

# Guía Universal: Mini Apps con IA Local y Modelos de Pesos Libres

## Índice

- [Introducción](#)
  - [Arquitectura Universal](#)
  - [Modelos Disponibles](#)
  - [Estructura de Proyecto](#)
  - [Implementación Paso a Paso](#)
  - [Integración de Modelos](#)
  - [Optimización para Dispositivos](#)
  - [Casos de Uso](#)
  - [Troubleshooting](#)
- 

## Introducción

Esta guía explica cómo crear mini aplicaciones completamente autónomas que utilizan modelos de IA de pesos libres (Llama, Mistral, Phi, etc.) sin depender de APIs externas o servicios de pago. El objetivo es crear aplicaciones que funcionen offline en cualquier dispositivo con recursos limitados.

### Ventajas de esta Arquitectura

- **Privacidad total:** Los datos nunca salen del dispositivo
- **Costo cero:** Sin APIs de pago ni suscripciones
- **Offline completo:** Funciona sin conexión a internet
- **Escalable:** Se adapta desde móviles hasta servidores
- **Personalizable:** Modelos especializables para dominios específicos

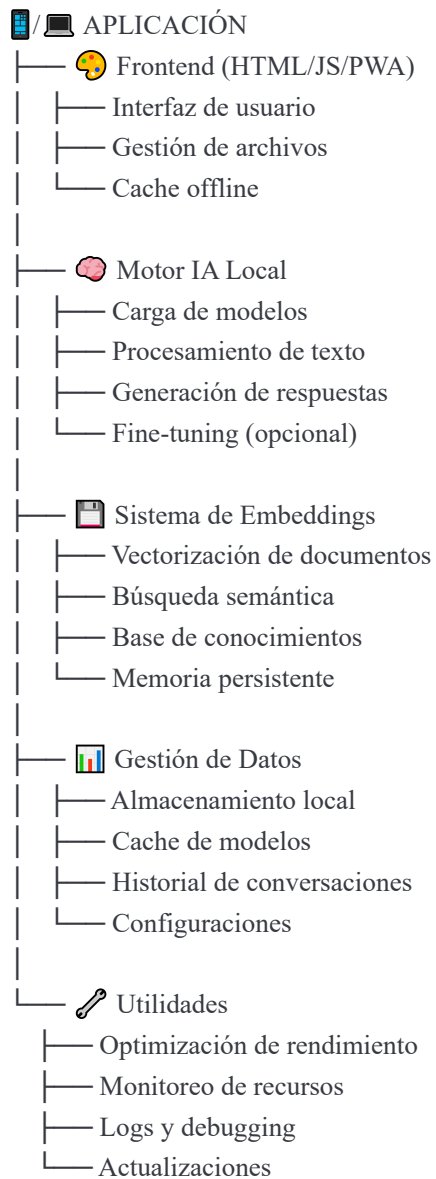
### Requisitos Mínimos

- Python 3.8+
  - 4GB RAM mínimo (8GB recomendado)
  - 2GB espacio disco para modelo básico
  - CPU moderna (recomendado: GPU opcional)
- 

## Arquitectura Universal

### Diagrama de Componentes





---

## Modelos Disponibles

### 1. Modelos Ligeros (< 1GB)



python

```
MODELOS_LIGEROS = {  
    "microsoft/DialoGPT-medium": {  
        "tamaño": "0.3GB",  
        "uso": "Conversación general",  
        "ram_min": "2GB",  
        "velocidad": "Alta"  
    },  
  
    "microsoft/phi-1_5": {  
        "tamaño": "0.8GB",  
        "uso": "Razonamiento básico",  
        "ram_min": "3GB",  
        "velocidad": "Alta"  
    },  
  
    "TinyLlama/TinyLlama-1.1B-Chat-v1.0": {  
        "tamaño": "0.6GB",  
        "uso": "Chat multiusos",  
        "ram_min": "2GB",  
        "velocidad": "Muy alta"  
    }  
}
```

## 2. Modelos Medianos (1-4GB)



python

```

MODELOS_MEDIANOS = {
    "microsoft/phi-2": {
        "tamaño": "2.7GB",
        "uso": "Razonamiento avanzado",
        "ram_min": "6GB",
        "velocidad": "Media-Alta"
    },

    "stabilityai/stablelm-3b-4e1t": {
        "tamaño": "3.2GB",
        "uso": "Generación de texto",
        "ram_min": "6GB",
        "velocidad": "Media"
    },

    "mistralai/Mistral-7B-Instruct-v0.2": {
        "tamaño": "3.8GB",
        "uso": "Instrucciones complejas",
        "ram_min": "8GB",
        "velocidad": "Media"
    }
}

```

### 3. Modelos Grandes (4-8GB)



python

```
MODELOS_GRANDES = {  
    "meta-llama/Llama-2-7b-chat-hf": {  
        "tamaño": "6.2GB",  
        "uso": "Chat avanzado multidominio",  
        "ram_min": "12GB",  
        "velocidad": "Media-Baja"  
    },  
  
    "NousResearch/Nous-Hermes-2-Mistral-7B-DPO": {  
        "tamaño": "6.8GB",  
        "uso": "Asistente especializado",  
        "ram_min": "12GB",  
        "velocidad": "Media-Baja"  
    }  
}
```

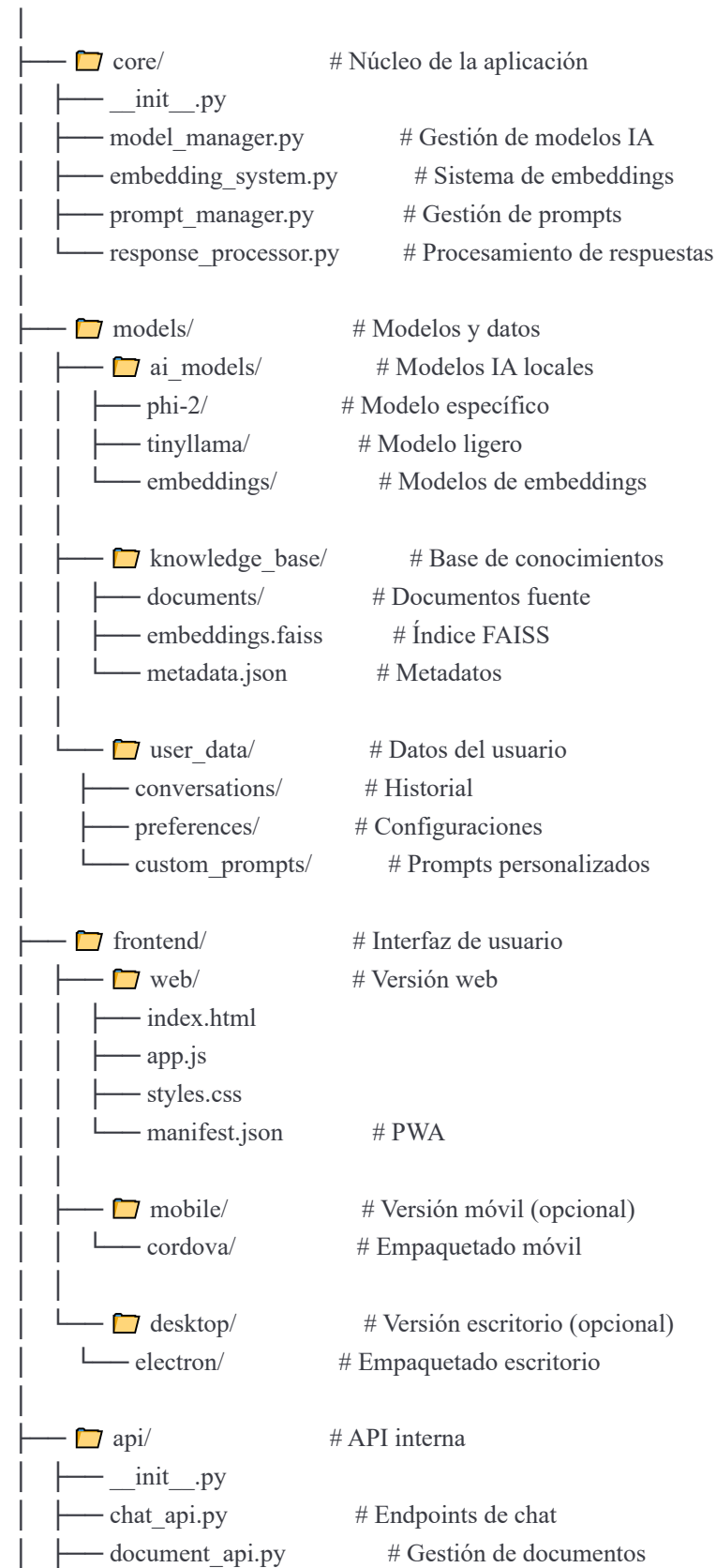
---




## Estructura de Proyecto

### Estructura Base Universal



mi-app-ia/



		└─ model_api.py	# Control de modelos
		└─  utils/	# Utilidades
			└─ __init__.py
			└─ device_optimizer.py # Optimización por dispositivo
			└─ memory_manager.py # Gestión de memoria
			└─ file_handler.py # Manejo de archivos
			└─ logger.py # Sistema de logs
		└─  config/	# Configuraciones
			└─ model_configs.json # Configuración de modelos
			└─ app_settings.json # Configuración de app
			└─ prompts.json # Templates de prompts
			└─ device_profiles.json # Perfiles por dispositivo
		└─  scripts/	# Scripts de utilidad
			└─ setup.py # Instalación automatizada
			└─ download_models.py # Descarga de modelos
			└─ build_knowledge_base.py # Construcción de KB
			└─ optimize_for_device.py # Optimización por dispositivo
		└─ requirements.txt	# Dependencias Python
		└─ app.py	# Aplicación principal
		└─ README.md	# Este archivo
		└─ setup_guide.md	# Guía específica del dominio

# Implementación Paso a Paso

## Paso 1: Configuración Inicial

### 1.1 Instalar Dependencias Base



bash

```
# requirements.txt
torch>=2.0.0
transformers>=4.35.0
sentence-transformers>=2.2.0
faiss-cpu>=1.7.4 # o faiss-gpu si tienes GPU
flask>=2.3.0
flask-cors>=4.0.0
numpy>=1.24.0
pandas>=2.0.0
tqdm>=4.65.0
psutil>=5.9.0 # Para monitoreo de recursos
accelerate>=0.24.0 # Para optimización
bitsandbytes>=0.41.0 # Para cuantización (opcional)
```

## 1.2 Script de Instalación Automática



python



```

# scripts/setup.py
#!/usr/bin/env python3
import os
import sys
import subprocess
from pathlib import Path

class UniversalSetup:
    def __init__(self):
        self.project_root = Path(__file__).parent.parent

    def detect_device(self):
        """Detecta capacidades del dispositivo"""
        import psutil

        device_info = {
            "ram_gb": round(psutil.virtual_memory().total / (1024**3)),
            "cpu_cores": psutil.cpu_count(),
            "has_gpu": self._check_gpu(),
            "storage_gb": round(psutil.disk_usage('/').total / (1024**3)),
            "platform": sys.platform
        }

        return device_info

    def _check_gpu(self):
        """Verifica disponibilidad de GPU"""
        try:
            import torch
            return torch.cuda.is_available()
        except:
            return False

    def recommend_model(self, device_info):
        """Recomienda modelo según capacidades"""
        ram_gb = device_info["ram_gb"]

        if ram_gb >= 16:
            return "mistralai/Mistral-7B-Instruct-v0.2"
        elif ram_gb >= 8:
            return "microsoft/phi-2"
        elif ram_gb >= 4:

```

```

        return "TinyLlama/TinyLlama-1.1B-Chat-v1.0"
    else:
        return "microsoft/DialoGPT-medium"

def install_requirements(self):
    """Instala dependencias optimizadas"""
    device_info = self.detect_device()

    # Instalar torch apropiado
    if device_info["has_gpu"]:
        subprocess.run([sys.executable, "-m", "pip", "install",
                        "torch", "torchvision", "--index-url",
                        "https://download.pytorch.org/whl/cu118"])
    else:
        subprocess.run([sys.executable, "-m", "pip", "install",
                        "torch", "torchvision", "--index-url",
                        "https://download.pytorch.org/whl/cpu"])

    # Instalar FAISS apropiado
    if device_info["has_gpu"]:
        subprocess.run([sys.executable, "-m", "pip", "install", "faiss-gpu"])
    else:
        subprocess.run([sys.executable, "-m", "pip", "install", "faiss-cpu"])

    # Instalar resto de dependencias
    subprocess.run([sys.executable, "-m", "pip", "install", "-r", "requirements.txt"])

def create_structure(self):
    """Crea estructura de directorios"""
    directories = [
        "core", "models/ai_models", "models/knowledge_base", "models/user_data",
        "frontend/web", "api", "utils", "config", "scripts", "logs"
    ]

    for directory in directories:
        (self.project_root / directory).mkdir(parents=True, exist_ok=True)

def run_setup(self):
    """Ejecuta configuración completa"""
    print("🤖 Configurando aplicación IA universal...")

    device_info = self.detect_device()
    print(f"Dispositivo detectado: {device_info['ram_gb']} GB RAM, {device_info['cpu_cores']} cores")

```

```
recommended_model = self.recommend_model(device_info)
print(f"Modelo recomendado: {recommended_model}")

self.create_structure()
self.install_requirements()

print("✅ Configuración completada")

if __name__ == "__main__":
    setup = UniversalSetup()
    setup.run_setup()
```

## Paso 2: Gestor de Modelos Universal



python

```

# core/model_manager.py
import torch
from transformers import (
    AutoTokenizer, AutoModelForCausalLM,
    TextIteratorStreamer, BitsAndBytesConfig
)
from threading import Thread
import json
import psutil
from pathlib import Path

class UniversalModelManager:
    def __init__(self, config_path="config/model_configs.json"):
        self.config = self.load_config(config_path)
        self.current_model = None
        self.current_tokenizer = None
        self.device_info = self.get_device_info()
        self.optimization_config = self.get_optimization_config()

    def load_config(self, config_path):
        """Carga configuración de modelos"""
        try:
            with open(config_path) as f:
                return json.load(f)
        except FileNotFoundError:
            return self.get_default_config()

    def get_default_config(self):
        """Configuración por defecto"""
        return {
            "models": {
                "tiny": {
                    "name": "TinyLlama/TinyLlama-1.1B-Chat-v1.0",
                    "max_length": 2048,
                    "temperature": 0.7,
                    "min_ram_gb": 2
                },
                "small": {
                    "name": "microsoft/phi-2",
                    "max_length": 2048,
                    "temperature": 0.7,
                    "min_ram_gb": 6
                }
            }
        }

```

```

    },
    "medium": {
        "name": "mistralai/Mistral-7B-Instruct-v0.2",
        "max_length": 4096,
        "temperature": 0.7,
        "min_ram_gb": 12
    }
}
}

```

```

def get_device_info(self):
    """Obtiene información del dispositivo"""
    return {
        "ram_gb": round(psutil.virtual_memory().total / (1024**3)),
        "cpu_count": psutil.cpu_count(),
        "has_cuda": torch.cuda.is_available(),
        "cuda_memory": torch.cuda.get_device_properties(0).total_memory / (1024**3) if torch.cuda.is_available() else 0
    }

```

```

def get_optimization_config(self):
    """Configuración de optimización según dispositivo"""
    ram_gb = self.device_info["ram_gb"]

```

```

    if ram_gb < 4:
        return {
            "load_in_8bit": True,
            "device_map": "cpu",
            "torch_dtype": torch.float16,
            "low_cpu_mem_usage": True
        }

```

```

    elif ram_gb < 8:
        return {
            "load_in_8bit": True,
            "device_map": "auto",
            "torch_dtype": torch.float16,
            "low_cpu_mem_usage": True
        }

```

```

    else:
        return {
            "device_map": "auto",
            "torch_dtype": torch.float16,
            "low_cpu_mem_usage": True
        }

```

```

def select_optimal_model(self):
    """Selecciona modelo óptimo para el dispositivo"""
    ram_gb = self.device_info["ram_gb"]

    for size, model_config in self.config["models"].items():
        if ram_gb >= model_config["min_ram_gb"]:
            return model_config

    # Fallback al modelo más pequeño
    return self.config["models"]["tiny"]

def load_model(self, model_name=None):
    """Carga modelo con optimizaciones"""
    if model_name is None:
        model_config = self.select_optimal_model()
        model_name = model_config["name"]

    print(f"Cargando modelo: {model_name}")
    print(f"Dispositivo: {self.device_info}")

    try:
        # Configurar cuantización si es necesario
        quantization_config = None
        if self.optimization_config.get("load_in_8bit"):
            quantization_config = BitsAndBytesConfig(
                load_in_8bit=True,
                llm_int8_enable_fp32_cpu_offload=True
            )

        # Cargar tokenizer
        self.current_tokenizer = AutoTokenizer.from_pretrained(
            model_name,
            trust_remote_code=True,
            padding_side="left"
        )

        # Agregar pad_token si no existe
        if self.current_tokenizer.pad_token is None:
            self.current_tokenizer.pad_token = self.current_tokenizer.eos_token

        # Cargar modelo con optimizaciones
        self.current_model = AutoModelForCausalLM.from_pretrained(

```

```

        model_name,
        quantization_config=quantization_config,
        device_map=self.optimization_config["device_map"],
        torch_dtype=self.optimization_config["torch_dtype"],
        low_cpu_mem_usage=self.optimization_config["low_cpu_mem_usage"],
        trust_remote_code=True
    )

    print(f"✅ Modelo cargado exitosamente")
    return True

except Exception as e:
    print(f"❌ Error cargando modelo: {e}")
    return False

def generate_response(self, prompt, max_length=None, temperature=None, stream=False):
    """Genera respuesta del modelo"""
    if not self.current_model or not self.current_tokenizer:
        raise RuntimeError("Modelo no cargado")

    # Usar configuraciones por defecto si no se especifican
    model_config = self.select_optimal_model()
    max_length = max_length or model_config["max_length"]
    temperature = temperature or model_config["temperature"]

    # Tokenizar prompt
    inputs = self.current_tokenizer.encode(prompt, return_tensors="pt")

    # Mover a dispositivo apropiado
    if torch.cuda.is_available() and "cuda" in str(self.current_model.device):
        inputs = inputs.to(self.current_model.device)

    # Configurar generación
    generation_config = {
        "max_length": min(max_length, inputs.shape[1] + 512),
        "temperature": temperature,
        "do_sample": True,
        "pad_token_id": self.current_tokenizer.eos_token_id,
        "eos_token_id": self.current_tokenizer.eos_token_id
    }

    if stream:
        return self._generate_stream(inputs, generation_config)

```

```

else:
    return self._generate_complete(inputs, generation_config)

def _generate_complete(self, inputs, generation_config):
    """Generación completa"""
    with torch.no_grad():
        outputs = self.current_model.generate(inputs, **generation_config)

        # Decodificar solo la parte nueva
        new_tokens = outputs[0][inputs.shape[1]:]
        response = self.current_tokenizer.decode(new_tokens, skip_special_tokens=True)

    return response.strip()

def _generate_stream(self, inputs, generation_config):
    """Generación en streaming"""
    streamer = TextIteratorStreamer(
        self.current_tokenizer,
        skip_prompt=True,
        skip_special_tokens=True
    )

    generation_config["streamer"] = streamer

    # Ejecutar generación en thread separado
    generation_thread = Thread(
        target=self.current_model.generate,
        args=(inputs,),
        kwargs=generation_config
    )
    generation_thread.start()

    # Retornar generador
    for new_text in streamer:
        yield new_text

def get_memory_usage(self):
    """Obtiene uso de memoria actual"""
    process = psutil.Process()
    return {
        "ram_mb": round(process.memory_info().rss / (1024**2)),
        "ram_percent": process.memory_percent(),
        "cuda_allocated": round(torch.cuda.memory_allocated() / (1024**2)) if torch.cuda.is_available() else 0,

```



```

        "cuda_reserved": round(torch.cuda.memory_reserved() / (1024**2)) if torch.cuda.is_available() else 0
    }

def unload_model(self):
    """Descarga modelo de memoria"""
    if self.current_model:
        del self.current_model
        self.current_model = None

    if self.current_tokenizer:
        del self.current_tokenizer
        self.current_tokenizer = None

    # Limpiar caché de GPU
    if torch.cuda.is_available():
        torch.cuda.empty_cache()

    print("✅ Modelo descargado de memoria")

```

### Paso 3: Sistema de Embeddings



python

```
# core/embedding_system.py
```

```
import numpy as np
import faiss
from sentence_transformers import SentenceTransformer
import pickle
import json
from pathlib import Path
from typing import List, Dict, Any
```

```
class UniversalEmbeddingSystem:
```

```
    def __init__(self, model_name="all-MiniLM-L6-v2", index_path="models/knowledge_base"):
        self.model_name = model_name
        self.index_path = Path(index_path)
        self.index_path.mkdir(parents=True, exist_ok=True)

        self.embedding_model = None
        self.faiss_index = None
        self.documents = []
        self.metadata = []

        self.load_embedding_model()
        self.load_existing_index()
```

```
    def load_embedding_model(self):
        """Carga modelo de embeddings optimizado"""
        try:
            print(f"Cargando modelo de embeddings: {self.model_name}")
            self.embedding_model = SentenceTransformer(self.model_name)
            print("✅ Modelo de embeddings cargado")
        except Exception as e:
            print(f"❌ Error cargando embeddings: {e}")
            # Fallback a modelo más ligero
            try:
                self.embedding_model = SentenceTransformer("all-MiniLM-L6-v2")
                print("✅ Modelo fallback cargado")
            except:
                print("❌ No se pudo cargar ningún modelo de embeddings")
```

```
    def load_existing_index(self):
        """Carga índice existente si está disponible"""
        index_file = self.index_path / "embeddings.faiss"
        metadata_file = self.index_path / "metadata.pkl"
```

```

if index_file.exists() and metadata_file.exists():
    try:
        self.faiss_index = faiss.read_index(str(index_file))
        with open(metadata_file, 'rb') as f:
            self.metadata = pickle.load(f)
        print(f"✅ Índice cargado: {self.faiss_index.ntotal} documentos")
    except Exception as e:
        print(f"⚠️ Error cargando índice: {e}")

def add_documents(self, documents: List[str], metadata: List[Dict[str, Any]] = None):
    """Agrega documentos al índice"""
    if not self.embedding_model:
        print("❌ Modelo de embeddings no disponible")
        return False

    print(f"Procesando {len(documents)} documentos...")

    # Crear embeddings
    embeddings = self.embedding_model.encode(
        documents,
        show_progress_bar=True,
        convert_to_numpy=True
    )

    # Crear índice si no existe
    if self.faiss_index is None:
        dimension = embeddings.shape[1]
        self.faiss_index = faiss.IndexFlatIP(dimension)

    # Normalizar embeddings para cosine similarity
    faiss.normalize_L2(embeddings)

    # Agregar al índice
    self.faiss_index.add(embeddings.astype('float32'))

    # Agregar documentos y metadata
    self.documents.extend(documents)
    if metadata:
        self.metadata.extend(metadata)
    else:
        # Crear metadata básica
        for i, doc in enumerate(documents):

```

```

        self.metadata.append({
            "id": len(self.metadata) + i,
            "text": doc[:200] + "..." if len(doc) > 200 else doc,
            "length": len(doc)
        })

    print(f"✅ {len(documents)} documentos agregados al índice")
    return True

def search(self, query: str, top_k: int = 5, threshold: float = 0.5):
    """Busca documentos similares"""
    if not self.embedding_model or not self.faiss_index:
        return []

    # Crear embedding de la consulta
    query_embedding = self.embedding_model.encode([query], convert_to_numpy=True)
    faiss.normalize_L2(query_embedding)

    # Buscar documentos similares
    scores, indices = self.faiss_index.search(query_embedding.astype('float32'), top_k)

    results = []
    for score, idx in zip(scores[0], indices[0]):
        if score > threshold and idx < len(self.metadata):
            results.append({
                "score": float(score),
                "metadata": self.metadata[idx],
                "text": self.documents[idx] if idx < len(self.documents) else ""
            })

    return results

def save_index(self):
    """Guarda índice en disco"""
    if self.faiss_index:
        index_file = self.index_path / "embeddings.faiss"
        metadata_file = self.index_path / "metadata.pkl"

        faiss.write_index(self.faiss_index, str(index_file))
        with open(metadata_file, 'wb') as f:
            pickle.dump(self.metadata, f)

    print(f"✅ Índice guardado: {self.faiss_index.ntotal} documentos")

```

```
    return True
return False
```

```
def build_from_directory(self, directory_path: str, file_extensions: List[str] = None):
```

```
    """Construye índice desde directorio de archivos"""
```

```
    directory = Path(directory_path)
```

```
    if not directory.exists():
```

```
        print(f"❌ Directorio no encontrado: {directory_path}")
```

```
        return False
```

```
    file_extensions = file_extensions or ['.txt', '.md', '.json']
```

```
    documents = []
```

```
    metadata = []
```

```
    for file_path in directory.rglob("*"):
```

```
        if file_path.is_file() and file_path.suffix.lower() in file_extensions:
```

```
            try:
```

```
                with open(file_path, 'r', encoding='utf-8') as f:
```

```
                    content = f.read()
```

```
                # Dividir documentos largos en chunks
```

```
                chunks = self.chunk_text(content)
```

```
                for i, chunk in enumerate(chunks):
```

```
                    documents.append(chunk)
```

```
                    metadata.append({
```

```
                        "file_path": str(file_path),
```

```
                        "file_name": file_path.name,
```

```
                        "chunk_id": i,
```

```
                        "total_chunks": len(chunks),
```

```
                        "text_preview": chunk[:200] + "..." if len(chunk) > 200 else chunk
```

```
                    })
```

```
            except Exception as e:
```

```
                print(f"⚠️ Error leyendo {file_path}: {e}")
```

```
    if documents:
```

```
        self.add_documents(documents, metadata)
```

```
        self.save_index()
```

```
        return True
```

```
print("❌ No se encontraron documentos válidos")
```

```
return False
```

```
def chunk_text(self, text: str, chunk_size: int = 500, overlap: int = 50):
```

```
    """Divide texto en chunks con overlap"""
```

```
    words = text.split()
```

```
    chunks = []
```

```
    for i in range(0, len(words), chunk_size - overlap):
```

```
        chunk_words = words[i:i + chunk_size]
```

```
        chunk_text = " ".join(chunk_words)
```

```
        if len(chunk_text.strip()) > 50: # Filtrar chunks muy pequeños
```

```
            chunks.append(chunk_text)
```

```
    return chunks if chunks else [text]
```

## Paso 4: API Flask Universal



python

```

# app.py
from flask import Flask, request, jsonify, render_template
from flask_cors import CORS
import json
from pathlib import Path

from core.model_manager import UniversalModelManager
from core.embedding_system import UniversalEmbeddingSystem

class UniversalAIApp:
    def __init__(self, debug=False):
        self.app = Flask(__name__,
                        template_folder='frontend/web',
                        static_folder='frontend/web')

        CORS(self.app)
        self.app.config['MAX_CONTENT_LENGTH'] = 50 * 1024 * 1024 # 50MB

        # Inicializar componentes
        self.model_manager = UniversalModelManager()
        self.embedding_system = UniversalEmbeddingSystem()

        # Cargar modelo por defecto
        self.model_loaded = self.model_manager.load_model()

        self.setup_routes()

    def setup_routes(self):
        """Configura rutas de la API"""

        @self.app.route('/')
        def index():
            return render_template('index.html')

        @self.app.route('/api/status')
        def api_status():
            """Estado del sistema"""
            return jsonify({
                "model_loaded": self.model_loaded,
                "memory_usage": self.model_manager.get_memory_usage() if self.model_loaded else None,
                "device_info": self.model_manager.device_info,
                "embedding_ready": self.embedding_system.faiss_index is not None
            })

```

```
}}
```

```
@self.app.route('/api/chat', methods=['POST'])
def api_chat():
    """Endpoint de chat principal"""
    if not self.model_loaded:
        return jsonify({"error": "Modelo no cargado"}), 500

    data = request.get_json()
    message = data.get('message', "").strip()

    if not message:
        return jsonify({"error": "Mensaje requerido"}), 400

    try:
        # Buscar contexto relevante si está disponible
        context = ""
        if self.embedding_system.faiss_index:
            search_results = self.embedding_system.search(message, top_k=3)
            if search_results:
                context = "\n\nContexto relevante:\n"
                for result in search_results:
                    context += f"- {result['metadata']['text_preview']}\n"

        # Crear prompt con contexto
        full_prompt = f"{context}\n\nUsuario: {message}\nAsistente:"

        # Generar respuesta
        response = self.model_manager.generate_response(full_prompt)

        return jsonify({
            "response": response,
            "context_used": bool(context),
            "memory_usage": self.model_manager.get_memory_usage()
        })

    except Exception as e:
        return jsonify({"error": str(e)}), 500

@self.app.route('/api/upload_documents', methods=['POST'])
def api_upload_documents():
    """Subida de documentos para base de conocimientos"""
    if 'files' not in request.files:
```



```

        return jsonify({"error": "No hay archivos"}), 400

files = request.files.getlist('files')
uploaded_count = 0

documents = []
metadata = []

for file in files:
    if file.filename:
        try:
            content = file.read().decode('utf-8')
            documents.append(content)
            metadata.append({
                "filename": file.filename,
                "size": len(content),
                "type": "uploaded_document"
            })
            uploaded_count += 1
        except Exception as e:
            print(f"Error procesando {file.filename}: {e}")

if documents:
    success = self.embedding_system.add_documents(documents, metadata)
    if success:
        self.embedding_system.save_index()
        return jsonify({
            "message": f"{uploaded_count} documentos procesados",
            "total_documents": self.embedding_system.faiss_index.ntotal
        })

    return jsonify({"error": "No se pudieron procesar los documentos"}), 500

@self.app.route('/api/search', methods=['POST'])
def api_search():
    """Búsqueda en base de conocimientos"""
    data = request.get_json()
    query = data.get('query', "").strip()

    if not query:
        return jsonify({"error": "Query requerido"}), 400

    results = self.embedding_system.search(query, top_k=5)

```

```
        return jsonify({
            "query": query,
            "results": results,
            "total": len(results)
        })

def run(self, host='0.0.0.0', port=5000):
    """Ejecuta la aplicación"""
    print(f"Iniciando aplicación en http://{host}:{port}")
    self.app.run(host=host, port=port, debug=False)

if __name__ == "__main__":
    app = UniversalAIApp()
    app.run()
```

## Paso 5: Frontend Web Universal



html

```
<!-- frontend/web/index.html -->
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Mi App IA Local</title>
  <link rel="manifest" href="manifest.json">
  <meta name="theme-color" content="#2196F3">
  <style>
    * { margin: 0; padding: 0; box-sizing: border-box; }

    body {
      font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif;
      background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
      min-height: 100vh;
      padding: 20px;
    }

    .container {
      max-width: 800px;
      margin: 0 auto;
      background: white;
      border-radius: 15px;
      box-shadow: 0 20px 40px rgba(0,0,0,0.1);
      overflow: hidden;
    }

    .header {
      background: linear-gradient(45deg, #2196F3, #21CBF3);
      color: white;
      padding: 30px;
      text-align: center;
    }

    .status-bar {
      display: flex;
      justify-content: space-between;
      align-items: center;
      padding: 15px 20px;
      background: #f5f5f5;
      border-bottom: 1px solid #ddd;
    }
```

```
}

.status-indicator {
  display: flex;
  align-items: center;
  gap: 10px;
}

.status-dot {
  width: 12px;
  height: 12px;
  border-radius: 50%;
  background: #4CAF50;
}

.status-dot.error { background: #f44336; }
.status-dot.warning { background: #ff9800; }

.chat-container {
  height: 400px;
  overflow-y: auto;
  padding: 20px;
  background: #fafafa;
}

.message {
  margin-bottom: 15px;
  padding: 12px 16px;
  border-radius: 18px;
  max-width: 80%;
  word-wrap: break-word;
}

.user-message {
  background: #2196F3;
  color: white;
  margin-left: auto;
}

.ai-message {
  background: white;
  color: #333;
  border: 1px solid #e0e0e0;
```

```
}
```

```
.input-area {  
  padding: 20px;  
  background: white;  
  border-top: 1px solid #ddd;  
}
```

```
.input-group {  
  display: flex;  
  gap: 10px;  
}
```

```
.message-input {  
  flex: 1;  
  padding: 12px 16px;  
  border: 2px solid #ddd;  
  border-radius: 25px;  
  outline: none;  
  font-size: 14px;  
}
```

```
.message-input:focus {  
  border-color: #2196F3;  
}
```

```
.send-btn {  
  background: #2196F3;  
  color: white;  
  border: none;  
  padding: 12px 24px;  
  border-radius: 25px;  
  cursor: pointer;  
  font-weight: bold;  
  transition: background 0.3s;  
}
```

```
.send-btn:hover {  
  background: #1976D2;  
}
```

```
.send-btn.disabled {  
  background: #ccc;
```

```
    cursor: not-allowed;
}

.tools-section {
    padding: 20px;
    background: white;
    border-top: 1px solid #ddd;
}

.file-upload {
    margin-bottom: 20px;
}

.upload-area {
    border: 2px dashed #ddd;
    border-radius: 10px;
    padding: 30px;
    text-align: center;
    cursor: pointer;
    transition: border-color 0.3s;
}

.upload-area:hover {
    border-color: #2196F3;
}

.upload-area.dragover {
    border-color: #4CAF50;
    background: #f0f8ff;
}

.hidden { display: none; }

.loading {
    display: flex;
    align-items: center;
    gap: 10px;
    color: #666;
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
```

```

    }

    .spinner {
        width: 20px;
        height: 20px;
        border: 2px solid #f3f3f3;
        border-top: 2px solid #2196F3;
        border-radius: 50%;
        animation: spin 1s linear infinite;
    }

    @media (max-width: 600px) {
        .container { margin: 10px; }
        .header h1 { font-size: 1.5em; }
        .message { max-width: 95%; }
    }
</style>
</head>
<body>
    <div class="container">
        <div class="header">
            <h1>Mi Asistente IA Local</h1>
            <p>Powered by modelos de pesos libres</p>
        </div>

        <div class="status-bar">
            <div class="status-indicator">
                <div class="status-dot" id="statusDot"></div>
                <span id="statusText">Iniciando...</span>
            </div>
            <div id="memoryUsage">RAM: --</div>
        </div>

        <div class="chat-container" id="chatContainer">
            <div class="ai-message">
                Hola! Soy tu asistente IA local. Estoy ejecutándome completamente en tu dispositivo
                sin enviar datos a servidores externos. ¿En qué puedo ayudarte?
            </div>
        </div>

        <div class="input-area">
            <div class="input-group">
                <input type="text"

```

```

        class="message-input"
        id="messageInput"
        placeholder="Escribe tu mensaje..."
        maxLength="500">
        <button class="send-btn" id="sendBtn" onclick="sendMessage()">Enviar</button>
    </div>
</div>

<div class="tools-section">
    <div class="file-upload">
        <h3>Subir Documentos</h3>
        <div class="upload-area" id="uploadArea">
            <p>Arrastra archivos aquí o haz clic para seleccionar</p>
            <p style="font-size: 12px; color: #666;">Formatos: TXT, MD, JSON</p>
            <input type="file" id="fileInput" multiple accept=".txt,.md,.json" class="hidden">
        </div>
    </div>
</div>
</div>

<script>
class UniversalAIApp {
    constructor() {
        this.apiBase = '/api';
        this.isLoading = false;
        this.init();
    }

    async init() {
        this.setupEventListeners();
        await this.checkStatus();
        this.setupFileUpload();
    }

    setupEventListeners() {
        document.getElementById('messageInput').addEventListener('keypress', (e) => {
            if (e.key === 'Enter' && !e.shiftKey) {
                e.preventDefault();
                this.sendMessage();
            }
        });
    }
}

```



```

async checkStatus() {
  try {
    const response = await fetch(`${this.apiBase}/status`);
    const status = await response.json();

    this.updateStatusIndicator(status);

    if (status.model_loaded) {
      this.updateStatusText('Modelo cargado - Listo para usar');
      this.updateMemoryUsage(status.memory_usage);
    } else {
      this.updateStatusText('Error: Modelo no cargado', 'error');
    }
  } catch (error) {
    this.updateStatusText('Error de conexión', 'error');
    console.error('Error checking status:', error);
  }
}

updateStatusIndicator(status) {
  const dot = document.getElementById('statusDot');
  const text = document.getElementById('statusText');

  if (status.model_loaded) {
    dot.className = 'status-dot';
    text.textContent = 'Sistema listo';
  } else {
    dot.className = 'status-dot error';
    text.textContent = 'Sistema no disponible';
  }
}

updateStatusText(text, type = 'success') {
  const statusText = document.getElementById('statusText');
  const statusDot = document.getElementById('statusDot');

  statusText.textContent = text;
  statusDot.className = `status-dot ${type === 'error' ? 'error' : ''}`;
}

updateMemoryUsage(memoryInfo) {
  if (memoryInfo) {

```

```

    const memoryElement = document.getElementById('memoryUsage');
    memoryElement.textContent = `RAM: ${memoryInfo.ram_mb}MB (${memoryInfo.ram_percent.toFixed(
  }
}

async sendMessage() {
  const input = document.getElementById('messageInput');
  const message = input.value.trim();

  if (!message || this.isLoading) return;

  // Mostrar mensaje del usuario
  this.addMessage(message, 'user');
  input.value = '';

  // Mostrar indicador de carga
  this.setLoading(true);

  try {
    const response = await fetch(`${this.apiBase}/chat`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ message })
    });

    const data = await response.json();

    if (response.ok) {
      this.addMessage(data.response, 'ai');

      // Actualizar uso de memoria
      if (data.memory_usage) {
        this.updateMemoryUsage(data.memory_usage);
      }

      // Indicar si se usó contexto
      if (data.context_used) {
        this.addMessage('(Respuesta generada usando contexto de documentos)', 'system');
      }
    } else {
      this.addMessage(`Error: ${data.error}`, 'error');
    }
  }
}

```

```

    }

    } catch (error) {
        this.addMessage('Error de conexión con el modelo', 'error');
        console.error('Error:', error);
    }

    this.setLoading(false);
}

addMessage(text, type) {
    const chatContainer = document.getElementById('chatContainer');
    const messageDiv = document.createElement('div');

    let className = 'message';
    switch(type) {
        case 'user':
            className += 'user-message';
            break;
        case 'ai':
            className += 'ai-message';
            break;
        case 'error':
            className += 'ai-message';
            text = '❌ ' + text;
            break;
        case 'system':
            className += 'ai-message';
            text = 'ℹ️ ' + text;
            break;
    }

    messageDiv.className = className;
    messageDiv.textContent = text;

    chatContainer.appendChild(messageDiv);
    chatContainer.scrollTop = chatContainer.scrollHeight;
}

setLoading(loading) {
    this.isLoading = loading;
    const sendBtn = document.getElementById('sendBtn');
    const messageInput = document.getElementById('messageInput');

```

```

if (loading) {
  sendBtn.disabled = true;
  sendBtn.innerHTML = '<div class="spinner"></div>';
  messageInput.disabled = true;

  // Mostrar mensaje de carga
  const loadingDiv = document.createElement('div');
  loadingDiv.className = 'message ai-message loading-message';
  loadingDiv.innerHTML = '<div class="loading"><div class="spinner"></div>Generando respuesta...</div>';
  document.getElementById('chatContainer').appendChild(loadingDiv);
} else {
  sendBtn.disabled = false;
  sendBtn.textContent = 'Enviar';
  messageInput.disabled = false;

  // Remover mensaje de carga
  const loadingMsg = document.querySelector('.loading-message');
  if (loadingMsg) {
    loadingMsg.remove();
  }
}

}

setupFileUpload() {
  const uploadArea = document.getElementById('uploadArea');
  const fileInput = document.getElementById('fileInput');

  uploadArea.addEventListener('click', () => fileInput.click());

  uploadArea.addEventListener('dragover', (e) => {
    e.preventDefault();
    uploadArea.classList.add('dragover');
  });

  uploadArea.addEventListener('dragleave', () => {
    uploadArea.classList.remove('dragover');
  });

  uploadArea.addEventListener('drop', (e) => {
    e.preventDefault();
    uploadArea.classList.remove('dragover');
    this.handleFileUpload(e.dataTransfer.files);
  });
}

```

```

    });

    fileInput.addEventListener('change', (e) => {
        this.handleFileUpload(e.target.files);
    });
}

async handleFileUpload(files) {
    if (files.length === 0) return;

    const formData = new FormData();
    Array.from(files).forEach(file => {
        formData.append('files', file);
    });

    this.addMessage(`Subiendo ${files.length} archivo(s)...`, 'system');

    try {
        const response = await fetch(`${this.apiBase}/upload_documents`, {
            method: 'POST',
            body: formData
        });

        const data = await response.json();

        if (response.ok) {
            this.addMessage(`✅ ${data.message}. Total en base de datos: ${data.total_documents}`, 'system');
        } else {
            this.addMessage(`❌ Error: ${data.error}`, 'error');
        }
    } catch (error) {
        this.addMessage(`❌ Error subiendo archivos`, 'error');
        console.error('Upload error:', error);
    }
}

// Inicializar aplicación
document.addEventListener('DOMContentLoaded', () => {
    new UniversalAIApp();
});

```

```

// Registrar Service Worker para PWA
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/sw.js')
    .then(registration => console.log('SW registered'))
    .catch(error => console.log('SW registration failed'));
}
</script>
</body>
</html>

```

## Paso 6: Configuración PWA



json

```

// frontend/web/manifest.json
{
  "name": "Mi Asistente IA Local",
  "short_name": "IA Local",
  "description": "Asistente IA completamente local con modelos de pesos libres",
  "start_url": "/",
  "display": "standalone",
  "background_color": "#667eea",
  "theme_color": "#2196F3",
  "orientation": "any",
  "icons": [
    {
      "src": "icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "icon-512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "categories": ["productivity", "ai", "assistant"],
  "lang": "es"
}

```

---

# Optimización para Dispositivos

## Perfiles por Dispositivo



json

```
// config/device_profiles.json
```

```
{
  "mobile_low": {
    "ram_limit_gb": 4,
    "recommended_models": ["TinyLlama/TinyLlama-1.1B-Chat-v1.0"],
    "optimizations": {
      "load_in_8bit": true,
      "max_length": 1024,
      "batch_size": 1
    }
  },

  "mobile_high": {
    "ram_limit_gb": 8,
    "recommended_models": ["microsoft/phi-2"],
    "optimizations": {
      "load_in_8bit": true,
      "max_length": 2048,
      "batch_size": 2
    }
  },

  "laptop": {
    "ram_limit_gb": 16,
    "recommended_models": ["mistralai/Mistral-7B-Instruct-v0.2"],
    "optimizations": {
      "load_in_8bit": false,
      "max_length": 4096,
      "batch_size": 4
    }
  },

  "desktop": {
    "ram_limit_gb": 32,
    "recommended_models": ["meta-llama/Llama-2-7b-chat-hf"],
    "optimizations": {
      "load_in_8bit": false,
      "max_length": 8192,
      "batch_size": 8
    }
  }
}
```



```
}  
}
```

---

## Casos de Uso Específicos

### 1. Aplicación Legal

- Modelos especializados en derecho
- Base de conocimientos con códigos y jurisprudencia
- Generación de documentos legales

### 2. Aplicación Médica

- Modelos fine-tuned con literatura médica
- Diagnóstico asistido (no reemplaza profesional)
- Base de conocimientos con síntomas y tratamientos

### 3. Aplicación Educativa

- Tutor personalizado por materia
- Generación de ejercicios y exámenes
- Explicaciones adaptadas al nivel del estudiante

### 4. Aplicación Técnica/Ingeniería

- Asistente para programación
- Documentación técnica especializada
- Solución de problemas específicos

### 5. Aplicación Creativa

- Generación de contenido creativo
  - Asistente de escritura
  - Brainstorming y desarrollo de ideas
- 

## Instalación Rápida

### Para Desarrolladores



bash

*# 1. Clonar estructura base*

```
git clone <tu-repo>
```

```
cd mi-app-ia
```

*# 2. Ejecutar setup automático*

```
python scripts/setup.py
```

*# 3. Iniciar aplicación*

```
python app.py
```

## Para Usuarios Finales



bash

*# 1. Descargar release*

```
wget <link-del-release>
```

```
unzip mi-app-ia.zip
```

*# 2. Ejecutar instalador*

```
./install.sh # Linux/Mac
```

```
install.bat # Windows
```

*# 3. Abrir aplicación*

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

---

## Troubleshooting

### Problemas Comunes

#### Error: Memoria Insuficiente



bash

*# Solución: Usar modelo más pequeño*

```
python scripts/optimize_for_device.py --force-tiny-model
```

#### Error: Modelo No Carga



bash

*# Limpiar cache y recargar*

```
rm -rf models/ai_models/*
```

```
python scripts/download_models.py --force-download
```

## Error: Aplicación Lenta



bash

*# Optimizar para dispositivo actual*

```
python scripts/optimize_for_device.py --auto-optimize
```

## Logs y Debugging



bash

*# Ver logs en tiempo real*

```
tail -f logs/app.log
```

*# Modo debug*

```
python app.py --debug
```

*# Verificar estado del sistema*

```
python -c "from core.model_manager import *; UniversalModelManager().get_device_info()"
```

---

## Personalización Avanzada

### Agregar Nuevo Modelo



python

*# En config/model\_configs.json*

```
{  
  "models": {  
    "mi_modelo_custom": {  
      "name": "ruta/a/mi/modelo",  
      "max_length": 2048,  
      "temperature": 0.7,  
      "min_ram_gb": 8,  
      "specialty": "dominio_específico"  
    }  
  }  
}
```

## Fine-tuning para Dominio Específico



python

```
# scripts/fine_tune_model.py
from transformers import AutoModelForCausalLM, TrainingArguments, Trainer

def fine_tune_for_domain(base_model, training_data):
    model = AutoModelForCausalLM.from_pretrained(base_model)

    training_args = TrainingArguments(
        output_dir="./fine_tuned_model",
        num_train_epochs=3,
        per_device_train_batch_size=4,
        gradient_accumulation_steps=2,
        warmup_steps=100,
        logging_steps=10,
    )

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=training_data,
    )

    trainer.train()
    trainer.save_model()
```

---

## Conclusión

Esta arquitectura universal permite crear aplicaciones IA locales robustas y escalables que funcionan completamente offline. La modularidad del sistema facilita la adaptación a diferentes dominios y dispositivos, mientras mantiene la privacidad y elimina dependencias externas.

### Beneficios Clave:

- **100% Local:** Privacidad y control total
- **Escalable:** Desde móviles hasta servidores
- **Modular:** Fácil de extender y personalizar
- **Optimizado:** Rendimiento adaptativo por dispositivo
- **Gratuito:** Sin costos de APIs o suscripciones

### Próximos Pasos:

1. Implementar la estructura base
2. Seleccionar modelo apropiado para tu caso de uso
3. Personalizar para tu dominio específico
4. Optimizar para tus dispositivos objetivo
5. Desplegar y distribuir

¿Listo para crear tu propia aplicación IA local?