SkillSphere: Real-Time Microlearning & Mentorship Platform

## Project Overview

SkillSphere is a comprehensive full-stack web application that connects learners with mentors for personalized 1-on-1 mentoring sessions. Built as a modern EdTech SaaS platform, it combines real-time communication, AI-powered recommendations, and robust user management to create an engaging learning ecosystem.

## Technical Architecture

## Frontend Stack

- React 18 with functional components and hooks  
- Vite 5 as the build tool and development server  
- Tailwind CSS 3 for responsive, utility-first styling  
- React Router 6 for client-side routing and navigation  
- Axios with interceptors for API communication  
- Stream Chat React for real-time messaging components  
- Agora RTC SDK for video calling functionality

## Backend Stack

- Node.js with Express.js framework  
- MongoDB Atlas with Mongoose ODM  
- JWT for authentication and authorization  
- bcrypt for password hashing  
- Stream Chat for real-time messaging infrastructure  
- Google Gemini AI for intelligent recommendations  
- Agora for video/audio communication  
- Nodemailer for email notifications  
- Passport.js with Google OAuth integration

## Application Workflow

## User Journey & Role-Based Access

1. Learners  
- Register and complete learning profiles with interests and goals  
- Browse and search mentors using advanced filters (skills, ratings, availability)  
- Book mentoring sessions through an intuitive calendar interface  
- Participate in video calls and real-time chat sessions  
- Receive AI-powered mentor recommendations based on learning history  
- Access session transcripts and progress tracking  
  
2. Mentors  
- Apply for mentor status with skills verification (requires admin approval)  
- Set availability schedules and manage booking requests  
- Conduct video sessions with integrated chat and screen sharing  
- Upload session resources and provide post-session notes  
- Access analytics dashboard showing ratings and student progress  
  
3. Administrators  
- Review and approve mentor applications  
- Monitor platform analytics and user engagement metrics  
- Moderate content and handle user reports  
- Export data and generate comprehensive reports

## Core Application Flow - Authentication & Onboarding

The app starts with a comprehensive authentication system supporting both email/password and Google OAuth. New users go through role selection and profile completion, with mentors requiring additional verification steps.

## AuthContext Code Example

const AuthContext = createContext({  
 user: null,  
 login: () => {},  
 logout: () => {},  
 loading: true  
})

## Booking & Session Management

The booking system is the heart of the platform:  
- Discovery: Learners browse mentors through a searchable directory with filters  
- Booking: Calendar-based scheduling with timezone awareness  
- Confirmation: Mentors receive requests and can accept/decline  
- Session Creation: Confirmed bookings automatically generate secure chat channels and video rooms

## Session Creation Code Example

exports.createSession = async (bookingId) => {  
 const booking = await Booking.findById(bookingId).populate('mentorId learnerId');  
 const chatRoomId = `booking\_${bookingId}`;  
 const videoRoomId = `session\_${bookingId}\_${Date.now()}`;  
 await client.upsertUsers([mentorUser, learnerUser]);  
 const channel = client.channel('messaging', chatRoomId, {  
 members: [mentorId, learnerId],  
 session\_date: booking.date,  
 session\_time: booking.time  
 });  
}

## Real-Time Communication

- Text Chat: Stream Chat provides persistent messaging with file sharing  
- Video Calls: Agora RTC enables high-quality video/audio sessions  
- Transcription: Automatic session recording with AI-powered transcripts

## AI Integration

- Mentor Recommendations  
- Session Insights  
- Learning Assistant  
- Progress Tracking

## AI Learning Assistant Code Example

const handleAIQuery = async (message, context) => {  
 const prompt = `  
 User Context: ${context.learnerProfile}  
 Session History: ${context.recentSessions}  
 Question: ${message}  
   
 Provide personalized learning guidance...  
 `;  
   
 const result = await genAI.generateContent(prompt);  
 return result.response.text();  
}

## Key Features Implementation

1. Responsive Design System  
- Dark/light theme switching with localStorage persistence  
- Mobile-first responsive layouts  
- Semantic component library (Button, Card, Input, FormField)  
- Consistent design tokens and spacing  
  
2. Real-Time Features  
- Live Chat: Stream Chat integration with channels scoped to confirmed bookings  
- Video Calls: Agora SDK implementation with mute/video controls  
- Notifications: Real-time booking updates and session reminders  
- Transcript Recording: Speech-to-text during sessions with searchable history  
  
3. Security & Performance  
- JWT-based authentication with automatic token refresh  
- Role-based route protection and API endpoint security  
- Database indexing for optimal query performance  
- Error handling with centralized middleware  
- Input validation and sanitization  
  
4. AI-Powered Enhancements  
- Personalized mentor matching based on learning goals  
- Session summaries with key takeaways and action items  
- Progress analytics with learning pattern recognition  
- Contextual learning assistant with domain expertise

## Database Design

User: { name, email, roles, isApproved, skills, availability }  
Booking: { mentorId, learnerId, date, time, status, message }  
Session: { bookingId, chatRoomId, videoRoomId, duration, notes }  
Transcript: { sessionId, participants, entries, analysis }  
Feedback: { bookingId, rating, comments, isAnonymous }

## Deployment Architecture

- Frontend: Deployed as static assets with CDN integration  
- Backend API: RESTful services with environment-based configuration  
- Database: MongoDB Atlas with automated backups  
- Real-time Services: Stream Chat and Agora as managed services  
- AI Services: Google Gemini API with fallback mechanisms

## Development Approach

- Component-Based Architecture: Modular, reusable React components  
- API-First Design: RESTful endpoints with consistent response formats  
- Environment Configuration: Secure environment variable management  
- Error Handling: Comprehensive error boundaries and user feedback  
- Performance Optimization: Lazy loading, image optimization, and caching

## Scalability Considerations

- Horizontal Scaling: Stateless API design allows easy server scaling  
- Database Optimization: Indexed queries and aggregation pipelines  
- CDN Integration: Static asset delivery optimization  
- Caching Strategy: Redis integration ready for session management  
- Rate Limiting: API protection against abuse and overuse