IACADEMY

DE FUNDAMENTOS A ARQUITECTURA DE IA

MÓDULO 29

Proyecto final guiado

Avanzado

iacedemy.com — 2026

MÓDULO 29

Proyecto final guiado

Nivel: Avanzado

Autor: David Moya

Publicación: Mayo 2026

Plataforma: iacedemy.com

Elegir tu proyecto: los 3 criterios

El proyecto final no es un ejercicio academico. Es la pieza que demuestra que puedes construir algo real con IA. Para que funcione, tu proyecto debe cumplir tres criterios.

Criterio 1: Util

Resuelve un problema real que tu o alguien cercano tiene. No construyas "un chatbot porque es facil". Construye algo que te ahorre tiempo, automatice una tarea repetitiva, o genere valor medible. Pregunta: "Si esto existiera manana, lo usaria alguien?"

Criterio 2: Construible en 2-4 semanas

No es un producto completo. Es un MVP funcional. Si la idea necesita 6 meses de desarrollo, reducela hasta que quepa en 2-4 semanas de trabajo real (no tiempo completo, bloques de 2 horas). Si no puedes explicar el scope en 3 frases, es demasiado grande.

Criterio 3: Demostrable en 30 segundos

El test definitivo: si alguien abre tu proyecto, debe entender que hace y ver el valor en 30 segundos o menos. Si necesitas 5 minutos de explicacion antes de que alguien lo "pille", el proyecto no es demostrable. Esto es critico para tu portfolio.

> **Ejemplos que cumplen los 3 criterios**
>
> - **Generador de emails de seguimiento**: pega el contexto de una reunion, genera un email de follow-up profesional. Util, 2 semanas, demo instantanea.
> - **Analizador de CVs para recruiters**: sube un CV, obtiene un resumen estructurado con puntos fuertes y debiles. Util, 3 semanas, demo visual.
> - **Asistente de estudio personal**: sube apuntes, genera flashcards y preguntas tipo test. Util, 3 semanas, demo interactiva.
> - **Clasificador de tickets de soporte**: recibe un ticket, clasifica prioridad y departamento, sugiere respuesta. Util, 2 semanas, demo clara.

Arquitectura: planificar antes de construir

Antes de escribir una linea de codigo, dedica 1-2 horas a planificar. La planificacion con IA es diferente: usas Claude Code en Plan Mode para iterar sobre la arquitectura antes de ejecutar.

Plan Mode: pensar antes de hacer

```
# En Claude Code, activa Plan Mode:
# Shift+Tab para cambiar a Plan mode

# Prompt de planificacion:
"Quiero construir un [descripcion del proyecto].
El usuario hace [accion principal].
El sistema responde con [output esperado].

Ayudame a disenar:
1. El flujo de datos completo (input -> procesamiento -> output)
2. El modelo de datos (que tablas/colecciones necesito)
3. Las APIs necesarias (endpoints y sus responsabilidades)
4. Donde entra el LLM en el flujo
5. Que puede fallar y como manejarlo"
```

El output de Plan Mode es un documento de arquitectura informal. No necesitas UML ni diagramas formales. Necesitas claridad sobre que hace cada pieza.

Estructura del plan

```
# plan.md (ejemplo: Analizador de CVs)

## Flujo principal
1. Usuario sube PDF del CV
2. Backend extrae texto del PDF (PyPDF2)
3. Texto se envia al LLM con prompt estructurado
4. LLM devuelve JSON con: resumen, skills, experiencia, puntuacion
5. Frontend muestra resultado en tarjetas visuales

## Modelo de datos
- users: id, email, created_at
- analyses: id, user_id, filename, raw_text, result_json, created_at

## Endpoints
- POST /api/upload -> sube PDF, extrae texto, lanza analisis
- GET /api/analyses -> lista analisis del usuario
- GET /api/analyses/:id -> detalle de un analisis

## Integracion LLM
- Modelo: Claude API (Haiku para rapido, Sonnet para detalle)
- Prompt: system prompt con formato JSON estricto
- Fallback: si el LLM falla, guardar raw_text y reintentar

## Riesgos
- PDFs escaneados (imagenes, no texto) -> avisar al usuario
- Rate limits de la API -> queue con retry
- CVs muy largos -> truncar a 4000 tokens
```

El stack recomendado

Para el proyecto final, recomendamos un stack que ya conoces de modulos anteriores y que te permite ir rapido sin sacrificar calidad.

CAPA	TECNOLOGIA	POR QUE
Base de datos	Supabase (PostgreSQL)	Auth gratis, RLS, API auto-generada, dashboard visual
Backend	FastAPI (Python)	Async, tipado, rapido de escribir, ideal para wrappers de LLM
Frontend	Next.js (React)	SSR, file-based routing, deploy en Cloudflare Pages
LLM	Claude API o OpenAI	API estable, buena documentacion, SDKs oficiales
Deploy	Cloudflare Pages + VPS	Frontend gratis en CF, backend en VPS si es necesario

No necesitas usar todo. Si tu proyecto es solo backend (una CLI, un bot, una API), no necesitas Next.js. Si es solo frontend (una SPA que llama directamente a la API del LLM), no necesitas FastAPI. Usa lo minimo necesario.

Alternativas validas

- **Solo frontend**: Next.js + Supabase + API de LLM directa (con edge functions para ocultar la key)
- **Solo backend**: FastAPI + CLI o Telegram bot + Supabase para persistencia
- **Full stack ligero**: FastAPI + templates HTML (sin React) + SQLite local

Fase de construccion

Trabaja en bloques de 2 horas. Cada bloque debe completar una feature end-to-end: desde el input del usuario hasta ver el resultado. No hagas todas las rutas primero, luego todo el frontend, luego toda la logica. Eso deja el proyecto "casi hecho" durante semanas.

Bloque 1: Auth + modelo de datos

```
# Supabase: crear tabla principal
CREATE TABLE analyses (
  id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
  user_id UUID REFERENCES auth.users(id) NOT NULL,
  filename TEXT NOT NULL,
  raw_text TEXT,
  result_json JSONB,
  status TEXT DEFAULT 'pending' CHECK (status IN ('pending', 'processing', 'done', 'error')),
  created_at TIMESTAMPTZ DEFAULT now()
);

-- RLS obligatorio
ALTER TABLE analyses ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Users see own analyses"
  ON analyses FOR SELECT
  USING (auth.uid() = user_id);

CREATE POLICY "Users create own analyses"
  ON analyses FOR INSERT
  WITH CHECK (auth.uid() = user_id);
```

```
# FastAPI: endpoint basico
from fastapi import FastAPI, UploadFile, Depends
from supabase import create_client

app = FastAPI()

@app.post("/api/upload")
async def upload_cv(file: UploadFile, user_id: str = Depends(get_current_user)):
    # Extraer texto del PDF
    text = extract_text(file)
    # Guardar en Supabase
    result = supabase.table("analyses").insert({
        "user_id": user_id,
        "filename": file.filename,
        "raw_text": text,
        "status": "pending"
    }).execute()
    # Lanzar analisis async
    background_tasks.add_task(analyze_cv, result.data[0]["id"], text)
    return {"id": result.data[0]["id"], "status": "pending"}
```

Bloque 2: Core flow (input, LLM, output)

```
import anthropic

client = anthropic.Anthropic()

SYSTEM_PROMPT = """Eres un analizador de CVs experto.
Dado el texto de un CV, devuelve un JSON con esta estructura exacta:
{
  "resumen": "3 frases sobre el candidato",
  "skills": ["skill1", "skill2", ...],
  "experiencia_anos": numero,
  "puntos_fuertes": ["punto1", "punto2", "punto3"],
  "areas_mejora": ["area1", "area2"],
  "puntuacion": numero del 1 al 10
}
Solo devuelve el JSON, sin texto adicional."""

async def analyze_cv(analysis_id: str, text: str):
    try:
        response = client.messages.create(
            model="claude-sonnet-4-20250514",
            max_tokens=1024,
            system=SYSTEM_PROMPT,
            messages=[{"role": "user", "content": f"Analiza este CV:\n\n{text[:4000]}"}]
        )
        result = json.loads(response.content[0].text)
        # Guardar resultado
        supabase.table("analyses").update({
            "result_json": result,
            "status": "done"
        }).eq("id", analysis_id).execute()
    except Exception as e:
        supabase.table("analyses").update({
            "status": "error"
        }).eq("id", analysis_id).execute()
```

Bloque 3: Polish (UI, errores, edge cases)

El tercer bloque es para:

- Manejar errores visibles para el usuario (no solo logs)
- Loading states (el usuario sabe que algo esta pasando)
- Edge cases: archivo vacio, PDF con imagenes, texto demasiado largo
- Un minimo de estilos (no tiene que ser bonito, pero si usable)

Testing e iteracion

Testing en un proyecto con IA tiene dos partes: testing tecnico (el codigo funciona) y testing de calidad (el LLM responde bien).

Testing tecnico: pytest

```
# tests/test_upload.py
import pytest
from fastapi.testclient import TestClient
from app.main import app

client = TestClient(app)

def test_upload_pdf():
    with open("tests/fixtures/sample_cv.pdf", "rb") as f:
        response = client.post(
            "/api/upload",
            files={"file": ("cv.pdf", f, "application/pdf")},
            headers={"Authorization": "Bearer test-token"}
        )
    assert response.status_code == 200
    assert response.json()["status"] == "pending"

def test_upload_invalid_file():
    response = client.post(
        "/api/upload",
        files={"file": ("image.png", b"not a pdf", "image/png")},
        headers={"Authorization": "Bearer test-token"}
    )
    assert response.status_code == 400

def test_get_analysis_not_found():
    response = client.get("/api/analyses/nonexistent-id")
    assert response.status_code == 404
```

Testing de calidad del LLM

```
# tests/test_llm_quality.py
"""Tests de calidad: verificar que el LLM responde con el formato esperado."""
import json

SAMPLE_CV = """Juan Garcia - Desarrollador Full Stack
5 anos de experiencia. Python, JavaScript, React, PostgreSQL.
Trabajo en TechCorp (2021-2024) como senior developer.
Certificaciones: AWS Solutions Architect, Scrum Master."""

def test_response_format():
    result = analyze_cv_sync(SAMPLE_CV)
    parsed = json.loads(result)
    # Verificar estructura
    assert "resumen" in parsed
    assert "skills" in parsed
    assert isinstance(parsed["skills"], list)
    assert "puntuacion" in parsed
    assert 1 <= parsed["puntuacion"] <= 10

def test_skills_extraction():
    result = analyze_cv_sync(SAMPLE_CV)
    parsed = json.loads(result)
    skills = [s.lower() for s in parsed["skills"]]
    # Al menos debe detectar Python y React
    assert any("python" in s for s in skills)
    assert any("react" in s for s in skills)
```

Feedback real de usuarios

El testing tecnico verifica que no se rompe. El feedback de usuarios verifica que es util. Necesitas 3-5 personas reales (no desarrolladores) que prueben tu proyecto.

Las 3 preguntas clave

1. **"Entiendes que hace?"** - Si necesitan mas de 10 segundos leyendo la landing, tienes un problema de comunicacion.
2. **"Lo usarias manana?"** - Si la respuesta es "si, pero..." lo que viene despues del "pero" es tu siguiente feature.
3. **"Que falta?"** - No preguntes "que te parece" (demasiado abierto). Pregunta que falta para que sea realmente util.

Como recoger feedback

```
# No necesitas herramientas complejas. Un JSON en Supabase:
CREATE TABLE feedback (
  id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
  user_email TEXT,
  understands BOOLEAN,        -- "Entiendes que hace?"
  would_use BOOLEAN,          -- "Lo usarias manana?"
  missing TEXT,               -- "Que falta?"
  nps INTEGER CHECK (nps >= 0 AND nps <= 10),
  created_at TIMESTAMPTZ DEFAULT now()
);
```

Envia el enlace a 5 personas. No les expliques nada antes. Observa donde se atascan. Esa informacion vale mas que 100 tests automatizados.

Documentacion y README

Tu README es la puerta de entrada al proyecto. Si alguien (un recruiter, un colega, un potencial cliente) llega a tu repo, el README decide si invierte 2 minutos mas o cierra la pestana.

Estructura de 5 secciones

````
# Nombre del Proyecto

Una frase que explica que hace y para quien.

## Para quien es

- Perfil 1 que se beneficia
- Perfil 2 que se beneficia

## Como funciona

1. El usuario hace X
2. El sistema hace Y
3. El resultado es Z

[Screenshot o GIF de demo aqui]

## Demo

Link a la demo desplegada: https://mi-proyecto.pages.dev

## Instalar y ejecutar

```bash
git clone https://github.com/tu-user/tu-proyecto
cd tu-proyecto
cp .env.example .env
# Editar .env con tus API keys
pip install -r requirements.txt
python -m uvicorn app.main:app --reload
```

## Stack

- Backend: FastAPI + Python 3.11
- DB: Supabase (PostgreSQL)
- LLM: Claude API (Sonnet)
- Deploy: Cloudflare Pages
````

El truco del GIF

Un GIF de 10 segundos mostrando el flujo principal vale mas que 500 palabras de documentacion. Graba tu pantalla con la demo funcionando, conviertelo a GIF (usa `ffmpeg` o cualquier herramienta), y ponlo despues de "Como funciona". Es lo primero que la gente mira.

Despliegue

Un proyecto sin desplegar no existe. Si alguien no puede hacer click en un enlace y ver tu proyecto funcionando, la barrera de entrada es demasiado alta.

Opcion 1: Cloudflare Pages (frontend)

```
# Para Next.js static export
# next.config.js
const nextConfig = {
  output: 'export',
  images: { unoptimized: true }
}

# Deploy
npx wrangler pages deploy out/ --project-name=mi-proyecto --branch=main
```

Opcion 2: VPS para backend

```
# docker-compose.yml minimo
version: '3.8'
services:
  api:
    build: .
    ports:
      - "8000:8000"
    env_file:
      - .env
    restart: unless-stopped

# Dockerfile
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Opcion 3: Todo en Supabase Edge Functions

```
// supabase/functions/analyze/index.ts
import { serve } from "https://deno.land/std@0.168.0/http/server.ts"

serve(async (req) => {
  const { text } = await req.json()

  const response = await fetch("https://api.anthropic.com/v1/messages", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "x-api-key": Deno.env.get("ANTHROPIC_API_KEY"),
      "anthropic-version": "2023-06-01"
    },
    body: JSON.stringify({
      model: "claude-sonnet-4-20250514",
      max_tokens: 1024,
      messages: [{ role: "user", content: `Analiza: ${text}` }]
    })
  })

  const data = await response.json()
  return new Response(JSON.stringify(data), {
    headers: { "Content-Type": "application/json" }
  })
})
```

Ejercicio practico

Ejercicio M29: Construye tu proyecto final

Este es el ejercicio mas largo del curso. No lo hagas en una sesion. Divide en bloques de 2 horas durante 2-4 semanas.

1. **Semana 1 (2h)**: Elige tu proyecto usando los 3 criterios. Escribe el plan.md con Claude Code en Plan Mode. Define el modelo de datos y los endpoints.
2. **Semana 1 (2h)**: Bloque 1. Auth + modelo de datos en Supabase. Endpoint basico que recibe input y lo guarda.
3. **Semana 2 (2h)**: Bloque 2. Core flow completo: input del usuario → LLM → resultado guardado y visible.
4. **Semana 2 (2h)**: Bloque 3. Manejo de errores, loading states, edge cases. El proyecto no se rompe con inputs inesperados.
5. **Semana 3 (2h)**: Testing. Al menos 5 tests con pytest. Al menos 2 tests de calidad del LLM.
6. **Semana 3 (1h)**: Documentacion. README con las 5 secciones. GIF de demo.
7. **Semana 3 (1h)**: Deploy. El proyecto funciona en un URL publico.
8. **Semana 4 (2h)**: Feedback. Envia a 3-5 personas. Recoge respuestas a las 3 preguntas. Itera una vez con lo aprendido.

Entregable final: Repositorio en GitHub con README completo, tests pasando, demo desplegada, y al menos 3 respuestas de feedback documentadas en un archivo `feedback.md`.

Bonus: Este proyecto cuenta como una de las 3 piezas de portfolio para la certificacion IAcademy Pro (M30).

Conclusiones clave

Key takeaways del M29

1. Tu proyecto debe cumplir 3 criterios: Util (resuelve un problema real), Construible (2-4 semanas), Demostrable (30 segundos para entenderlo).
2. Planifica antes de construir. Usa Claude Code Plan Mode para iterar sobre la arquitectura sin escribir codigo.
3. Trabaja en bloques de 2 horas, completando una feature end-to-end por bloque. Bloque 1: auth + datos. Bloque 2: core flow con LLM. Bloque 3: polish.
4. Testing tiene dos caras: tecnico (pytest, el codigo no se rompe) y calidad (el LLM responde bien, formato correcto, contenido util).
5. Feedback de 3-5 usuarios reales es obligatorio. Las 3 preguntas: entiendes que hace, lo usarias manana, que falta.
6. README de 5 secciones: que es, para quien, como funciona, demo, instalar. Un GIF de 10 segundos vale mas que 500 palabras.
7. Un proyecto sin desplegar no existe. Cloudflare Pages para frontend, VPS o Edge Functions para backend.