# Sistema de Sandbox - Code Execution Environment

## Descripción

El sistema de sandbox permite la ejecución segura de código en múltiples lenguajes de programación dentro de un entorno aislado y controlado.

## Características Principales

### 🔒 Seguridad

* **Entorno aislado**: Ejecución en contenedores Docker (cuando está disponible) o procesos limitados
* **Filtros de seguridad**: Bloqueo automático de código potencialmente peligroso
* **Límites de recursos**: Control de memoria, tiempo de ejecución y acceso a red
* **Sin persistencia**: No se almacenan archivos ni estado entre ejecuciones

### 🚀 Lenguajes Soportados

* **Python 3.11**: Ideal para algoritmos, data science y scripting
* **JavaScript (Node.js)**: Para lógica frontend y backend
* **Bash**: Para scripts de sistema y automatización

### ⚡ Rendimiento

* **Timeouts configurables**: 15-30 segundos según el lenguaje
* **Límites de memoria**: 64-128MB por ejecución
* **Ejecución paralela**: Múltiples sandboxes simultáneos

## Endpoints de la API

### POST /api/sandbox/execute

Ejecutar código en el sandbox.

**Request Body:**

{  
 "code": "print('Hello, World!')",  
 "language": "python",  
 "input": "datos opcionales"  
}

**Response:**

{  
 "success": true,  
 "output": "Hello, World!\n",  
 "exit\_code": 0,  
 "execution\_time": 0.123,  
 "language": "python"  
}

### GET /api/sandbox/languages

Obtener lista de lenguajes soportados y configuraciones.

### GET /api/sandbox/examples

Obtener ejemplos de código para cada lenguaje.

### GET /api/sandbox/health

Verificar estado del sistema de sandbox.

## Configuración de Seguridad

### Patrones Bloqueados

El sistema bloquea automáticamente código que contenga:

* **Imports peligrosos**: import os, import subprocess, import sys
* **Ejecución dinámica**: eval(), exec(), \_\_import\_\_
* **Acceso a archivos**: open(), file()
* **Input del usuario**: input(), raw\_input()
* **Comandos de sistema**: rm -rf, sudo, wget, curl

### Límites por Lenguaje

| Lenguaje | Timeout | Memoria | Extensión |
| --- | --- | --- | --- |
| Python | 30s | 128MB | .py |
| JavaScript | 30s | 128MB | .js |
| Bash | 15s | 64MB | .sh |

## Arquitectura

### Con Docker (Recomendado)

User Request → Flask API → Docker Container → Isolated Execution → Response

**Ventajas:** - Máximo aislamiento - Control completo de recursos - Sin riesgo para el host

### Sin Docker (Fallback)

User Request → Flask API → Subprocess → Limited Execution → Response

**Ventajas:** - No requiere Docker - Más compatible - Menor overhead

## Ejemplos de Uso

### Python - Algoritmos

def fibonacci(n):  
 if n <= 1:  
 return n  
 return fibonacci(n-1) + fibonacci(n-2)  
  
for i in range(10):  
 print(f"F({i}) = {fibonacci(i)}")

### JavaScript - Procesamiento de Datos

const data = [1, 2, 3, 4, 5];  
const result = data  
 .map(x => x \* 2)  
 .filter(x => x > 5)  
 .reduce((a, b) => a + b, 0);  
  
console.log(`Resultado: ${result}`);

### Bash - Automatización

#!/bin/bash  
echo "Análisis del sistema:"  
echo "Fecha: $(date)"  
echo "Usuario: $USER"  
  
for i in {1..5}; do  
 echo "Iteración $i"  
done

## Instalación y Configuración

### Dependencias

pip install docker psutil

### Variables de Entorno

# Opcional: Configurar límites personalizados  
export SANDBOX\_TIMEOUT=30  
export SANDBOX\_MEMORY\_LIMIT=128m  
export SANDBOX\_ENABLE\_DOCKER=true

### Docker Setup (Opcional)

# Verificar Docker  
docker --version  
  
# Pull imágenes necesarias  
docker pull python:3.11-alpine  
docker pull node:18-alpine  
docker pull alpine:latest

## Monitoreo y Logs

### Métricas Disponibles

* Tiempo de ejecución por request
* Uso de memoria por ejecución
* Tasa de éxito/error
* Lenguajes más utilizados

### Logs de Seguridad

* Código bloqueado por filtros
* Timeouts y límites alcanzados
* Errores de ejecución

## Casos de Uso

### 🎓 Educación

* Plataformas de aprendizaje online
* Evaluación automática de código
* Tutoriales interactivos

### 🧪 Prototipado

* Testing rápido de algoritmos
* Validación de lógica
* Experimentación de código

### 🤖 IA y Chatbots

* Ejecución de código generado por IA
* Validación automática de respuestas
* Demos en tiempo real

## Limitaciones Conocidas

### Funcionalidad Restringida

* ❌ Sin acceso a red/internet
* ❌ Sin persistencia de archivos
* ❌ Sin instalación de paquetes
* ❌ Sin acceso a GPU
* ❌ Sin operaciones de E/O de archivos

### Consideraciones de Rendimiento

* Contenedores Docker tienen overhead inicial
* Subprocess puede ser menos seguro
* Límites de memoria pueden ser restrictivos para algoritmos complejos

## Roadmap Futuro

### Próximas Características

* ☐ Soporte para más lenguajes (Go, Rust, Java)
* ☐ Paquetes preinstalados por lenguaje
* ☐ Métricas de rendimiento en tiempo real
* ☐ API de streaming para output en tiempo real
* ☐ Soporte para múltiples archivos
* ☐ Integration con CI/CD pipelines

### Mejoras de Seguridad

* ☐ Sandboxing a nivel de kernel
* ☐ Análisis estático de código antes de ejecución
* ☐ Rate limiting por usuario
* ☐ Audit logs detallados