Multiplayer AI Chat Experience

Project Overview

A real-time multiplayer chat room where multiple users can talk to the same AI character simultaneously. The AI maintains coherent conversations with everyone while keeping track of group dynamics and inter-user conversations.

Challenge Requirements

- **API**: JanitorAI JLLM API (`https://janitorai.com/hackathon/completions`)
- **Authorization**: `calhacks2047`
- **Context Length**: 25,000 tokens
- **Format**: Standard OpenAI chat completions compatible
- **Goal**: Create something fun, original, maybe multimodal, maybe interactive, maybe cute!

Architecture Overview

```
Frontend (React + WebSocket)
    ↕
Backend Orchestrator (Node.js/Express)
    ↕
Context Handler (Message Buffer + User Memory)
    ↕
JLLM API (JanitorAI Hackathon Endpoint)
```

Core Components

1. Frontend - Real-time Chat Interface

Technology Stack:

- React.js with TypeScript
- Socket.io Client for WebSocket connections
- Tailwind CSS for styling
- Framer Motion for animations

Key Features:

- Real-time message display
- User avatars and typing indicators
- AI character personality visualization
- Voice input/output (optional)
- Emoji reactions and mood indicators

2. Backend - Message Orchestrator

Technology Stack:

- Node.js with Express
- Socket.io for WebSocket management
- Redis for session storage
- Rate limiting and message queuing

Core Responsibilities:

- Manage WebSocket connections
- Queue and batch user messages
- Implement AI response timing logic
- Handle context window management
- Broadcast messages to all connected clients

3. Context Management System

Multi-User Prompting Strategy:

```
{
  "messages": [
    {
      "role": "system",
      "content": "You are Nomi, a witty AI character in a shared chatroom. Keep track of user personalities and maintain coherent group conversations."
    },
    {
      "role": "assistant",
      "content": "Group Summary: [Rolling summary of last 10-15 exchanges]"
    },
    {
      "role": "user",
      "name": "Alice",
      "content": "Hey everyone! Just joined the hackathon chat 🚀"
    },
    {
      "role": "user",
      "name": "Bob",
      "content": "@Alice Welcome! We're building something cool with AI"
    },
    {
      "role": "assistant",
      "content": "Welcome Alice! 🎉 Bob's right - this hackathon energy is infectious. What brings you to our AI corner of the internet?"
    }
  ]
}
```

Implementation Plan

Phase 1: Core Infrastructure (2-3 hours)

1. **Backend Setup**
   - Express server with Socket.io
   - JLLM API integration and testing
   - Basic message routing
2. **Frontend Scaffold**
   - React app with Socket.io client
   - Basic chat UI components
   - Message display and input
3. **WebSocket Communication**
   - Real-time message broadcasting
   - User connection/disconnection handling
   - Basic error handling

Phase 2: AI Integration (3-4 hours)

1. **Context Management**
   - User message buffering
   - Group conversation summarization
   - Context window optimization
2. **AI Response Logic**
   - When to respond (timing heuristics)
   - Multi-user context composition
   - Response generation and streaming
3. **Character Personality**
   - System prompt engineering
   - Consistent character voice
   - Memory of user interactions

Phase 3: Enhanced Features (2-3 hours)

1. **Smart Turn-Taking**
   - Detect conversation lulls
   - Handle @mentions and direct questions
   - Avoid interrupting human conversations
2. **Visual Enhancements**
   - Typing indicators

    - AI mood/emotion display
    - Message animations
    - User presence indicators

3. **Advanced Context**

    - Per-user memory storage
    - Topic tracking and transitions
    - Conversation threading

Technical Implementation Details

API Integration

```
// JLLM API Client
const callJLLM = async (messages) => {
  const response = await fetch('https://janitorai.com/hackathon/completions', {
    method: 'POST',
    headers: {
      'Authorization': 'calhacks2047',
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      messages,
      temperature: 0.8,
      max_tokens: 500,
      stream: true
    })
  });

  return response;
};
```

Context Window Management

```
class ContextManager {
  constructor() {
    this.userBuffers = new Map(); // userId -> recent messages
    this.groupSummary = "";
    this.maxTokens = 25000;
  }

  addMessage(userId, message) {
    if (!this.userBuffers.has(userId)) {
      this.userBuffers.set(userId, []);
    }

    this.userBuffers.get(userId).push(message);
```

```
      this.trimIfNeeded();
    }

    buildPrompt() {
      const context = {
        system: this.getSystemPrompt(),
        groupSummary: this.groupSummary,
        recentMessages: this.getRecentMessages(),
        userContext: this.getUserContext()
      };

      return this.formatPrompt(context);
    }
  }
```

AI Response Timing

```
class ResponseScheduler {
  constructor() {
    this.lastAIResponse = Date.now();
    this.messagesSinceAI = 0;
    this.silenceThreshold = 15000; // 15 seconds
  }

  shouldRespond(newMessage) {
    const timeSinceLastAI = Date.now() - this.lastAIResponse;
    this.messagesSinceAI++;

    // Respond if:
    // 1. Directly mentioned
    // 2. 2-3 users have spoken since last AI message
    // 3. 15 seconds of silence
    // 4. Question directed to AI

    return (
      newMessage.content.includes('@Nomi') ||
      this.messagesSinceAI >= 3 ||
      timeSinceLastAI > this.silenceThreshold ||
      this.isQuestionForAI(newMessage)
    );
  }
}
```

Creative Features

1. AI Personality Modes

- **Facilitator**: Guides group discussions
- **Entertainer**: Tells jokes and stories

- **Helper**: Answers technical questions
- **Observer**: Quietly watches, occasional witty comments

2. Visual Character Expression

```
// AI mood indicators
const moodStates = {
  excited: "🤩",
  thoughtful: "🤔",
  playful: "😄",
  curious: "🧐",
  sleepy: "😴"
};
```

3. Memory System

- Remember user preferences and interests
- Recall previous conversations
- Build relationships over time
- Inside jokes and shared references

4. Interactive Elements

- Polls and group decisions
- Mini-games facilitated by AI
- Collaborative storytelling
- Code review sessions

Performance Optimizations

1. Message Batching

- Collect messages within 500ms windows
- Process multiple user inputs together
- Reduce API calls and improve coherence

2. Context Compression

- Summarize old conversations
- Keep only relevant user context
- Efficient token usage within 25k limit

3. Streaming Responses

- Stream AI responses token by token
- Show typing indicators
- Better perceived performance

4. Caching Strategy

- Cache user context summaries
- Redis for session management
- Precompute frequent responses

Deployment Strategy

Development

```
# Backend
cd backend
npm install
npm run dev

# Frontend
cd frontend
npm install
npm run dev
```

Production

- Frontend: Vercel/Netlify
- Backend: Railway/Render
- Database: Redis Cloud
- WebSocket: Socket.io with clustering

Success Metrics

Technical

- **Latency**: < 2 seconds response time
- **Coherence**: AI maintains context across users
- **Scalability**: Support 10+ concurrent users
- **Reliability**: Handle connection drops gracefully

User Experience

- **Engagement**: Users stay and participate
- **Natural Flow**: Conversations feel organic
- **AI Integration**: AI enhances rather than disrupts
- **Fun Factor**: Genuinely enjoyable to use

Risk Mitigation

Technical Risks

- **API Rate Limits**: Implement queuing and batching

- **Context Overflow**: Aggressive summarization
- **WebSocket Issues**: Reconnection logic
- **Scaling**: Use Redis pub/sub for multi-server

Product Risks

- **AI Interruptions**: Smart timing heuristics
- **Boring AI**: Rich personality prompting
- **User Confusion**: Clear UI/UX design
- **Privacy**: No persistent message storage

File Structure

```
multiplayer-ai-chat/
├── backend/
│   ├── src/
│   │   ├── server.js
│   │   ├── api/
│   │   │   └── jllm.js
│   │   ├── services/
│   │   │   ├── contextManager.js
│   │   │   ├── responseScheduler.js
│   │   │   └── messageHandler.js
│   │   └── utils/
│   │       └── tokenCounter.js
│   ├── package.json
│   └── .env
├── frontend/
│   ├── src/
│   │   ├── App.jsx
│   │   ├── components/
│   │   │   ├── ChatRoom.jsx
│   │   │   ├── MessageList.jsx
│   │   │   ├── MessageInput.jsx
│   │   │   └── AICharacter.jsx
│   │   ├── hooks/
│   │   │   └── useSocket.js
│   │   └── utils/
│   │       └── api.js
│   ├── package.json
│   └── vite.config.js
├── docs/
│   ├── api.md
│   └── deployment.md
└── README.md
```

Next Steps

1. **MVP Development**: Build core chat functionality

2. **AI Integration**: Implement JLLM API calls
3. **Context Logic**: Multi-user prompt engineering
4. **UI Polish**: Animations and visual feedback
5. **Testing**: Multi-user scenarios
6. **Demo Preparation**: Showcase compelling use cases

Judging Criteria Alignment

- ✅ **Fun & Original**: Unique multi-user AI interaction
- ✅ **Coherent Conversations**: Advanced context management
- ✅ **Creative Prompting**: Novel multi-user prompt architecture
- ✅ **Interactive Features**: Real-time, engaging experience
- ✅ **Technical Innovation**: Smart timing and context algorithms

---

Prize Target: $200K yearly internship 🎯

This project showcases advanced LLM integration, real-time systems design, and creative user experience - perfect for demonstrating both technical skills and product thinking!