

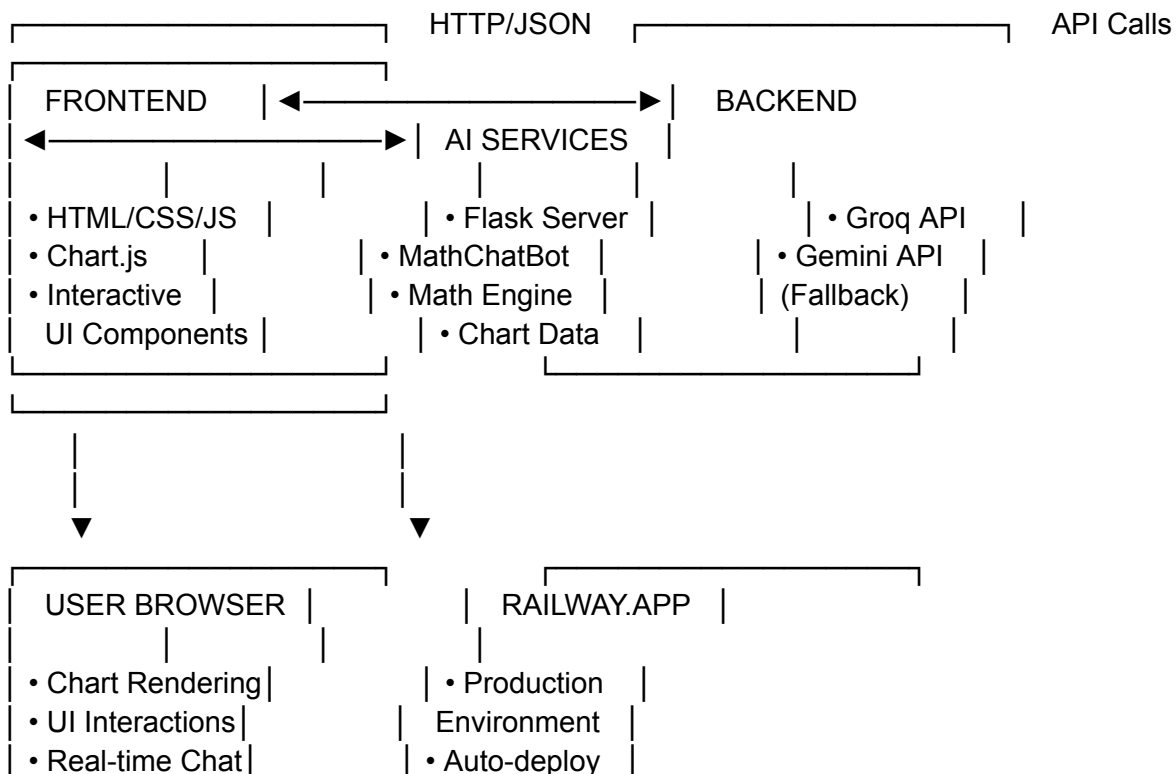
Documentación Completa - ChatBot Matemático con IA y Gráficas

Tabla de Contenido

1. [Arquitectura General](#)
 2. [Backend - Python Flask](#)
 3. [Frontend - HTML/CSS/JavaScript](#)
 4. [API y Comunicación](#)
 5. [Sistema de Gráficas](#)
 6. [Inteligencia Artificial](#)
 7. [Deployment y Producción](#)
 8. [Guía para Desarrolladores](#)
-

Arquitectura General

Diagrama de Arquitectura



Tecnologías Utilizadas

Backend

- **Python 3.11+** - Lenguaje principal
- **Flask 2.3.3** - Framework web
- **Flask-CORS** - Manejo de CORS para frontend
- **Requests** - Llamadas HTTP a APIs externas
- **Math Library** - Cálculos matemáticos avanzados
- **Python-dotenv** - Manejo de variables de entorno
- **Gunicorn** - Servidor WSGI para producción

Frontend

- **HTML5** - Estructura semántica
- **CSS3** - Estilos modernos con transiciones
- **Vanilla JavaScript** - Lógica del cliente (sin frameworks)
- **Chart.js 3.9.1** - Renderizado de gráficas
- **CSS Grid/Flexbox** - Layout responsivo

APIs Externas

- **Groq API** - IA conversacional (LLaMA-3.1)
- **Gemini API** - IA de respaldo (Google)

Infraestructura

- **Railway.app** - Hosting y deployment
- **GitHub** - Control de versiones
- **Netlify** - Hosting alternativo para frontend

Backend - Python Flask

Estructura del Proyecto

chatbot-matematico/

```
|— app.py          # Servidor Flask principal
|— chatbot.py      # Lógica del ChatBot con IA
|— requirements.txt # Dependencias Python
|— Procfile       # Configuración Railway
```

- └─ Dockerfile # Contenedor Docker
- └─ nixpacks.toml # Configuración Nixpacks
- └─ .env.example # Variables de entorno ejemplo
- └─ verify_deployment.py # Script de verificación

app.py - Servidor Flask Principal

Funciones Principales

`__init__` - Inicialización

```
app = Flask(__name__)
CORS(app) # Permite requests del frontend
bot = MathChatBot() # Instancia del chatbot
```

`index()` - Ruta Principal

```
@app.route('/')
def index():
    """Servir la página principal HTML"""
    # Lee index.html y lo sirve
    # Fallback si no encuentra el archivo
```

`css()` y `js()` - Recursos Estáticos

```
@app.route('/style.css')
def css():
    """Servir archivo CSS con headers de cache"""

@app.route('/script.js')
def js():
    """Servir archivo JavaScript con headers de cache"""
```

`/api/chat` - Endpoint Principal

```
@app.route('/api/chat', methods=['POST'])
def chat():
    """
    Endpoint principal del chatbot

    Input: JSON con 'message'
    Output: JSON con 'response', 'type', 'chart_data' (opcional)

    Flujo:
```

1. Validar input JSON
 2. Extraer mensaje del usuario
 3. Procesar con MathChatBot
 4. Retornar respuesta estructurada
- ```
"""
```

### **/health - Health Check**

```
@app.route('/health')
def health_check():
 """
 Verificación de estado del sistema

 Output: JSON con estado del chatbot, versión, features
 Usado por Railway para monitoreo
 """
```

### **Manejo de Errores**

```
@app.errorhandler(404)
def not_found(error):
 """Manejo de páginas no encontradas"""

@app.errorhandler(500)
def internal_error(error):
 """Manejo de errores internos"""
```

## **chatbot.py - Núcleo del ChatBot**

### **Clase MathChatBot**

#### **\_\_init\_\_() - Inicialización**

```
def __init__(self):
 """
 Inicializa el chatbot con:
 - Contexto de conversación
 - Variables matemáticas
 - Configuración de APIs
 - Funciones matemáticas
 - Rate limiting
 """

 self.context = [] # Historial de mensajes
 self.conversation_history = [] # Conversaciones completas
 self.last_result = None # Último resultado calculado
```

```
self.variables = {} # Variables definidas por usuario
self.api_key = os.getenv('GROQ_API_KEY') # Clave API
self.math_functions = {...} # Funciones matemáticas disponibles
```

### **is\_chart\_request()** - Detección de Gráficas

```
def is_chart_request(self, message):
 """
 Detecta si el usuario solicita una gráfica

 Busca:
 - Palabras clave: 'grafica', 'dibuja', 'muestra'
 - Funciones: sin, cos, tan, log, etc.
 - Patrones algebraicos: 3x+2, x^2, f(x)=

 Returns: bool
 """
```

### **parse\_chart\_request()** - Análisis de Gráficas

```
def parse_chart_request(self, message):
 """
 Analiza qué función(es) graficar

 Proceso:
 1. Normalizar lenguaje natural
 2. Separar múltiples funciones
 3. Identificar tipo (algebraica/predefinida)
 4. Extraer rango de valores

 Returns: dict con functions, range, type
 """
```

### **generate\_chart\_data()** - Generación de Datos

```
def generate_chart_data(self, chart_info):
 """
 Genera datos para Chart.js

 Proceso:
 1. Crear 201 puntos en el rango x
 2. Evaluar función para cada x
 3. Manejar valores indefinidos/extremos
 4. Crear datasets con colores
```

## 5. Configurar opciones de Chart.js

Returns: dict compatible con Chart.js

"""

### **is\_mathematical\_expression()** - Detección de Cálculos

def is\_mathematical\_expression(self, message):

"""

Detecta expresiones matemáticas calculables

Busca:

- Operadores: +, -, \*, /, ^, \*\*
- Funciones: sin(), sqrt(), log()
- Frases: "cuánto es", "calcula"

Returns: bool

"""

### **safe\_eval()** - Evaluación Segura

def safe\_eval(self, expression):

"""

Evalúa expresiones matemáticas de forma segura

Seguridad:

- Solo permite funciones matemáticas
- Bloquea \_\_builtins\_\_
- Reemplaza operadores problemáticos

Returns: resultado numérico

"""

### **get\_ai\_response\_sync()** - Comunicación con IA

def get\_ai\_response\_sync(self, message):

"""

Obtiene respuesta de IA (Groq/Gemini)

Features:

- Rate limiting (2 segundos)
- Fallback entre APIs
- Manejo de errores
- Timeouts configurables

Returns: string con respuesta IA

"""

### **get\_response()** - Método Principal

```
def get_response(self, message):
```

"""

Método principal que coordina toda la respuesta

Flujo de decisión:

1. ¿Es solicitud conceptual? → IA explicativa
2. ¿Es solicitud de gráfica? → Generar gráfica + IA
3. ¿Es expresión matemática? → Calcular + IA
4. ¿Otro? → IA conversacional
5. Fallback → Respuestas predefinidas

Returns: dict con response, type, chart\_data

"""

---

## Frontend - HTML/CSS/JavaScript

### Estructura del Frontend


frontend/

|              |                         |
|--------------|-------------------------|
| — index.html | # Estructura HTML       |
| — style.css  | # Estilos y animaciones |
| — script.js  | # Lógica JavaScript     |

### index.html - Estructura

#### Componentes Principales

##### Header

```
<div class="chat-header">
 <div class="logo-header">
 <div class="logo-icon">  <!-- Logo animado -->
 </div>
 <h2>CUCHAO CHAT</h2>
 Conectando...
</div>
```

### Panel de Bienvenida

```
<div class="welcome-panel" id="welcomePanel">
 <!-- Se oculta automáticamente cuando usuario escribe -->
 <div class="welcome-options">
 <div class="option-card" onclick="selectCategory('ejemplos')">
 <div class="option-card" onclick="selectCategory('graficas')">
 <div class="option-card" onclick="selectCategory('escribir')">
 </div>
</div>
```

### Paneles Dinámicos

```
<!-- Categorías de ejemplos -->
<div class="categories-panel" id="categoriesPanel">

<!-- Ejemplos específicos -->
<div class="examples-panel" id="examplesPanel">



<!-- Ejemplos de gráficas -->
<div class="charts-panel" id="chartsPanel">

<!-- Teclado matemático -->
<div class="math-keyboard" id="mathKeyboard">
```

### Área de Chat

```
<div class="chat-messages" id="chatMessages">
 <!-- Mensajes se agregan dinámicamente -->
</div>
```

### Input y Herramientas

```
<div class="chat-input">
 <div class="input-tools">
 <button class="tool-btn" onclick="toggleKeyboard()">  </button>
 <button class="tool-btn" onclick="toggleExamples()"> ⚡ </button>
 <button class="tool-btn" onclick="toggleCharts()">  </button>
 <button class="tool-btn" onclick="showHelp()"> ? </button>
 </div>
 <div class="input-container">
 <input type="text" id="messageInput">
 <button id="sendButton">Enviar</button>
 </div>
```



</div>

## style.css - Estilos Modernos

### Sistema de Colores

```
:root {
 --primary-red: #ff0000;
 --secondary-red: #cc0000;
 --background: linear-gradient(135deg, #000000 0%, #1a1a1a 50%, #000000 100%);
 --glass-bg: rgba(255, 255, 255, 0.95);
 --shadow: 0 32px 64px rgba(255, 0, 0, 0.1);
}
```

### Animaciones Clave

```
@keyframes slideIn {
 /* Entrada suave del contenedor principal */
}
```

```
@keyframes shimmer {
 /* Efecto brillante en el header */
}
```

```
@keyframes logoGlow {
 /* Animación del logo */
}
```

```
@keyframes bounce {
 /* Indicador de escritura */
}
```

### Layout Responsivo

```
.chat-container {
 /* Container principal con glassmorphism */
 backdrop-filter: blur(10px);
 border-radius: 24px;
}
```

```
@media (max-width: 600px) {
 /* Adaptación móvil */
}
```

## script.js - Lógica JavaScript

### Clase MathChatBot

#### Constructor

```
constructor() {
 // Referencias DOM
 this.messageInput = document.getElementById('messageInput');
 this.chatMessages = document.getElementById('chatMessages');

 // Estado de la interfaz
 this.welcomePanelHidden = false;
 this.chatCount = 0;

 // URLs API
 this.apiUrl = this.getApiUrl() + '/api/chat';

 // Datos
 this.examples = {...}; // Ejemplos por categoría
 this.suggestions_data = [...]; // Sugerencias automáticas
}
```

#### getApiUrl() - Detección de Entorno

```
getApiUrl() {
 // Desarrollo: localhost:5000
 if (window.location.hostname === 'localhost') {
 return 'http://localhost:5000';
 } else {
 // Producción: mismo origen
 return window.location.origin;
 }
}
```

#### setupEventListeners() - Event Handlers

```
setupEventListeners() {
 // Enter para enviar
 this.messageInput.addEventListener('keypress', (e) => {
 if (e.key === 'Enter') this.sendMessage();
 });

 // Ocultar welcome panel al escribir
 this.messageInput.addEventListener('input', (e) => {
```

```

 if (e.target.value.trim().length > 0) {
 this.hideWelcomePanelSmooth();
 }
 this.showSuggestions(e.target.value);
 });
}

```

### hideWelcomePanelSmooth() - Transición Suave

```

hideWelcomePanelSmooth() {
 this.welcomePanelHidden = true;
 this.welcomePanel.style.transition = 'all 0.5s cubic-bezier(0.4, 0, 0.2, 1)';
 this.welcomePanel.style.opacity = '0';
 this.welcomePanel.style.transform = 'translateY(-20px)';

 setTimeout(() => {
 this.welcomePanel.style.display = 'none';
 }, 500);
}

```

### createChart() - Renderizado de Gráficas

```

createChart(chartData) {
 this.chartCount++;

 // Crear contenedor
 const chartContainer = document.createElement('div');
 chartContainer.className = 'chart-container';
 chartContainer.innerHTML = `
 <canvas id="chart-${this.chartCount}"></canvas>
 `;

 // Renderizar con Chart.js
 const ctx = document.getElementById(`chart-${this.chartCount}`).getContext('2d');
 new Chart(ctx, chartData);
}

```

### sendMessage() - Comunicación con Backend

```

async sendMessage() {
 const message = this.messageInput.value.trim();

 // Validaciones
 // Mostrar mensaje usuario
}

```

```
// Indicador de carga

try {
 const response = await fetch(this.apiUrl, {
 method: 'POST',
 headers: { 'Content-Type': 'application/json' },
 body: JSON.stringify({ message }),
 signal: AbortSignal.timeout(25000)
 });

 const data = await response.json();

 // Mostrar respuesta bot
 this.addMessage(data.response, 'bot');

 // Renderizar gráfica si existe
 if (data.chart_data) {
 this.createChart(data.chart_data);
 }

} catch (error) {
 // Manejo de errores
}
}
```

---

## API y Comunicación

### Protocolo de Comunicación

#### Request Format

POST /api/chat

Content-Type: application/json

```
{
 "message": "grafica sin(x)"
}
```

#### Response Format - Texto Simple

```
{
 "response": "El resultado es 42",
}
```

```
"status": "success",
"type": "calculation"
}
```

### Response Format - Con Gráfica

```
{
 "response": "Aquí tienes la gráfica de sin(x)...",
 "status": "success",
 "type": "chart",
 "chart_data": {
 "type": "line",
 "data": {
 "datasets": [{
 "label": "sin(x)",
 "data": [{"x": -10, "y": 0.544}, ...],
 "borderColor": "#ff0000"
 }]
 },
 "options": {...}
 }
}
```

### Tipos de Respuesta

- "calculation" - Resultado matemático
- "chart" - Gráfica incluida
- "conversation" - Respuesta IA
- "concept" - Explicación educativa
- "fallback" - Respuesta predefinida
- "error" - Error del sistema

### Health Check Endpoint

#### Request

GET /health

#### Response

```
{
 "status": "healthy",
 "chatbot": "ai_math_specialist_with_charts",
 "version": "4.0",
}
```

```
"environment": "production",
"features": {
 "ai_enabled": true,
 "charts_enabled": true,
 "math_calculations": true,
 "conversation": true
},
"chatbot_status": "operational"
}
```

## Error Handling

### Client-Side Errors

```
// Timeout
if (error.name === 'AbortError') {
 this.addMessage('La respuesta tardó demasiado.', 'bot');
}

// Network Error
catch (error) {
 this.addMessage('No pude conectar con el servidor.', 'bot');
 this.updateStatus('Sin conexión', 'disconnected');
}
```

### Server-Side Errors

```
Input validation
if not message:
 return jsonify({
 'error': 'El campo "message" es requerido',
 'status': 'bad_request'
 }), 400

Internal errors
except Exception as e:
 return jsonify({
 'error': 'Error interno del servidor',
 'status': 'internal_error'
 }), 500
```

---



# Sistema de Gráficas

## Flujo de Generación de Gráficas

### 1. Detección (Frontend)

```
// Usuario escribe: "grafica sin(x)"
// Script.js detecta palabras clave y envía al backend
```

### 2. Parsing (Backend)

```
def parse_chart_request(self, message):
 # "grafica sin(x)" →
 {
 'functions': [{ 'type': 'predefined', 'name': 'sin' }],
 'range': [-10, 10],
 'type': 'single'
 }
```

### 3. Generación de Datos

```
def generate_chart_data(self, chart_info):
 # Crear 201 puntos de x entre -10 y 10
 x_values = [x_min + i * (x_max - x_min) / 200 for i in range(201)]

 # Evaluar función para cada x
 for x in x_values:
 y = math.sin(x) # Ejemplo: sin(x)
 y_values.append(y)

 # Formato Chart.js
 return {
 'type': 'line',
 'data': { 'datasets': [...]},
 'options': {...}
 }
```

### 4. Renderizado (Frontend)

```
createChart(chartData) {
 // Crear canvas único
 const canvas = document.createElement('canvas');
 canvas.id = `chart-${this.chartCount}`;

 // Renderizar con Chart.js
```

```
new Chart(canvas.getContext('2d'), chartData);
}
```

## Tipos de Funciones Soportadas

### Trigonométricas

'sin': math.sin, # sin(x)  
'cos': math.cos, # cos(x)  
'tan': math.tan, # tan(x) - limitado a  $|y| < 10$

### Algebraicas

# Ejemplos de input → output

"3x+2" → "3\*x+2"

"x^2" → "x\*\*2"

"2(x+1)" → "2\*(x+1)"

"f(x)=x^2" → "x\*\*2"

### Logarítmicas/Exponenciales

'log': math.log, # log(x) - solo  $x > 0$   
'exp': math.exp, # exp(x) - limitado a  $y < 1000$   
'sqrt': math.sqrt, # sqrt(x) - solo  $x \geq 0$

## Configuración de Chart.js

### Opciones Base

```
options: {
 responsive: true,
 interaction: {
 intersect: false,
 mode: 'index'
 },
 scales: {
 x: {
 type: 'linear',
 title: { display: true, text: 'x' }
 },
 y: {
 title: { display: true, text: 'y' }
 }
 }
}
```



```
}
```

## Colores y Estilos

```
colors = [
 '#ff0000', // Rojo (función principal)
 '#0066cc', // Azul (segunda función)
 '#009900', // Verde (tercera función)
 '#ff9900', // Naranja
 '#990099', // Morado
 '#cc6600' // Marrón
]
```

---

## Inteligencia Artificial

### Proveedores de IA

#### Groq API (Primario)

```
Configuración
model = "llama-3.1-8b-instant"
api_url = "https://api.groq.com/openai/v1/chat/completions"
```

#### # Ventajas

- Más rápido (respuestas en 1-3 segundos)
- Menos restrictivo con contenido
- Rate limit más permisivo
- Mejor para matemáticas

#### Gemini API (Fallback)

```
Configuración
model = "gemini-1.5-flash"
api_url = "https://generativelanguage.googleapis.com/v1beta/models/..."
```

#### # Limitaciones

- Más lento (3-8 segundos)
- Más restrictivo
- Rate limits más estrictos

## Sistema de Prompting

## System Prompt

```
system_prompt = ""
```

Eres un profesor de matemáticas experto y amigable.

Explica conceptos claramente y proporciona ejemplos útiles.

Sé conciso pero educativo.

```
"""
```

## Contexto Dinámico

# Para cálculos

f"El usuario calculó '{expression}' = {result}. Explica brevemente esta operación."

# Para gráficas

f"Explica brevemente la función matemática {function\_name} y sus características."

# Para conceptos

f"Da una explicación clara, breve y con ejemplos sobre: {message}"

## Rate Limiting

### Implementación

```
def get_ai_response_sync(self, message):
```

```
 # Verificar tiempo desde última llamada
```

```
 current_time = time.time()
```

```
 time_since_last = current_time - self.last_api_call
```

```
 if time_since_last < self.min_interval:
```

```
 sleep_time = self.min_interval - time_since_last
```

```
 time.sleep(sleep_time)
```

```
 # Realizar llamada
```

```
 self.last_api_call = time.time()
```

### Configuración

```
self.min_interval = 2.0 # 2 segundos entre llamadas (Groq)
```

```
self.min_interval = 5.0 # 5 segundos para Gemini
```

## Fallback System

### Jerarquía de Respuestas

1. **IA Disponible** → Respuesta personalizada
2. **IA No Disponible** → Respuestas predefinidas por categoría
3. **Error Total** → Mensaje genérico de error

### Respuestas Predefinidas

```
def get_fallback_response(self, message):
 # Saludos
 if 'hola' in message.lower():
 return random.choice([
 "¡Hola! Soy tu asistente matemático...",
 "¡Saludos! Puedo resolver problemas...",
 "¡Bienvenido! Estoy listo para..."
])

 # Ayuda
 if 'ayuda' in message.lower():
 return "Soy tu asistente matemático..."
```

---

## Deployment y Producción

### Railway.app Configuration

#### Archivos de Configuración

##### Procfile

```
web: gunicorn --bind 0.0.0.0:$PORT --workers 1 --timeout 30 app:app
```

##### Dockerfile

```
FROM python:3.11-slim
```

```
WORKDIR /app
```

```
Instalar dependencias del sistema
```

```
RUN apt-get update && apt-get install -y gcc
```

```
Instalar dependencias Python
```

```
COPY requirements.txt .
```

```
RUN pip install --no-cache-dir -r requirements.txt
```

```
Copiar código
```

```
COPY . .
```

```
Comando de ejecución
CMD ["sh", "-c", "gunicorn --bind 0.0.0.0:$PORT app:app"]
```

#### **nixpacks.toml**

```
[phases.setup]
nixpkgs = ['python311', 'pip']

[phases.install]
cmds = [
 'python -m venv /opt/venv',
 '. /opt/venv/bin/activate && pip install -r requirements.txt'
]

[start]
cmd = '. /opt/venv/bin/activate && gunicorn --bind 0.0.0.0:$PORT app:app'
```

### **Variables de Entorno**

#### **Producción (Railway)**

```
GROQ_API_KEY=gsk_XXXXXXXXXXXXXXXXXXXXX
FLASK_ENV=production
FLASK_DEBUG=False
PORT=5000
HOST=0.0.0.0
```

#### **Desarrollo Local**

```
GROQ_API_KEY=tu_clave_de_desarrollo
FLASK_ENV=development
FLASK_DEBUG=True
PORT=5000
HOST=127.0.0.1
```

### **Monitoreo y Logs**

#### **Health Check**

```
@app.route('/health')
def health_check():
 """
```

Endpoint de monitoreo que verifica:

- Estado del chatbot
- Conexión a APIs externas

- Funcionalidades disponibles  
""

## Logging en Producción

```
if os.environ.get('FLASK_ENV') == 'development':
 print(f"🧠 Consulta: {user_message}")
 print(f"✅ Respuesta {response_type}")
```

## Performance Optimization

### Backend

- **Gunicorn** con 1 worker (Railway free tier)
- **Timeout** de 30 segundos para gráficas complejas
- **Rate limiting** para APIs externas
- **Cache headers** para archivos estáticos

### Frontend

- **Chart.js** cargado desde CDN
- **Lazy loading** de gráficas
- **CSS optimizado** con animaciones hardware-accelerated
- **Responsive design** para móviles



## Guía para Desarrolladores

### Setup Local

#### 1. Clonar Repositorio

```
git clone https://github.com/tu-usuario/chatbot-matematico.git
cd chatbot-matematico
```

#### 2. Crear Entorno Virtual

```
python -m venv venv
source venv/bin/activate # Linux/Mac
venv\Scripts\activate # Windows
```

#### 3. Instalar Dependencias

```
pip install -r requirements.txt
```

#### 4. Configurar Variables de Entorno

```
cp .env.example .env
```

```
Editar .env con tu GROQ_API_KEY
```

#### 5. Ejecutar Localmente

```
python app.py
```

```
Abrir http://localhost:5000
```

### Estructura de Desarrollo

#### Flujo de Trabajo

1. Desarrollar feature → rama feature/nueva-funcionalidad
2. Probar localmente → python app.py
3. Verificar deployment → python verify\_deployment.py
4. Push a main → git push origin main
5. Auto-deploy → Railway detecta y despliega

#### Testing

```
Verificar sistema completo
```

```
python verify_deployment.py
```

```
Test manual de endpoints
```

```
curl http://localhost:5000/health
```

```
curl -X POST http://localhost:5000/api/chat \
```

```
-H "Content-Type: application/json" \
```

```
-d '{"message": "grafica sin(x)}'
```

### Extensiones Futuras

#### Nuevas Funcionalidades

1. **Gráficas 3D** - Three.js integration
2. **Ecuaciones LaTeX** - MathJax rendering
3. **Exportar Gráficas** - Canvas to PNG/SVG
4. **Modo Oscuro** - CSS custom properties
5. **Múltiples Idiomas** - i18n support

#### Mejoras de Backend

```
Ejemplo: Cache de respuestas IA
from functools import lru_cache
```

```
@lru_cache(maxsize=100)
def get_cached_ai_response(self, message_hash):
 # Cache respuestas comunes para reducir API calls
```

## Mejoras de Frontend

```
// Ejemplo: WebSocket para chat real-time
const socket = new WebSocket('ws://localhost:5000/ws');
socket.onmessage = (event) => {
 const data = JSON.parse(event.data);
 this.addMessage(data.response, 'bot');
};
```

## Debugging

### Backend Debug

```
En app.py - activar logs detallados
import logging
logging.basicConfig(level=logging.DEBUG)
```

```
En chatbot.py - debug específico
print(f"📄 Parseando solicitud: {message}")
print(f"📋 Info de gráfica: {chart_info}")
```

### Frontend Debug

```
// En script.js - console logging
console.log('📤 Enviando mensaje:', message);
console.log('📥 Respuesta recibida:', data);
console.log('📊 Datos de gráfica:', data.chart_data);
```

## Common Issues

### Gráficas No Aparecen

1. Verificar que Chart.js se carga: `console.log(Chart)`
2. Check datos de respuesta: `console.log(data.chart_data)`
3. Verificar errores Canvas: Check Developer Tools

### API No Responde

1. Verificar GROQ\_API\_KEY en Railway
2. Check rate limiting: responder cada 2+ segundos
3. Verificar logs en Railway dashboard

### UI No Responsive

1. Verificar viewport meta tag
2. Check CSS media queries
3. Test en diferentes dispositivos

## Contribuir al Proyecto

### Pull Request Process

1. Fork del repositorio
2. Crear feature branch
3. Implementar cambios con tests
4. Documentar nuevas funcionalidades
5. Crear PR con descripción detallada

### Code Style

- **Python:** PEP 8
- **JavaScript:** ES6+ features
- **CSS:** BEM methodology
- **Commits:** Conventional Commits



## Conclusión

Este ChatBot Matemático es un sistema completo que combina:

- **Frontend moderno** con UI intuitiva
- **Backend robusto** con Flask y Python
- **IA conversacional** con Groq/Gemini
- **Sistema de gráficas** dinámico con Chart.js
- **Deployment automático** en Railway

La arquitectura modular permite extensiones fáciles y el código está bien documentado para facilitar el mantenimiento y contribuciones futuras.

---



*Documentación generada para ChatBot Matemático v4.0*  
*Última actualización: 2025*