REPORT DETAILING THE DEVELOPMENT PROCESS, THE AI FEATURES USED, AND HOW I INTEGRATED THEM.

Development Process

The development of the "Recipe Room" application followed a standard Node.js and Express.js backend setup, integrated with Socket.io for real-time communication. The process involved:

- **Project Initialization:** Setting up the basic Node.js project structure with *`package.json`* and installing necessary dependencies.
- **Server Setup:** Creating an Express.js server to handle HTTP requests and serve static files.
- **Real-time Communication:** Implementing Socket.io for real-time interactions between users in different "recipe rooms." This included handling user connections, room joining, chat messages, step progression, and timer synchronization.
- **AI Integration:** Incorporating the Google AI (Gemini Pro) API to provide intelligent cooking assistance. This involved making API calls and processing the AI's responses.
- **Environment Configuration:** Utilizing *`.env`* to manage environment variables, specifically for the Google AI API key.
- **CORS Handling:** Configuring CORS to allow cross-origin requests for both the Express server and Socket.io.
- **Dependency Management:** Using *`npm`* for managing project dependencies, as evidenced by *`package.json`* and *`package-lock.json`.*

AI Features Used

The primary AI feature utilized in this application is the **Google Gemini Pro model** for generating cooking advice.

Model:

`gemini-pro`

Endpoint:

`https://generativelanguage.googleapis.com/v1beta/models/gemini-pro:generateContent`

This AI model is used to provide context-aware responses to user questions related to cooking.

AI Integration

The integration of the Google AI feature is handled within the *`server.js`* file, specifically in the *`socket.on('ai-question', async (data) => { ... });`* block.

API Key Management: The ***`GOOGLE_AI_API_KEY`*** is loaded from environment variables using ***`process.env.GOOGLE_AI_API_KEY`.*** A fallback hardcoded key is provided for development purposes, though in a production environment, it should strictly rely on the environment variable.

Javascript:

```
const GOOGLE_AI_API_KEY = process.env.GOOGLE_AI_API_KEY || 'AIzaSyAZnxe66ZIqTWQafOQ1eJUv8DQxfLvD3f4';
```

Contextual Prompt Generation: When a user sends an `ai-question` event, the server constructs a detailed prompt for the AI. This prompt includes:

- The role of the AI (expert cooking assistant).
- Information about the current recipe (name, ingredients, cooking time) if available in the room's state.
- The user's current cooking step.
- The user's specific question.
- Instructions for the AI on the type of advice to provide (techniques, substitutions, timing, troubleshooting) and the desired tone (friendly, practical).

Javascript:

```
const contextPrompt = `You are an expert cooking assistant helping users cook ${currentRecipe?.name || 'a recipe'} in real-time.

Current recipe context:

${currentRecipe ? `

Recipe: ${currentRecipe.name}

Ingredients: ${currentRecipe.ingredients?.join(', ')}

Cooking time: ${currentRecipe.cookingTime} minutes

`: 'No specific recipe selected'}

User is currently on step: ${userCurrentStep}.

User question: ${data.question}
```

API Call: An *`axios.post`* request is made to the *`GOOGLE_AI_ENDPOINT`* with the constructed prompt.

Javascript:

```
const response = await axios.post(`${GOOGLE_AI_ENDPOINT}?key=${GOOGLE_AI_API_KEY}`,
{
 contents: [{
  parts: [{
   text: contextPrompt
  }]
 }]
}, {
 headers: {
  'Content-Type': 'application/json'
 }
});
```

Response Handling: The AI's response is extracted from the API call, formatted as a chat message from "AI Chef 🤖", and then broadcasted to all participants in the user's room via Socket.io.

Javascript:

```
const aiMessage = response.data.candidates[0].content.parts[0].text;
const aiResponse = {
 id: Date.now(),
 username: 'AI Chef 🤖',
 message: aiMessage,
 timestamp: new Date().toLocaleTimeString(),
 type: 'ai'
};
room.messages.push(aiResponse);
io.to(user.roomCode).emit('new-message', aiResponse);
```

Error Handling: Basic error handling is implemented to catch issues with the AI API call and send a user-friendly error message back to the chat.

This integration allows the AI to act as a knowledgeable and helpful assistant, providing real-time, context-aware cooking guidance to users within their collaborative cooking sessions.