🧪 Guía de Entrenamiento y Experimentación Local

📋 Tabla de Contenidos

🚀 Setup Inicial Rápido

1. Instalación (5 minutos)

```bash
# Clonar estructura
mkdir restaurant-chatbot-saas && cd restaurant-chatbot-saas

# Inicializar proyecto
npm init -y

# Instalar dependencias core
npm install express mongoose bcryptjs jsonwebtoken cors dotenv @anthropic-ai/sdk

# Herramientas de desarrollo
npm install nodemon --save-dev
```

2. Estructura de Carpetas

```bash
mkdir -p backend/{models,routes,middleware,config}
mkdir -p frontend/{landing,admin,widget}
mkdir -p tests
mkdir -p logs
```

3. Variables de Entorno

Crear archivo `.env`:

```env
# MongoDB
MONGODB_URI=mongodb://localhost:27017/restaurant-chatbot

# API Keys
ANTHROPIC_API_KEY=sk-ant-tu-api-key-aqui

# JWT
JWT_SECRET=super-secret-key-change-in-production

# Server
PORT=5000
NODE_ENV=development

# Logging
LOG_LEVEL=debug
```

4. Inicializar Base de Datos

```bash
# Ejecutar seed
node seed.js

# Verificar datos
mongosh restaurant-chatbot
> db.restaurants.find().pretty()
> db.users.find().pretty()
```

🧠 Entrenamiento del Chatbot

Estrategia de Entrenamiento

El chatbot usa **prompt engineering** con Claude, no entrenamiento tradicional. Se optimiza mediante:

1. **System Prompts** - Instrucciones base
2. **Few-shot Learning** - Ejemplos en contexto

3. **Contextual Memory** - Historial de conversación

4. **Dynamic Context** - Datos en tiempo real

Archivo: **backend/training/prompt-templates.js**

javascript

```
const promptTemplates = {
  // Template base para todos los restaurantes
  base: `Eres {chatbotName}, asistente virtual de {restaurantName}.

CONTEXTO DEL RESTAURANTE:
{restaurantContext}

MENÚ DISPONIBLE:
{menuContext}

PERSONALIDAD:
- Amigable y profesional
- Respuestas concisas (2-3 líneas máximo)
- Usa emojis apropiadamente
- Adapta el tono al tipo de restaurante

CAPACIDADES:
{capabilities}

INSTRUCCIONES:
1. Saluda calurosamente
2. Ayuda a elegir del menú
3. Confirma detalles del pedido
4. Solicita información de contacto cuando sea necesario
5. Resuelve dudas sobre el restaurante`,

  // Template específico para pedidos
  orderTaking: `MODO: Toma de pedidos

PROCESO:
1. Pregunta qué desea ordenar
2. Confirma cada item y precio
3. Sugiere complementos (bebidas, postres)
4. Calcula total
5. Solicita dirección y forma de pago
6. Confirma tiempo estimado de entrega

EJEMPLO:
Usuario: "Quiero una pizza"
Asistente: "¡Perfecto! Tenemos Margarita ($28.000), Pepperoni ($32.000), etc. ¿Cuál prefieres?"`,

  // Template para reservaciones
  reservations: `MODO: Sistema de reservaciones
```

```
INFORMACIÓN REQUERIDA:
- Nombre completo
- Número de personas
- Fecha y hora preferida
- Teléfono de contacto
- Ocasión especial (opcional)

PROCESO:
1. Preguntar fecha y hora
2. Verificar disponibilidad
3. Solicitar datos del cliente
4. Confirmar reservación
5. Enviar recordatorio`,

  // Template para consultas generales
  information: `MODO: Información general

TEMAS A CUBRIR:
- Ubicación y direcciones
- Horarios de atención
- Métodos de pago
- Política de entregas
- Políticas de cancelación
- Menú y precios
- Promociones actuales`
};

// Función para generar prompt dinámico
function generatePrompt(restaurant, mode = 'base') {
  const template = promptTemplates[mode] || promptTemplates.base;

  return template
    .replace('{chatbotName}', restaurant.chatbotName)
    .replace('{restaurantName}', restaurant.restaurantName)
    .replace('{restaurantContext}', generateRestaurantContext(restaurant))
    .replace('{menuContext}', generateMenuContext(restaurant))
    .replace('{capabilities}', generateCapabilities(restaurant));
}

function generateRestaurantContext(restaurant) {
  return `
Nombre: ${restaurant.restaurantName}
Teléfono: ${restaurant.phone}
```

```
Dirección: ${restaurant.address}
Horario: ${restaurant.businessHours}
`.trim();
}

function generateMenuContext(restaurant) {
  return restaurant.menuItems
    .filter(item => item.available)
    .map(item => `- ${item.name}: $${item.price.toLocaleString()} ${item.description ? `(${item.description})` : ''}`)
    .join('\n');
}

function generateCapabilities(restaurant) {
  const caps = [];
  if (restaurant.enableDelivery) caps.push('✓ Pedidos a domicilio');
  if (restaurant.enableReservations) caps.push('✓ Reservaciones de mesa');
  if (restaurant.enableWhatsApp) caps.push('✓ Atención por WhatsApp');
  return caps.join('\n');
}

module.exports = { promptTemplates, generatePrompt };
```

Experimentación con Prompts

Crear archivo: `tests/prompt-experiments.js`

```javascript
```

```
const { generatePrompt } = require('../backend/training/prompt-templates');
const Anthropic = require('@anthropic-ai/sdk');
require('dotenv').config();

const anthropic = new Anthropic({
  apiKey: process.env.ANTHROPIC_API_KEY
});

// Casos de prueba
const testCases = [
  {
    name: 'Pedido Simple',
    userMessage: 'Quiero una pizza grande',
    expected: 'Debe preguntar qué tipo de pizza'
  },
  {
    name: 'Consulta de Precio',
    userMessage: '¿Cuánto cuesta la hamburguesa?',
    expected: 'Debe dar el precio exacto'
  },
  {
    name: 'Reservación',
    userMessage: 'Quiero reservar para 4 personas',
    expected: 'Debe preguntar fecha y hora'
  },
  {
    name: 'Consulta de Horario',
    userMessage: '¿A qué hora abren?',
    expected: 'Debe dar el horario exacto'
  },
  {
    name: 'Pedido Complejo',
    userMessage: 'Quiero 2 pizzas margarita, 1 pepperoni y 3 coca-colas',
    expected: 'Debe listar todos los items con precios'
  }
];

// Restaurante de prueba
const testRestaurant = {
  restaurantName: 'La Pizzería Test',
  chatbotName: 'TestBot',
  phone: '+57 300 123 4567',
  address: 'Calle 123',
```

```
  businessHours: '9AM - 10PM',
  enableDelivery: true,
  enableReservations: true,
  menuItems: [
    { name: 'Pizza Margarita', price: 28000, description: 'Clásica', available: true },
    { name: 'Pizza Pepperoni', price: 32000, description: 'Con pepperoni', available: true },
    { name: 'Coca-Cola', price: 4000, description: '400ml', available: true }
  ]
};

async function runExperiment(testCase, promptVersion) {
  console.log(`\n${'='.repeat(60)}`);
  console.log(`Prueba: ${testCase.name}`);
  console.log(`Entrada: "${testCase.userMessage}"`);
  console.log(`Esperado: ${testCase.expected}`);
  console.log(`${'='.repeat(60)}\n`);

  const systemPrompt = generatePrompt(testRestaurant, promptVersion);

  try {
    const response = await anthropic.messages.create({
      model: 'claude-sonnet-4-20250514',
      max_tokens: 1024,
      system: systemPrompt,
      messages: [
        { role: 'user', content: testCase.userMessage }
      ]
    });

    const botResponse = response.content[0].text;
    console.log(`Respuesta del Bot:\n${botResponse}\n`);

    // Análisis básico
    const analysis = analyzeResponse(botResponse, testCase);
    console.log(`Análisis: ${analysis.pass ? '✅ PASS' : '❌ FAIL'}`);
    console.log(`Detalles: ${analysis.details}\n`);

    return { testCase, response: botResponse, analysis };
  } catch (error) {
    console.error('Error:', error.message);
    return { testCase, error: error.message };
  }
}
```

```
function analyzeResponse(response, testCase) {
  const analysis = {
    pass: false,
    details: ''
  };

  // Análisis simple basado en keywords
  const lowerResponse = response.toLowerCase();

  switch (testCase.name) {
    case 'Pedido Simple':
      analysis.pass = lowerResponse.includes('tipo') || lowerResponse.includes('cuál');
      analysis.details = analysis.pass ? 'Pregunta correctamente' : 'No pregunta por tipo';
      break;

    case 'Consulta de Precio':
      analysis.pass = lowerResponse.includes('28000') || lowerResponse.includes('28.000');
      analysis.details = analysis.pass ? 'Precio correcto' : 'Precio incorrecto o ausente';
      break;

    case 'Reservación':
      analysis.pass = (lowerResponse.includes('fecha') || lowerResponse.includes('hora'));
      analysis.details = analysis.pass ? 'Solicita fecha/hora' : 'No solicita información necesaria';
      break;

    case 'Consulta de Horario':
      analysis.pass = lowerResponse.includes('9') && lowerResponse.includes('10');
      analysis.details = analysis.pass ? 'Horario correcto' : 'Horario incorrecto';
      break;

    default:
      analysis.pass = response.length > 20;
      analysis.details = 'Respuesta generada';
  }

  return analysis;
}

async function runAllTests() {
  console.log('🧪 Iniciando experimentos de prompts...\n');

  const results = [];

  for (const testCase of testCases) {
```

```
    const result = await runExperiment(testCase, 'base');
    results.push(result);

    // Esperar un poco entre llamadas
    await new Promise(resolve => setTimeout(resolve, 1000));
  }

  // Resumen
  console.log('\n📊 RESUMEN DE RESULTADOS');
  console.log('='.repeat(60));

  const passed = results.filter(r => r.analysis?.pass).length;
  const total = results.length;

  console.log(`\nTests pasados: ${passed}/${total} (${(passed/total*100).toFixed(1)}%)\n`);

  results.forEach((r, i) => {
    const status = r.analysis?.pass ? '✅' : '❌';
    console.log(`${i + 1}. ${status} ${r.testCase.name}`);
  });

  console.log('\n');
}

// Ejecutar
runAllTests().catch(console.error);
```

🔬 Testing y Optimización

Archivo: `tests/conversation-simulator.js`

```
javascript
```

```
// Simulador de conversaciones completas
const Anthropic = require('@anthropic-ai/sdk');
const { generatePrompt } = require('../backend/training/prompt-templates');

class ConversationSimulator {
  constructor(restaurant, apiKey) {
    this.restaurant = restaurant;
    this.anthropic = new Anthropic({ apiKey });
    this.conversationHistory = [];
  }

  async startConversation() {
    const welcomeMsg = this.restaurant.welcomeMessage;
    this.conversationHistory.push({
      role: 'assistant',
      content: welcomeMsg
    });
    return welcomeMsg;
  }

  async sendMessage(userMessage) {
    // Agregar mensaje del usuario
    this.conversationHistory.push({
      role: 'user',
      content: userMessage
    });

    // Generar respuesta
    const systemPrompt = generatePrompt(this.restaurant);

    const response = await this.anthropic.messages.create({
      model: 'claude-sonnet-4-20250514',
      max_tokens: 1024,
      system: systemPrompt,
      messages: this.conversationHistory
    });

    const botResponse = response.content[0].text;

    // Agregar respuesta al historial
    this.conversationHistory.push({
      role: 'assistant',
      content: botResponse
```

```
    });

    return botResponse;
  }

  getHistory() {
    return this.conversationHistory;
  }

  exportHistory() {
    return JSON.stringify(this.conversationHistory, null, 2);
  }
}

// Escenarios de prueba
async function simulateOrderScenario() {
  console.log('🍕 Simulando escenario: Pedido completo\n');

  const restaurant = {
    restaurantName: 'Test Restaurant',
    chatbotName: 'TestBot',
    welcomeMessage: '¡Hola! ¿En qué puedo ayudarte?',
    phone: '+57 300 123 4567',
    address: 'Calle 123',
    businessHours: '9AM-10PM',
    enableDelivery: true,
    menuItems: [
      { name: 'Pizza', price: 28000, available: true },
      { name: 'Coca-Cola', price: 4000, available: true }
    ]
  };

  const simulator = new ConversationSimulator(
    restaurant,
    process.env.ANTHROPIC_API_KEY
  );

  // Iniciar conversación
  console.log('Bot:', await simulator.startConversation());

  // Secuencia de mensajes
  const messages = [
    'Hola',
    'Quiero una pizza',
```

```
    'La margarita',
    'Sí, una coca-cola también',
    'Mi dirección es Calle 45 #12-34',
    'Pago con tarjeta'
  ];

  for (const msg of messages) {
    console.log(`\nUsuario: ${msg}`);
    const response = await simulator.sendMessage(msg);
    console.log(`Bot: ${response}`);

    // Pausa entre mensajes
    await new Promise(resolve => setTimeout(resolve, 1500));
  }

  // Exportar historial
  const history = simulator.exportHistory();
  require('fs').writeFileSync(
    `./logs/conversation_${Date.now()}.json`,
    history
  );

  console.log('\n✅ Conversación completada y guardada');
}

module.exports = { ConversationSimulator, simulateOrderScenario };
```

📊 Métricas y Análisis

Archivo: `backend/analytics/metrics.js`

```
javascript
```

```
// Sistema de métricas para evaluar calidad del chatbot

class ChatbotMetrics {
  constructor() {
    this.metrics = {
      totalConversations: 0,
      completedOrders: 0,
      averageMessageCount: 0,
      averageResponseTime: 0,
      satisfactionScore: 0,
      commonQuestions: {},
      failedIntents: []
    };
  }

  recordConversation(conversation) {
    this.metrics.totalConversations++;

    if (conversation.orderPlaced) {
      this.metrics.completedOrders++;
    }

    const messageCount = conversation.messages.length;
    this.metrics.averageMessageCount =
      (this.metrics.averageMessageCount * (this.metrics.totalConversations - 1) + messageCount)
      / this.metrics.totalConversations;

    // Analizar intenciones
    conversation.messages
      .filter(m => m.role === 'user')
      .forEach(m => this.detectIntent(m.content));
  }

  detectIntent(message) {
    const intents = {
      'menu': ['menú', 'qué tienen', 'opciones', 'platos'],
      'order': ['quiero', 'pedir', 'ordenar'],
      'price': ['cuánto', 'precio', 'cuesta'],
      'hours': ['horario', 'abren', 'cierran'],
      'location': ['dónde', 'dirección', 'ubicación']
    };

    const lowerMsg = message.toLowerCase();
```

```
  for (const [intent, keywords] of Object.entries(intents)) {
    if (keywords.some(kw => lowerMsg.includes(kw))) {
      this.metrics.commonQuestions[intent] =
        (this.metrics.commonQuestions[intent] || 0) + 1;
      return;
    }
  }

  // Si no matchea, es intención desconocida
  this.metrics.failedIntents.push(message);
}

getConversionRate() {
  return this.metrics.totalConversations > 0
    ? (this.metrics.completedOrders / this.metrics.totalConversations * 100).toFixed(2)
    : 0;
}

getTopQuestions(limit = 5) {
  return Object.entries(this.metrics.commonQuestions)
    .sort(([,a], [,b]) => b - a)
    .slice(0, limit);
}

generateReport() {
  return {
    summary: {
      totalConversations: this.metrics.totalConversations,
      completedOrders: this.metrics.completedOrders,
      conversionRate: this.getConversionRate() + '%',
      avgMessages: this.metrics.averageMessageCount.toFixed(1)
    },
    topQuestions: this.getTopQuestions(),
    failedIntents: this.metrics.failedIntents.slice(-10),
    recommendations: this.generateRecommendations()
  };
}

generateRecommendations() {
  const recs = [];

  if (parseFloat(this.getConversionRate()) < 50) {
    recs.push('Baja tasa de conversión. Revisa el flujo de pedidos.');
```

```
    }

    if (this.metrics.failedIntents.length > 10) {
      recs.push('Muchas intenciones no detectadas. Amplía el vocabulario del bot.');
    }

    if (this.metrics.averageMessageCount > 15) {
      recs.push('Conversaciones muy largas. Simplifica el proceso.');
    }

    return recs;
  }
}

module.exports = ChatbotMetrics;
```

🎯 Casos de Uso Avanzados

1. A/B Testing de Prompts

javascript

```javascript
// tests/ab-testing.js
const variants = {
  A: {
    tone: 'formal',
    emojiUse: 'minimal',
    responseLength: 'short'
  },
  B: {
    tone: 'casual',
    emojiUse: 'frequent',
    responseLength: 'medium'
  }
};

async function runABTest(testCases) {
  const results = { A: [], B: [] };

  for (const variant of ['A', 'B']) {
    for (const testCase of testCases) {
      const result = await testWithVariant(testCase, variants[variant]);
      results[variant].push(result);
    }
  }

  return analyzeABResults(results);
}
```

2. Entrenamiento Continuo

```javascript
// Guardar conversaciones exitosas como ejemplos
async function saveSuccessfulConversation(conversation) {
  if (conversation.orderPlaced && conversation.customerFeedback > 4) {
    // Guardar en base de datos de ejemplos
    await TrainingExample.create({
      messages: conversation.messages,
      outcome: 'success',
      tags: ['order_completed', 'high_satisfaction']
    });
  }
}
```

3. Personalización por Restaurante

javascript

```javascript
// Cada restaurante puede tener su propio "estilo"
const restaurantStyles = {
  formal: {
    tone: 'professional',
    greeting: 'Buenos días, ¿en qué puedo asistirle?',
    farewell: 'Gracias por su preferencia'
  },
  casual: {
    tone: 'friendly',
    greeting: '¡Hola! ¿Qué se te antoja hoy?',
    farewell: '¡Que disfrutes tu comida!'
  },
  youthful: {
    tone: 'energetic',
    greeting: '¡Hey! 🎉 ¿Listo para ordenar?',
    farewell: '¡Nos vemos! 🍕'
  }
};
```

🚀 Script de Prueba Completo

bash

```
#!/bin/bash
# run-experiments.sh

echo "🧪 Iniciando suite de experimentos..."

# 1. Pruebas de prompts
echo "1️⃣ Ejecutando tests de prompts..."
node tests/prompt-experiments.js

# 2. Simulación de conversaciones
echo "2️⃣ Simulando conversaciones..."
node tests/conversation-simulator.js

# 3. Análisis de métricas
echo "3️⃣ Generando reportes de métricas..."
node backend/analytics/generate-report.js

# 4. A/B Testing
echo "4️⃣ Ejecutando A/B tests..."
node tests/ab-testing.js

echo "✅ Todos los experimentos completados!"
echo "📊 Revisa los resultados en ./logs/"
```

📝 Checklist de Optimización

- [ ] Probar con 20+ conversaciones reales
- [ ] Medir tiempos de respuesta
- [ ] Evaluar tasa de conversión
- [ ] Identificar preguntas frecuentes
- [ ] Documentar casos edge
- [ ] Optimizar longitud de prompts
- [ ] A/B test de variantes
- [ ] Recopilar feedback de usuarios
- [ ] Iterar basado en métricas
- [ ] Documentar mejores prácticas

¿Listo para experimentar? Ejecuta `npm run experiments` y comienza a optimizar tu chatbot! 🚀