Corrección del Simulador de Role Play

Problema Identificado

El simulador de Role Play no avanzaba después de la primera respuesta del usuario. El sistema se quedaba bloqueado y no generaba respuestas del cliente IA para continuar la conversación.

Análisis del Problema

Problemas Encontrados:

1. **Manejo incorrecto del streaming en el frontend**: La lógica de actualización de mensajes durante el streaming era confusa y causaba inconsistencias.
2. **Buffer de streaming mal manejado**: El buffer se estaba manejando de manera incorrecta, causando que las respuestas no se mostraran correctamente.
3. **Lógica de actualización de mensajes inconsistente**: El código intentaba actualizar mensajes existentes de manera incorrecta durante el streaming.
4. **Falta de manejo de errores robusto**: No había suficiente manejo de errores para casos edge.

Correcciones Implementadas

1. Frontend (`components/roleplay-simulator.tsx`)

Función `sendMessage` corregida:

- **Antes**: Lógica confusa para actualizar mensajes durante streaming
- **Después**:
- Crear un mensaje de IA vacío con ID único al inicio
- Actualizar específicamente ese mensaje durante el streaming
- Manejo más limpio del buffer de contenido
- Mejor manejo de errores

```
// Crear mensaje inicial de IA vacío
const initialAiMessage: Message = {
  id: aiMessageId,
  content: '',
  sender: 'cliente_ia',
  timestamp: new Date().toISOString()
};

setMessages(prev => [...prev, initialAiMessage]);

// Actualizar el mensaje específico durante streaming
setMessages(prev => {
  return prev.map(msg =>
    msg.id === aiMessageId
      ? { ...msg, content: aiResponseContent }
      : msg
  );
});
```

Función `startSimulation` mejorada:

- **Antes**: Flujo de inicio confuso
- **Después**:
- Limpiar mensajes anteriores al inicio
- Establecer estado correctamente antes de iniciar
- Mejor manejo de errores

2. Backend (`app/api/roleplay/simulate/route.ts`)

Streaming mejorado:

- **Antes**: Manejo básico del streaming
- **Después**:
- Agregado `[DONE]` explícito al final del stream
- Mejor logging de errores
- Campo `totalContent` para debugging
- Manejo más robusto de errores de parsing

```
controller.enqueue(encoder.encode(`data: ${finalData}\n\n`));
controller.enqueue(encoder.encode(`data: [DONE]\n\n`));
```

3. Manejo de Errores Mejorado

- Agregado manejo específico para errores de API
- Mejor logging para debugging
- Limpieza de mensajes vacíos en caso de error
- Mensajes de error más descriptivos para el usuario

Flujo Corregido

1. Inicio de Simulación:

1. Usuario selecciona escenario y hace clic en “Comenzar Simulación”
2. Se limpia el estado anterior

3. Se muestra mensaje de bienvenida
4. Se inicia la primera respuesta del cliente IA

2. Intercambio de Mensajes:

1. Usuario escribe y envía mensaje
2. Se agrega mensaje del usuario a la conversación
3. Se crea mensaje vacío del cliente IA
4. Se inicia streaming de respuesta
5. Se actualiza el mensaje del IA en tiempo real
6. Se completa la respuesta y se guarda en BD

3. Continuación:

1. El usuario puede enviar otro mensaje inmediatamente
2. El proceso se repite sin bloqueos
3. La conversación fluye naturalmente

Archivos Modificados

1. `components/roleplay-simulator.tsx`
   - Función `sendMessage`: Lógica de streaming corregida
   - Función `startSimulation`: Flujo de inicio mejorado
   - Manejo de errores mejorado

2. `app/api/roleplay/simulate/route.ts`
   - Streaming backend mejorado
   - Mejor manejo de errores
   - Logging mejorado

Pruebas

Se creó un script de prueba (`test-roleplay-fix.js`) que verifica:
- Inicio correcto de simulación
- Envío de primer mensaje
- Envío de segundo mensaje (verificar que no se bloquea)
- Finalización de sesión

Para ejecutar las pruebas:

```
# Asegúrate de que el servidor esté corriendo
npm run dev

# En otra terminal, ejecuta las pruebas
node test-roleplay-fix.js
```

Verificación Manual

1. Ir a `/dashboard/roleplay`
2. Seleccionar un escenario
3. Hacer clic en “Comenzar Simulación”

4. Enviar un mensaje como vendedor
5. Verificar que el cliente IA responde
6. Enviar otro mensaje
7. Verificar que la conversación continúa sin bloqueos

Beneficios de las Correcciones

1. **Flujo de conversación continuo**: Ya no se bloquea después del primer mensaje
2. **Mejor experiencia de usuario**: Streaming en tiempo real más fluido
3. **Manejo de errores robusto**: Mejor recuperación ante fallos
4. **Código más mantenible**: Lógica más clara y fácil de entender
5. **Debugging mejorado**: Mejor logging para identificar problemas futuros

Notas Técnicas

- Las correcciones mantienen compatibilidad con el código existente
- No se requieren cambios en la base de datos
- El streaming sigue funcionando en tiempo real
- Se mantiene la funcionalidad de evaluación automática
- Compatible con todos los escenarios existentes

Próximos Pasos Recomendados

1. Probar con diferentes escenarios
2. Verificar en diferentes navegadores
3. Probar con conexiones lentas
4. Considerar agregar tests automatizados
5. Monitorear logs de producción para identificar edge cases