🚀 Sistema SaaS ChatBot para Restaurantes - Setup Completo

📋 Estructura del Proyecto

```
restaurant-chatbot-saas/
├── backend/
│   ├── server.js              # Servidor principal Node.js
│   ├── config/
│   │   ├── database.js        # Configuración MongoDB
│   │   └── anthropic.js       # Config API Anthropic
│   ├── models/
│   │   ├── Restaurant.js      # Modelo de restaurante
│   │   ├── Conversation.js    # Modelo de conversaciones
│   │   └── User.js            # Modelo de usuarios
│   ├── routes/
│   │   ├── auth.js            # Autenticación
│   │   ├── chatbot.js         # Endpoints del chatbot
│   │   └── admin.js           # Panel administrativo
│   └── middleware/
│       └── auth.js            # Middleware autenticación
├── frontend/
│   ├── landing/               # Landing page
│   ├── admin/                 # Panel administrativo
│   └── widget/                # Widget del chatbot
└── .env                       # Variables de entorno
```

🛠️ Instalación Paso a Paso

1. Prerrequisitos

bash

```
# Node.js (v18 o superior)
node --version

# MongoDB (local o Atlas)
# https://www.mongodb.com/try/download/community

# Git
git --version
```

2. Clonar e Instalar

```bash
# Crear proyecto
mkdir restaurant-chatbot-saas
cd restaurant-chatbot-saas

# Inicializar npm
npm init -y

# Instalar dependencias
npm install express mongoose bcryptjs jsonwebtoken cors dotenv
npm install @anthropic-ai/sdk
npm install nodemon --save-dev
```

3. Estructura de Archivos

backend/server.js

```javascript
```

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
require('dotenv').config();

const app = express();

// Middleware
app.use(cors());
app.use(express.json());

// Rutas
app.use('/api/auth', require('./routes/auth'));
app.use('/api/chatbot', require('./routes/chatbot'));
app.use('/api/admin', require('./routes/admin'));

// Conexión MongoDB
mongoose.connect(process.env.MONGODB_URI || 'mongodb://localhost:27017/restaurant-chatbot')
  .then(() => console.log('✅ MongoDB conectado'))
  .catch(err => console.error('❌ Error MongoDB:', err));

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`🚀 Servidor corriendo en http://localhost:${PORT}`);
});
```

backend/models/Restaurant.js

javascript

```
const mongoose = require('mongoose');

const menuItemSchema = new mongoose.Schema({
  name: { type: String, required: true },
  category: String,
  price: { type: Number, required: true },
  description: String,
  available: { type: Boolean, default: true }
});

const restaurantSchema = new mongoose.Schema({
  ownerEmail: { type: String, required: true, unique: true },
  restaurantName: { type: String, required: true },
  phone: String,
  address: String,
  businessHours: String,

  // Configuración del chatbot
  chatbotName: { type: String, default: 'RestauBot' },
  welcomeMessage: String,
  primaryColor: { type: String, default: '#f97316' },

  // Funcionalidades
  enableReservations: { type: Boolean, default: true },
  enableDelivery: { type: Boolean, default: true },
  enableWhatsApp: { type: Boolean, default: false },
  whatsappNumber: String,

  // API Keys
  anthropicApiKey: String,

  // Menú
  menuItems: [menuItemSchema],

  // Suscripción
  plan: { type: String, enum: ['basic', 'pro', 'enterprise'], default: 'basic' },
  subscriptionStatus: { type: String, default: 'active' },

  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now }
});
```

```
module.exports = mongoose.model('Restaurant', restaurantSchema);
```

backend/models/User.js

javascript

```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const userSchema = new mongoose.Schema({
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  restaurantId: { type: mongoose.Schema.Types.ObjectId, ref: 'Restaurant' },
  role: { type: String, enum: ['owner', 'admin'], default: 'owner' },
  createdAt: { type: Date, default: Date.now }
});

// Hash password antes de guardar
userSchema.pre('save', async function(next) {
  if (!this.isModified('password')) return next();
  this.password = await bcrypt.hash(this.password, 10);
  next();
});

// Método para comparar passwords
userSchema.methods.comparePassword = async function(candidatePassword) {
  return await bcrypt.compare(candidatePassword, this.password);
};

module.exports = mongoose.model('User', userSchema);
```

backend/models/Conversation.js

javascript

```
const mongoose = require('mongoose');

const messageSchema = new mongoose.Schema({
  role: { type: String, enum: ['user', 'assistant'], required: true },
  content: { type: String, required: true },
  timestamp: { type: Date, default: Date.now }
});

const conversationSchema = new mongoose.Schema({
  restaurantId: { type: mongoose.Schema.Types.ObjectId, ref: 'Restaurant', required: true },
  sessionId: { type: String, required: true },
  customerPhone: String,
  customerName: String,
  messages: [messageSchema],
  status: { type: String, enum: ['active', 'completed', 'abandoned'], default: 'active' },
  orderPlaced: { type: Boolean, default: false },
  orderTotal: Number,
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Conversation', conversationSchema);
```

backend/routes/auth.js

```
javascript
```

```
const express = require('express');
const router = express.Router();
const jwt = require('jsonwebtoken');
const User = require('../models/User');
const Restaurant = require('../models/Restaurant');

// Registro
router.post('/register', async (req, res) => {
  try {
    const { email, password, restaurantName } = req.body;

    // Verificar si usuario existe
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(400).json({ error: 'Email ya registrado' });
    }

    // Crear restaurante
    const restaurant = new Restaurant({
      ownerEmail: email,
      restaurantName,
      welcomeMessage: `¡Hola! Soy el asistente de ${restaurantName}. ¿En qué puedo ayudarte?`
    });
    await restaurant.save();

    // Crear usuario
    const user = new User({
      email,
      password,
      restaurantId: restaurant._id
    });
    await user.save();

    // Generar token
    const token = jwt.sign(
      { userId: user._id, restaurantId: restaurant._id },
      process.env.JWT_SECRET || 'secret-key',
      { expiresIn: '7d' }
    );

    res.json({
      token,
      user: { email: user.email },
```

```
      restaurant: { id: restaurant._id, name: restaurant.restaurantName }
    });
  } catch (error) {
    console.error('Error en registro:', error);
    res.status(500).json({ error: 'Error en el servidor' });
  }
});

// Login
router.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body;

    // Buscar usuario
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(401).json({ error: 'Credenciales inválidas' });
    }

    // Verificar password
    const isMatch = await user.comparePassword(password);
    if (!isMatch) {
      return res.status(401).json({ error: 'Credenciales inválidas' });
    }

    // Generar token
    const token = jwt.sign(
      { userId: user._id, restaurantId: user.restaurantId },
      process.env.JWT_SECRET || 'secret-key',
      { expiresIn: '7d' }
    );

    res.json({
      token,
      user: { email: user.email },
      restaurantId: user.restaurantId
    });
  } catch (error) {
    console.error('Error en login:', error);
    res.status(500).json({ error: 'Error en el servidor' });
  }
});
```

```
module.exports = router;
```

backend/routes/chatbot.js

```
javascript
```

```
const express = require('express');
const router = express.Router();
const Anthropic = require('@anthropic-ai/sdk');
const Restaurant = require('../models/Restaurant');
const Conversation = require('../models/Conversation');

// Endpoint del chatbot (público)
router.post('/message', async (req, res) => {
  try {
    const { restaurantId, sessionId, message, userMessage } = req.body;

    // Obtener configuración del restaurante
    const restaurant = await Restaurant.findById(restaurantId);
    if (!restaurant) {
      return res.status(404).json({ error: 'Restaurante no encontrado' });
    }

    // Buscar o crear conversación
    let conversation = await Conversation.findOne({ restaurantId, sessionId });
    if (!conversation) {
      conversation = new Conversation({
        restaurantId,
        sessionId,
        messages: []
      });
    }

    // Agregar mensaje del usuario
    conversation.messages.push({
      role: 'user',
      content: userMessage || message
    });

    // Preparar contexto del menú
    const menuContext = restaurant.menuItems
      .filter(item => item.available)
      .map(item => `- ${item.name}: $${item.price.toLocaleString()} ${item.description ? `(${item.description})` : ''}`)
      .join('\n');

    // Llamar a Anthropic
    const anthropic = new Anthropic({
      apiKey: restaurant.anthropicApiKey || process.env.ANTHROPIC_API_KEY
    });
```

```
    const response = await anthropic.messages.create({
      model: 'claude-sonnet-4-20250514',
      max_tokens: 1024,
      system: `Eres ${restaurant.chatbotName}, asistente virtual de ${restaurant.restaurantName}.

INFORMACIÓN DEL RESTAURANTE:
- Teléfono: ${restaurant.phone}
- Dirección: ${restaurant.address}
- Horario: ${restaurant.businessHours}

MENÚ DISPONIBLE:
${menuContext}

FUNCIONALIDADES HABILITADAS:
${restaurant.enableDelivery ? '✓ Pedidos a domicilio' : ''}
${restaurant.enableReservations ? '✓ Reservaciones' : ''}

INSTRUCCIONES:
- Sé amigable y profesional
- Ayuda al cliente a hacer pedidos
- Responde preguntas sobre el menú
- Confirma detalles antes de procesar
- Si necesitan reservación o domicilio, solicita datos de contacto
- Respuestas concisas (máximo 3 líneas)`,
      messages: conversation.messages.map(msg => ({
        role: msg.role,
        content: msg.content
      }))
    });

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

    // Guardar respuesta
    conversation.messages.push({
      role: 'assistant',
      content: assistantMessage
    });
    conversation.updatedAt = new Date();
    await conversation.save();

    res.json({
      message: assistantMessage,
      conversationId: conversation._id
```

```
    });

  } catch (error) {
    console.error('Error en chatbot:', error);
    res.status(500).json({ error: 'Error al procesar mensaje' });
  }
});

// Obtener historial de conversación
router.get('/conversation/:sessionId', async (req, res) => {
  try {
    const { sessionId } = req.params;
    const { restaurantId } = req.query;

    const conversation = await Conversation.findOne({ restaurantId, sessionId });
    if (!conversation) {
      return res.json({ messages: [] });
    }

    res.json({ messages: conversation.messages });
  } catch (error) {
    console.error('Error al obtener conversación:', error);
    res.status(500).json({ error: 'Error en el servidor' });
  }
});

module.exports = router;
```

backend/routes/admin.js

```
javascript
```

```
const express = require('express');
const router = express.Router();
const Restaurant = require('../models/Restaurant');
const Conversation = require('../models/Conversation');
const authMiddleware = require('../middleware/auth');

// Aplicar middleware de autenticación a todas las rutas
router.use(authMiddleware);

// Obtener configuración del restaurante
router.get('/config', async (req, res) => {
  try {
    const restaurant = await Restaurant.findById(req.restaurantId);
    if (!restaurant) {
      return res.status(404).json({ error: 'Restaurante no encontrado' });
    }
    res.json(restaurant);
  } catch (error) {
    console.error('Error al obtener config:', error);
    res.status(500).json({ error: 'Error en el servidor' });
  }
});

// Actualizar configuración
router.put('/config', async (req, res) => {
  try {
    const restaurant = await Restaurant.findByIdAndUpdate(
      req.restaurantId,
      { ...req.body, updatedAt: new Date() },
      { new: true }
    );
    res.json(restaurant);
  } catch (error) {
    console.error('Error al actualizar config:', error);
    res.status(500).json({ error: 'Error en el servidor' });
  }
});

// Obtener estadísticas
router.get('/stats', async (req, res) => {
  try {
    const today = new Date();
    today.setHours(0, 0, 0, 0);
```

```
    const [totalConversations, todayConversations, totalOrders] = await Promise.all([
      Conversation.countDocuments({ restaurantId: req.restaurantId }),
      Conversation.countDocuments({ restaurantId: req.restaurantId, createdAt: { $gte: today } }),
      Conversation.countDocuments({ restaurantId: req.restaurantId, orderPlaced: true })
    ]);

    res.json({
      totalConversations,
      todayConversations,
      totalOrders,
      conversionRate: totalConversations > 0 ? ((totalOrders / totalConversations) * 100).toFixed(1) : 0
    });
  } catch (error) {
    console.error('Error al obtener stats:', error);
    res.status(500).json({ error: 'Error en el servidor' });
  }
});

// Obtener conversaciones recientes
router.get('/conversations', async (req, res) => {
  try {
    const conversations = await Conversation.find({ restaurantId: req.restaurantId })
      .sort({ updatedAt: -1 })
      .limit(50);
    res.json(conversations);
  } catch (error) {
    console.error('Error al obtener conversaciones:', error);
    res.status(500).json({ error: 'Error en el servidor' });
  }
});

module.exports = router;
```

backend/middleware/auth.js

javascript

```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
  try {
    const token = req.header('Authorization')?.replace('Bearer ', '');

    if (!token) {
      return res.status(401).json({ error: 'Acceso denegado' });
    }

    const decoded = jwt.verify(token, process.env.JWT_SECRET || 'secret-key');
    req.userId = decoded.userId;
    req.restaurantId = decoded.restaurantId;
    next();
  } catch (error) {
    res.status(401).json({ error: 'Token inválido' });
  }
};
```

4. Archivo `.env`

```
env

# Puerto del servidor
PORT=5000

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

# JWT Secret
JWT_SECRET=tu-secret-key-super-segura-aqui

# Anthropic API
ANTHROPIC_API_KEY=tu-api-key-de-anthropic

# Ambiente
NODE_ENV=development
```

5. `package.json` scripts

```
json
```

```
{
  "scripts": {
    "start": "node backend/server.js",
    "dev": "nodemon backend/server.js",
    "test": "node test-connection.js"
  }
}
```

🧪 Testing Local

Archivo `test-connection.js`

```javascript
```

```
const mongoose = require('mongoose');
const Anthropic = require('@anthropic-ai/sdk');
require('dotenv').config();

async function testConnections() {
  console.log('🧪 Probando conexiones...\n');

  // Test MongoDB
  try {
    await mongoose.connect(process.env.MONGODB_URI || 'mongodb://localhost:27017/restaurant-chatbot');
    console.log('✅ MongoDB conectado correctamente');
    await mongoose.disconnect();
  } catch (error) {
    console.error('❌ Error MongoDB:', error.message);
  }

  // Test Anthropic
  try {
    const anthropic = new Anthropic({
      apiKey: process.env.ANTHROPIC_API_KEY
    });
    const response = await anthropic.messages.create({
      model: 'claude-sonnet-4-20250514',
      max_tokens: 100,
      messages: [{ role: 'user', content: 'Di hola' }]
    });
    console.log('✅ Anthropic API funcionando');
    console.log('   Respuesta:', response.content[0].text);
  } catch (error) {
    console.error('❌ Error Anthropic:', error.message);
  }
}

testConnections();
```

🚀 Comandos para Ejecutar

bash

```
# Instalar dependencias
npm install

# Probar conexiones
npm test

# Iniciar servidor en desarrollo
npm run dev

# Producción
npm start
```

📝 Endpoints API

Autenticación

- `POST /api/auth/register` - Crear cuenta
- `POST /api/auth/login` - Iniciar sesión

Chatbot (público)

- `POST /api/chatbot/message` - Enviar mensaje
- `GET /api/chatbot/conversation/:sessionId` - Obtener historial

Admin (requiere autenticación)

- `GET /api/admin/config` - Obtener configuración
- `PUT /api/admin/config` - Actualizar configuración
- `GET /api/admin/stats` - Obtener estadísticas
- `GET /api/admin/conversations` - Listar conversaciones

🔒 Seguridad

1. **Variables de entorno**: Nunca subir `.env` a git
2. **JWT**: Cambiar `JWT_SECRET` en producción
3. **Rate limiting**: Implementar para prevenir abuso
4. **CORS**: Configurar orígenes permitidos
5. **Validación**: Validar todos los inputs

📊 Monitoreo y Logs

Agregar logging con Winston:

```bash
npm install winston
```

```javascript
const winston = require('winston');

const logger = winston.createLogger({
  level: 'info',
  format: winston.format.json(),
  transports: [
    new winston.transports.File({ filename: 'error.log', level: 'error' }),
    new winston.transports.File({ filename: 'combined.log' })
  ]
});

if (process.env.NODE_ENV !== 'production') {
  logger.add(new winston.transports.Console({
    format: winston.format.simple()
  }));
}
```

🌐 Deploy en Producción

Opciones recomendadas:

1. **Backend**: Railway, Render, DigitalOcean
2. **Frontend**: Vercel, Netlify
3. **Database**: MongoDB Atlas (gratuito)

Variables de entorno en producción:

```env
NODE_ENV=production
MONGODB_URI=mongodb+srv://usuario:password@cluster.mongodb.net/
JWT_SECRET=super-secret-production-key-256-bits
ANTHROPIC_API_KEY=sk-ant-...
ALLOWED_ORIGINS=https://tu-dominio.com
```

📚 Próximos Pasos

1. ✅ Implementar panel de billing con Stripe
2. ✅ Agregar webhooks para WhatsApp Business
3. ✅ Sistema de analytics avanzado
4. ✅ Multi-idioma
5. ✅ Plantillas de mensajes personalizables
6. ✅ Integración con sistemas POS

¿Necesitas ayuda? Contacta al equipo de soporte.