation

Feature 4.1: Core Authentication API

Task ID: P4-M1-API
Priority: Critical
Estimated: 5-7 hours
Dependencies: None

Sub-tasks:

- P4-M1-API-01: Database schema and models
  - Description: User table design with proper constraints
  - Acceptance Criteria: Normalized schema with validation rules
  - Estimated: 90 minutes
- P4-M1-API-02: Password security implementation
  - Description: bcrypt hashing with secure salt rounds
  - Acceptance Criteria: 12+ rounds, secure password policies
  - Estimated: 120 minutes
- P4-M1-API-03: JWT token management
  - Description: Secure token generation and validation
  - Acceptance Criteria: Proper expiration, secret management
  - Estimated: 150 minutes

Milestone 2: Frontend Authentication Interface

Feature 4.2: React Authentication Components

Task ID: P4-M2-FRONTEND
Priority: High

Estimated: 4-6 hours
Dependencies: P4-M1-API completion

Sub-tasks:

- P4-M2-FRONTEND-01: Authentication forms
  - Description: Login, registration, profile forms with validation
  - Acceptance Criteria: Client-side validation, error handling
  - Estimated: 180 minutes
- P4-M2-FRONTEND-02: Protected route system
  - Description: Route guards and authentication state management
  - Acceptance Criteria: Seamless navigation, persistent login
  - Estimated: 120 minutes

Milestone 3: Production Security & Integration

Feature 4.3: Security Hardening

Task ID: P4-M3-SECURITY
Priority: Critical
Estimated: 3-4 hours
Dependencies: P4-M2-FRONTEND completion

Sub-tasks:

- P4-M3-SECURITY-01: Session management
  - Description: Secure logout, token refresh, session persistence
  - Acceptance Criteria: Complete security lifecycle
  - Estimated: 120 minutes
- P4-M3-SECURITY-02: Security testing and validation
  - Description: Authentication flow testing and vulnerability assessment
  - Acceptance Criteria: All security criteria passed
  - Estimated: 90 minutes

Milestone 4: Social Authentication Integration

Feature 4.4: OAuth Provider Integration

Task ID: P4-M4-OAUTH
Priority: Medium
Estimated: 5-7 hours
Dependencies: P4-M3-SECURITY completion

Sub-tasks:

- P4-M4-OAUTH-01: OAuth provider configuration
  - Description: Setup configuration for Google, Facebook, and LinkedIn OAuth
  - Acceptance Criteria: Proper OAuth scopes, redirect URIs, and client credentials
  - Estimated: 90 minutes
- P4-M4-OAUTH-02: Backend OAuth endpoints
  - Description: Implement authorization and callback endpoints for providers
  - Acceptance Criteria: Secure token exchange and user profile retrieval
  - Estimated: 120 minutes
- P4-M4-OAUTH-03: Frontend social login components
  - Description: Social login buttons and OAuth flow handling
  - Acceptance Criteria: Intuitive UI with proper loading and error states
  - Estimated: 90 minutes
- P4-M4-OAUTH-04: Account linking system
  - Description: Link social identities with existing user accounts
  - Acceptance Criteria: Proper handling of duplicate emails across providers
  - Estimated: 90 minutes

Technical Specifications

Database Schema

```sql
-- Users table
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),
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Optional: User sessions table for token blacklisting
CREATE TABLE user_sessions (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    token_jti VARCHAR(255) UNIQUE NOT NULL,
    expires_at TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Social accounts table for OAuth provider linking
CREATE TABLE social_accounts (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) NOT NULL,
    provider VARCHAR(20) NOT NULL,  -- 'google', 'facebook', 'linkedin'
    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)
);
```

Implementation Details: For ORM models and database integration patterns, see Database Design and ORM Integration in the Key Concepts document.

API Endpoints

```python
# POST /auth/register
# POST /auth/login
# GET /auth/me (Protected)
# PUT /auth/profile (Protected)
# GET /auth/login/{provider} (OAuth redirect for Google/Facebook/LinkedIn)
# GET /auth/callback/{provider} (OAuth callback handler)
```

Dashboard Layout Requirements

The dashboard should replicate a Claude-like interface structure:

Left Sidebar (25% width):

- Header with user avatar and name
- "New Chat" button (non-functional placeholder)
- Chat threads list (empty state with placeholder text)
- Settings/Profile access button
- Logout button

Main Content Area (75% width):

- Header with page title
- Welcome message for new users
- Placeholder prompt input area (styled but non-functional)

- Empty state messaging: "Start a new conversation"
- Footer with app information

JWT Token Structure

```python
# Token payload
{
    "sub": "1",  # user_id
    "username": "johndoe",
    "exp": 1642694400,  # expiration timestamp
    "iat": 1642608000,  # issued at
    "jti": "unique-token-id"  # JWT ID for blacklisting
}
```

Security Note: For comprehensive JWT implementation and security best practices, refer to JWT Authentication and Authorization in the Key Concepts document.

Project Structure

- Organize backend with FastAPI, SQLAlchemy models, authentication routes, and middleware
- Structure React frontend with components, hooks, routing, and API service layers
- Implement proper separation of concerns between authentication, UI, and business logic

Stretch Goals (Nice-to-have)

- Email Verification: Send verification emails for new registrations
- Password Reset: Forgot password functionality with email links
- Role-Based Access: Admin/User roles with different permissions
- Two-Factor Authentication: TOTP-based 2FA

- User Settings: Theme preferences, notification settings
- Account Deletion: Self-service account deletion
- Login History: Track user login attempts and sessions
- Account Merging: Merge duplicate accounts when users sign up with both email and social providers
- Additional OAuth Providers: GitHub, Twitter, or Apple Sign-In integration

Deliverables

1. GitHub Repository Link (public or invite @mentor)
2. Live Demo with working registration and login
3. DASHBOARD_PREVIEW.md - Include:
    - Screenshots of registration flow
    - Screenshots of login flow
    - Screenshots of protected dashboard layout
    - Mobile responsive screenshots
    - Sample user credentials for testing
    - Database schema documentation
4. Technical_Learnings.md

Evaluation Rubric (100 Points)

Criterion	Points	Details
Backend Implementation	30 pts	Proper API endpoints and JWT authentication Password hashing and security Database integration and models
Frontend Implementation	25 pts	Clean React components with Vite Tailwind CSS styling Proper form handling and validation

Authentication Flow	20 pts	Registration and login working correctly Protected routes implementation Token management and persistence
Database Design	15 pts	Proper PostgreSQL schema Migrations and data integrity Efficient queries
Code Quality & Security	10 pts	Clean, organized code structure Security best practices Error handling and validation

Security Checklist

- Password Security: Bcrypt hashing with salt
- JWT Security: Proper token expiration and validation
- Input Validation: Server-side validation for all inputs
- SQL Injection Prevention: Parameterized queries with SQLAlchemy
- CORS Configuration: Proper CORS settings for frontend
- Environment Variables: Sensitive data in .env files
- HTTPS Ready: Code prepared for SSL deployment
- Rate Limiting: Consider implementing rate limiting for auth endpoints

Testing Scenarios

Registration Testing

- Valid user registration
- Duplicate email handling
- Duplicate username handling
- Password strength validation

- Invalid email format handling

Login Testing

- Valid credentials login
- Invalid credentials handling
- Non-existent user handling
- JWT token generation and validation
- Token expiration handling

Protected Routes Testing

- Access with valid token
- Access with invalid token
- Access with expired token
- Proper redirection to login

Dashboard Layout Testing

- Left sidebar displays correctly with user info
- Placeholder chat threads show proper empty state
- Main chat area shows welcome message
- "New Chat" button is disabled with coming soon text
- Responsive design works on mobile (collapsible sidebar)
- Settings and logout buttons functional
- Disabled prompt input area displays correctly

Social Login Testing

- Google OAuth login flow
- Facebook OAuth login flow
- LinkedIn OAuth login flow
- Account linking when email matches existing user
- New account creation with social provider
- Social profile data synchronization
- Error handling for declined permissions

- Token refresh for social providers

Quick Start Resources

- FastAPI Security: https://fastapi.tiangolo.com/tutorial/security/
- SQLAlchemy Tutorial: https://docs.sqlalchemy.org/en/14/tutorial/
- React Router: https://reactrouter.com/en/main
- Tailwind CSS: https://tailwindcss.com/docs
- JWT.io: https://jwt.io/ - JWT debugger
- Vite Guide: https://vitejs.dev/guide/

Success Criteria Checklist

- User can register with valid information
- User can login with correct credentials
- Dashboard is accessible only when logged in
- User profile can be viewed and updated
- Proper error messages for invalid inputs
- JWT tokens are properly generated and validated
- Frontend properly handles authentication state
- Database stores user information securely
- Responsive design works on mobile devices
- Code is well-organized and documented