

05 – Despliegue



Centro de Servicios y Gestión Empresarial
SENA Regional Antioquia



www.sena.edu.co

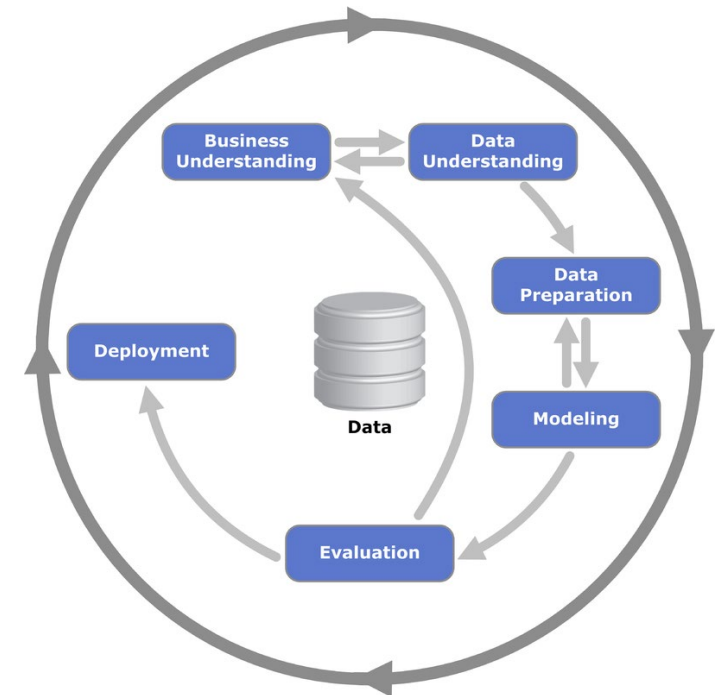


Despliegue

Despliegue

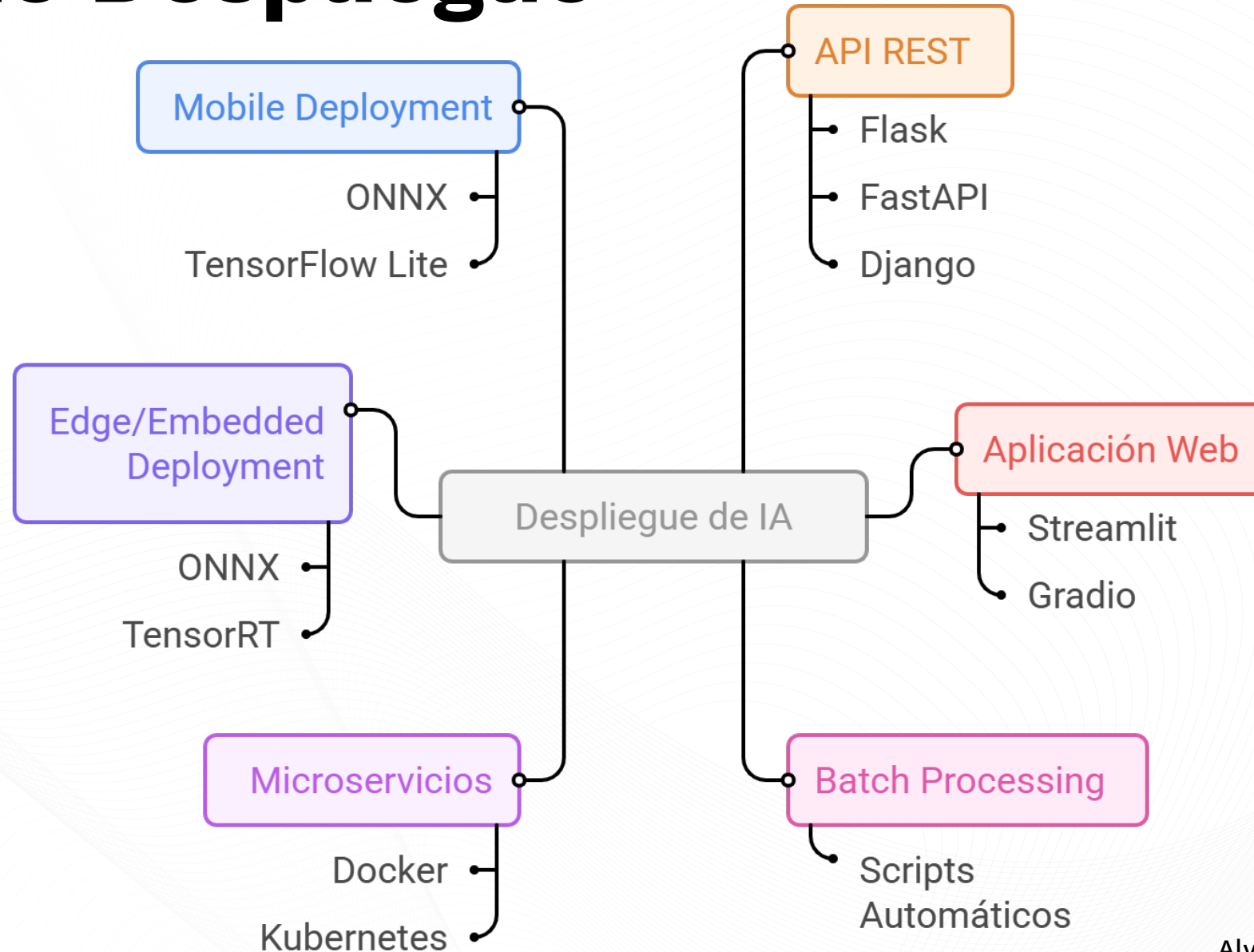
En esta fase el modelo se implementa en el entorno real donde será utilizado. Se integran los resultados a los procesos de la organización y se establecen mecanismos de monitoreo, mantenimiento y actualización para asegurar su funcionamiento continuo.

Objetivo: Poner el modelo en producción y garantizar su uso efectivo y sostenible.



Elementos de Ingeniería de Software

Tipos de Despliegue



Arquitectura



Patrón MVC

Modelo: Manejo de datos y BD

Vista: Plantillas HTML via Jinja2

Controlador: Lógica y rutas

Arquitectura REST

Cómo se diseña la API

Endpoints bien definidos

Uso correcto de métodos HTTP y estados

Respuestas en JSON

Recursos (modulo o proceso)

Microservicios

Dividir un sistema grande en múltiples servicios pequeños, independientes.

La comunican normalmente es por HTTP o mensajería.

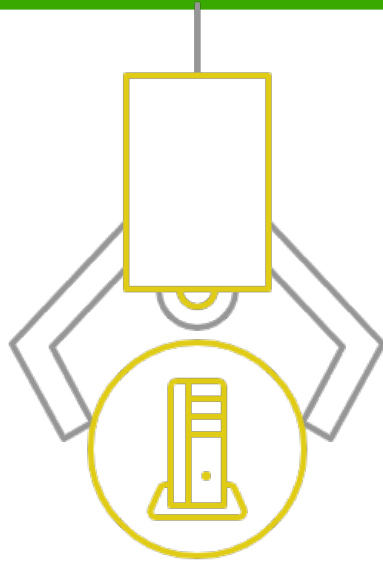
Limpia de N capas

Rutas: Recibir solicitudes y entregarlas a servicios

Servicios: Validación, cálculos, reglas de negocio

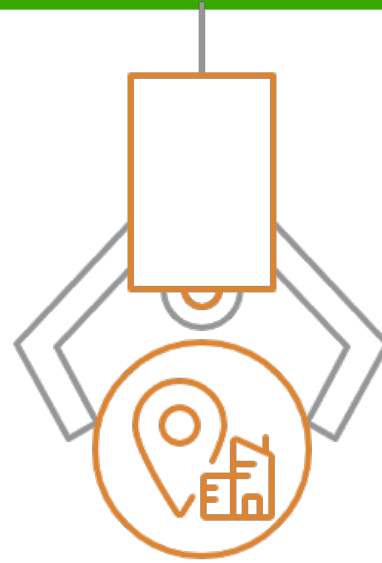
Repositorios: Interactuar con la base de datos o IA

Infraestructura necesaria



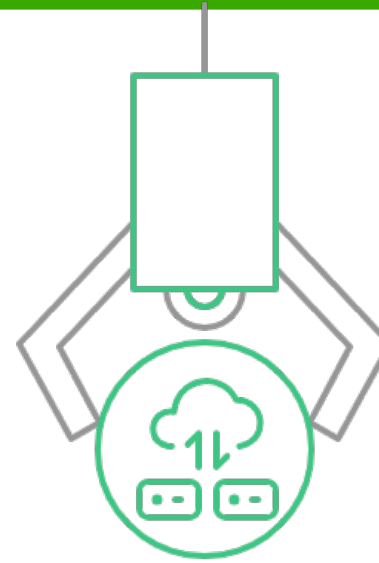
Servidor local

Alojamiento de modelos en un servidor local.



On-Premise

Alojamiento de modelos en la infraestructura propia.



Cloud

Alojamiento de modelos en la nube.



CI/CD pipelines

Uso de pipelines CI/CD para actualizaciones automáticas.

Lenguajes de Programación



- **ONNX** es un formato universal para modelos de IA que funciona en múltiples lenguajes y plataformas.
- **PMML** es un formato estándar XML para compartir modelos tradicionales de ML entre diferentes sistemas.



PKL es el formato nativo para guardar modelos en Python, pero no funciona fuera del ecosistema Python.



Lenguajes de Programación

Formato	¿Qué es?	¿Dónde se usa?
ONNX	Formato abierto para modelos de IA	Python, Java, C#, Node.js, C++, Edge, Cloud
PMML	Estándar XML para ML tradicional	Sistemas empresariales, banca, Java
JPMML	Motor Java para ejecutar PMML	Aplicaciones Java/JVM
PKL	Serialización nativa de Python	Proyectos Python, APIs Python

Frameworks

Los modelo .pkl están hechos usualmente con scikit-learn, por lo que Python es el lenguaje natural para su despliegue.

Frameworks populares en Python:

- **Flask** → APIs REST simples y rápidas
- **FastAPI** → APIs modernas, muy rápidas y con validación automática
- **Django** → Despliegues más estructurados, aplicaciones web grandes
- **Streamlit** → Interfaces web para visualización o demos de IA
- **Gradio** → Interfaz web rápida para probar modelos
- **Dash** (Plotly) → Dashboards interactivos con IA



Flask

Flask



Flask es un *framework* minimalista para crear aplicaciones web en Python. Se caracteriza por ser:

- **Ligero** (no impone estructuras estrictas)
- **Flexible** (el desarrollador define la arquitectura a su manera)
- **Extensible** (permite agregar módulos como autenticación, ORM, formularios, etc.)
- **Rápido de aprender**
- Basado en el principio “**micro, pero no limitado**”

Flask permite construir desde APIs pequeñas hasta sistemas completos como portales institucionales o microservicios.

¿Qué arquitectura maneja Flask?

Arquitectura MVC (Modelo – Vista – Controlador).

- Aunque por ser un framework minimalista no obliga a usarla. Sin embargo, en la práctica es la arquitectura más usada.

Arquitectura de microservicios

- Permite crear varias APIs pequeñas, cada una ejecutándose independiente.

Arquitectura RESTful

- Crear APIs RESTEndpoints JSON
- Integración con apps de React, Angular, Vue, Flutter, etc.

Arquitectura por Blueprints

- Permite modularizar la aplicación y dividirla en módulos.

Instalación de Flask

En una ventana de cmd o powershell se debe ejecutar el siguiente comando:

```
pip install flask
```



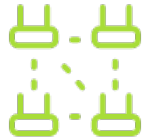
Ventajas



Desventajas



Fácil de aprender



Ideal para APIs



Flexible



Compatible con ML



Comunidad grande



Falta de componentes



Arquitectura dependiente



Configuración manual



Sin autenticación



Proyecto en Flask

Primeros Pasos - MVC

Contexto

Desarrollar una aplicación que permita a un usuario ingresar las dimensiones de un triángulo (base, altura y dos de sus lados) y obtener como resultado el área y el perímetro calculados automáticamente.

Requerimientos:

- La interfaz gráfica debe ser amigable e intuitiva.
- Aplicar buenas prácticas de organización de código.
- La aplicación web básica debe ser implementada con el framework Flask, aplicando el patrón MVC (Modelo–Vista–Controlador).
- Aplicar los conceptos fundamentales del desarrollo web backend.

Desarrollo de la aplicación

Paso 01: Crear el entorno virtual (opcional)

```
python -m venv env
```

Paso 02: Activar el entorno virtual (opcional)

```
env\Scripts\Activate
```

Paso 03: Instalar Flask

```
pip install flask
```


Desarrollo de la aplicación

Estructura del proyecto - MVC

```
flask_mvc/  
├── app.py                # Controlador / rutas  
  
├── models/  
│   └── triangulo.py     # Modelo (lógica de negocio)  
  
├── templates/  
│   └── triangulo.html   # Vista (HTML con Jinja2 + Bootstrap)  
  
└── static/  
    ├── css/  
    │   └── estilos.css  # Estilos personalizados - otros
```

Desarrollo de la aplicación

Paso 04: Crear el controlador y las rutas del proyecto **app.py**

```
# Importaciones necesarias
from flask import Flask
from models.triangulo import calcular_triangulo

# Inicialización de la aplicación Flask
app = Flask(__name__)

# Ruta principal que maneja el formulario y muestra resultados
@app.route('/', methods=['GET', 'POST'])
def formulario_triangulo():
    base, altura, lado1, lado2 = 5, 10, 7, 8
    area, perimetro = calcular_triangulo(base, altura, lado1, lado2)
    return f"Área: {area}, Perímetro: {perimetro}"

# Ejecución de la aplicación Flask
if __name__ == '__main__':
    app.run(debug=True)
```

Desarrollo de la aplicación

Paso 05: Crear los modelos – lógica del negocio

```
# Funciones para calcular el área de un triángulo
def calcular_area(base, altura):
    return (base * altura) / 2

# Funciones para calcular el perímetro de un triángulo
def calcular_perimetro(base, lado1, lado2):
    return base + lado1 + lado2

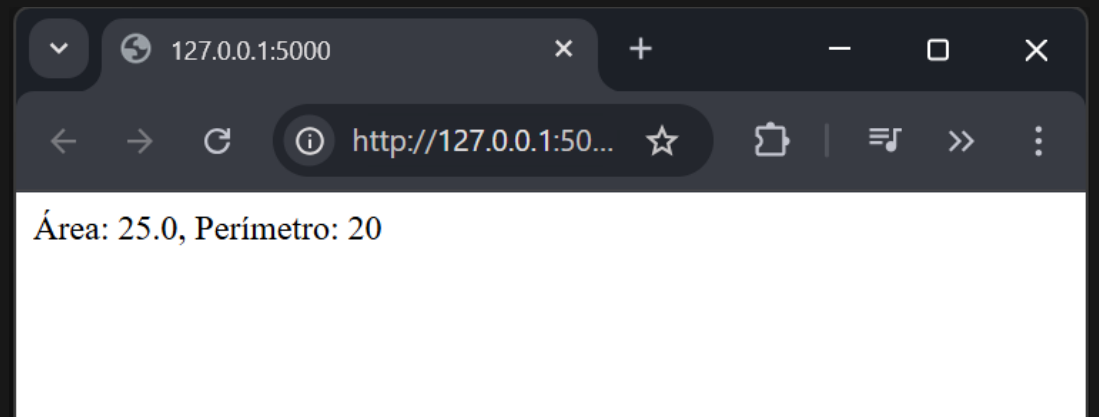
# Función de alto nivel que utiliza las funciones anteriores
def calcular_triángulo(base, altura, lado1, lado2):
    area = calcular_area(base, altura)
    perimetro = calcular_perimetro(base, lado1, lado2)
    return area, perimetro
```

Desarrollo de la aplicación

Paso 06: Ejecución de *prueba* de la aplicación

```
python app.py
```

```
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 901-256-227
```



Desarrollo de la aplicación

Paso 07: Construir las vistas de la aplicación.

Cálculo de Área y Perímetro de un Triángulo

Base: *

Altura: *

Lado 1: *

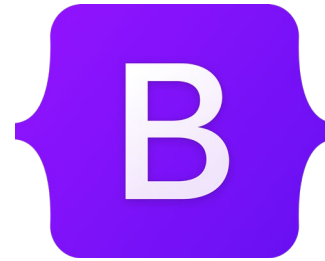
Lado 2: *

Calcular



Desarrollo de la aplicación

Paso 08: Construir los archivos estáticos (css – js - archivos).



Paso 09: Ejecución *final* de la aplicación

```
python app.py
```

Resultados

Base: 34.0

Altura: 89.0

Lado 1: 45.0

Lado 2: 43.0

Área: 1513.0

Perímetro: 122.0



Proyecto en Flask

Primeros Pasos - Rest

Contexto

Desarrollar una aplicación que permita a un usuario ingresar las dimensiones de un triángulo (base, altura y dos de sus lados) y obtener como resultado el área y el perímetro calculados automáticamente.

Por lo anterior, un cliente (Postman, frontend en React, otra app, etc.) enviará los datos del triángulo y Flask responderá con el área y el perímetro.

Desarrollo de la aplicación

Paso 01: Crear el entorno virtual (opcional)

```
python -m venv env
```

Paso 02: Activar el entorno virtual (opcional)

```
env\Scripts\Activate
```

Paso 03: Instalar Flask

```
pip install flask
```

Desarrollo de la aplicación

Estructura del proyecto - Rest

```
flask_rest/  
├── app.py           # API REST (rutas)  
└── models/  
    ├── triangulo.py # Modelo (lógica de negocio)
```


Desarrollo de la aplicación

Paso 04: Crear los modelos – lógica del negocio

```
# Funciones para calcular el área de un triángulo
def calcular_area(base, altura):
    return (base * altura) / 2

# Funciones para calcular el perímetro de un triángulo
def calcular_perimetro(base, lado1, lado2):
    return base + lado1 + lado2

# Función de alto nivel que utiliza las funciones anteriores
def calcular_triangulo(base, altura, lado1, lado2):
    area = calcular_area(base, altura)
    perimetro = calcular_perimetro(base, lado1, lado2)
    return area, perimetro
```

Paso 05: Crear las rutas de la aplicación



```
from flask import Flask, request, jsonify
from models.triangulo import calcular_triangulo
# Inicialización de la aplicación Flask
app = Flask(__name__)
# Ruta que maneja el endpoint REST para calcular área y perímetro
@app.route("/api/triangulo", methods=["POST"])
def calcular_triangulo_endpoint():
    # 1. Obtener datos del request
    base, altura, lado1, lado2 = 5, 10, 7, 8
    # 2. Llamar al modelo
    area, perimetro = calcular_triangulo(base, altura, lado1, lado2)
    # 3. Responder en formato JSON
    respuesta = {
        "base": base, "altura": altura,
        "lado1": lado1, "lado2": lado2,
        "area": area, "perimetro": perimetro
    }
    # 4. Devolver la respuesta
    return jsonify(respuesta), 200
# Ejecución de la aplicación Flask
if __name__ == "__main__":
    app.run(debug=True)
```

Desarrollo de la aplicación

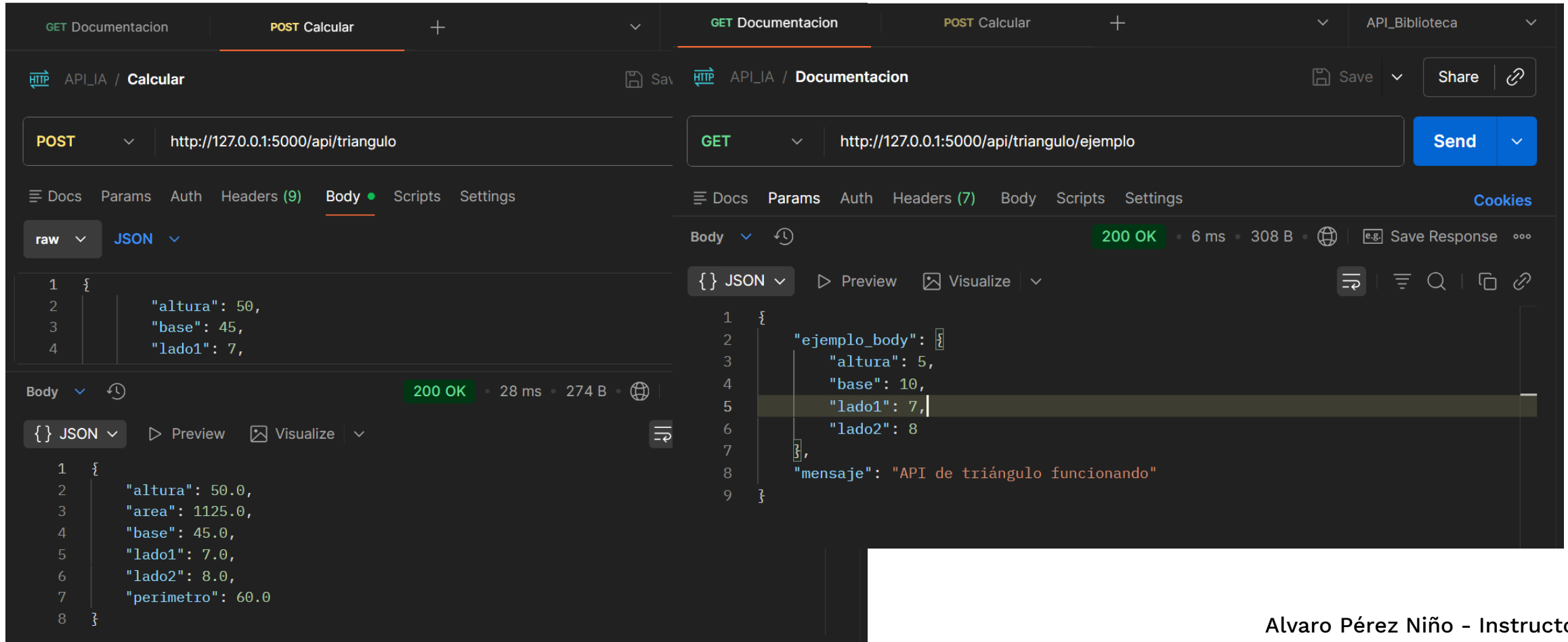
Paso 06: Ejecutar la API - Server

```
python app.py
```

```
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 901-256-227
```

Desarrollo de la aplicación

Paso 07: Ejecutar la API - Cliente



The image displays two side-by-side screenshots of an API client interface, likely Postman, showing the execution of two different API requests.

Left Screenshot (POST Request):

- Method:** POST
- URL:** `http://127.0.0.1:5000/api/triangulo`
- Body (JSON):**

```
{  "altura": 50,  "base": 45,  "lado1": 7,
```
- Response (200 OK):**

```
{  "altura": 50.0,  "area": 1125.0,  "base": 45.0,  "lado1": 7.0,  "lado2": 8.0,  "perimetro": 60.0}
```

Right Screenshot (GET Request):

- Method:** GET
- URL:** `http://127.0.0.1:5000/api/triangulo/ejemplo`
- Body (JSON):**

```
{  "ejemplo_body": {    "altura": 5,    "base": 10,    "lado1": 7,    "lado2": 8  },  "mensaje": "API de triángulo funcionando"}
```



Alojamiento Cloud

Despliegue en Render

<https://render.com>



Ventajas

- Soporta bien Flask, FastAPI, Django.
- Despliegue automático desde GitHub.
- Dominios gratuitos
- Permite ejecutar APIs REST fácilmente.
- Reinicio automático cuando hay cambios en GitHub.

Desventajas

- El plan free duerme después de 15 minutos sin tráfico.
- Puede tardar 20–30 segundos en “despertar”.

Despliegue de la aplicación

Paso 01: Preparar el archivo principal (app.py).

```
if __name__ == "__main__":  
    # Desarrollo  
    app.run(debug=True)
```



```
# Producción  
app.run()
```

Paso 02: Crear archivo requirements.txt


```
pip freeze > requirements.txt
```

```
Flask==3.1.2  
gunicorn==21.2.0
```

Gunicorn es un servidor WSGI que ejecuta aplicaciones Python (como Flask o Django) en producción, permitiendo manejar múltiples peticiones de forma eficiente y estable.

Despliegue de la aplicación

Paso 03: Subir el proyecto a GitHub.


 **trianguloMVC** Public

main


1 Branch

0 Tags

Go to

 **aperezn298** Add gunicorn to requirements

models	MVC
static/css	MVC
templates	MVC
app.py	MVC
requirements.txt	Add gunicorn to requirements


 **trianguloRest** Public

main

1 Branch

0 Tags

Go to file

 **aperezn298** Add gunicorn to requirements

models	Triangulo_Rest
app.py	Triangulo_Rest
requirements.txt	Add gunicorn to requirements

Despliegue de la aplicación

Paso 03: Subir el proyecto a GitHub.

- Crea un repositorio en GitHub: *flask-triangulo-api*
- En el PC, desde la carpeta del proyecto:

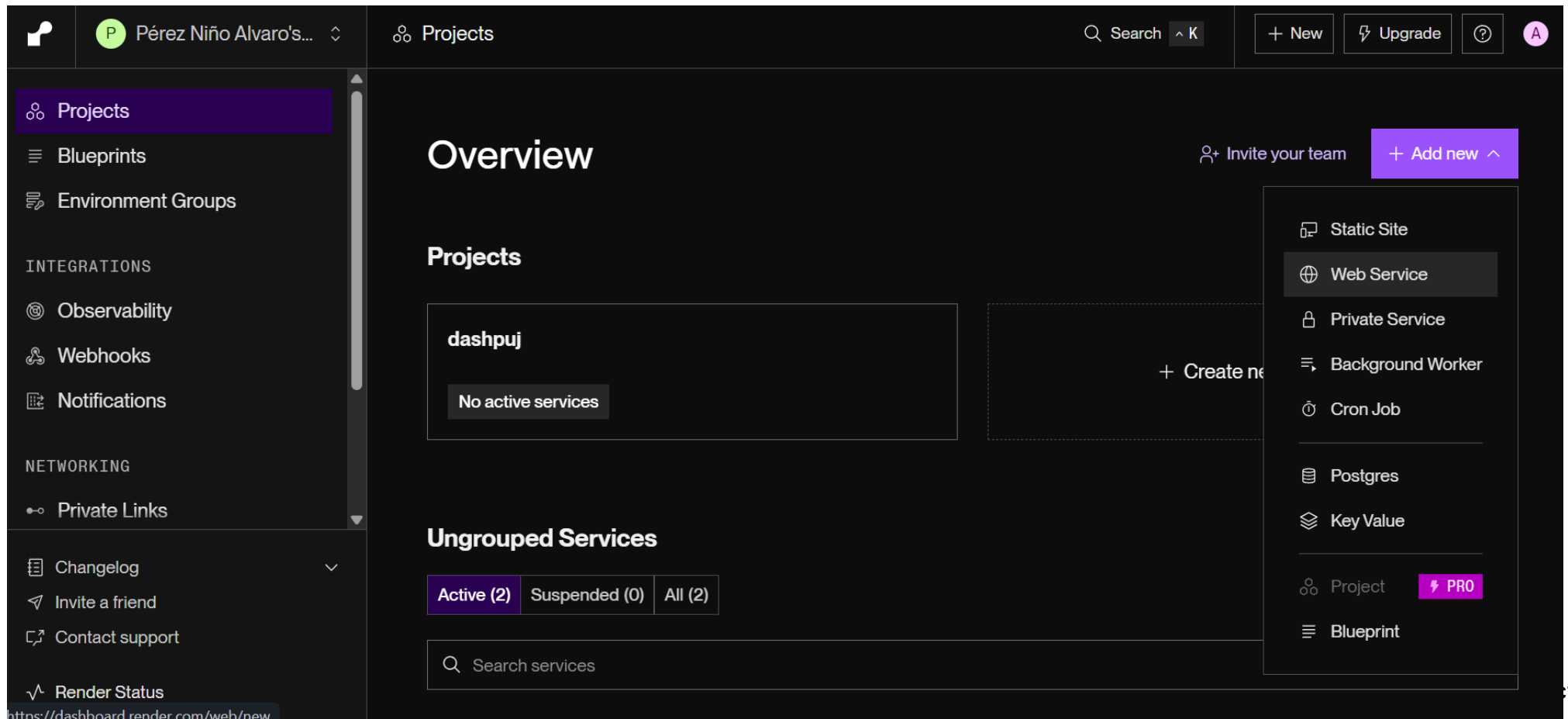


```
git init
git add .
git commit -m "Versión inicial API Triángulo"
git branch -M main
git remote add origin https://github.com/USUARIO_GIT/flask-triangulo-api.git
git push -u origin main
```



Despliegue de la aplicación

Paso 04: Crear una cuenta en Render y crear un nuevo Web Service.



Despliegue de la aplicación

Paso 05: Seleccionar el repositorio a desplegar.

New Web Service

It looks like you're using **Python**, so we've autofilled some fields accordingly.

Source Code

Git Provider

Public Git Repository

Existing Image

Credentials (1) ▾

aperezn298 / trianguloMVC 1h ago

aperezn298 / trianguloRest 1h ago

aperezn298 / CienciaDatosSENA 10h ago [View repo](#)

aperezn298 / Flask_Hepatitis 4d ago

aperezn298 / FundBDSENA Sep 24

MHGeronimo / Frontend_SteticSoft Sep 16

aperezn298 / ImplantacionSENA Aug 20

Despliegue de la aplicación

Paso 06: Configurar el render y Deplegar

Build Command
Render runs this command to build your app before each deploy.

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

Start Command
Render runs this command to start your app with each deploy.

```
$ gunicorn app:app
```


Instance Type

For hobby projects

Free
\$0 / month

512 MB (RAM)
0.1 CPU

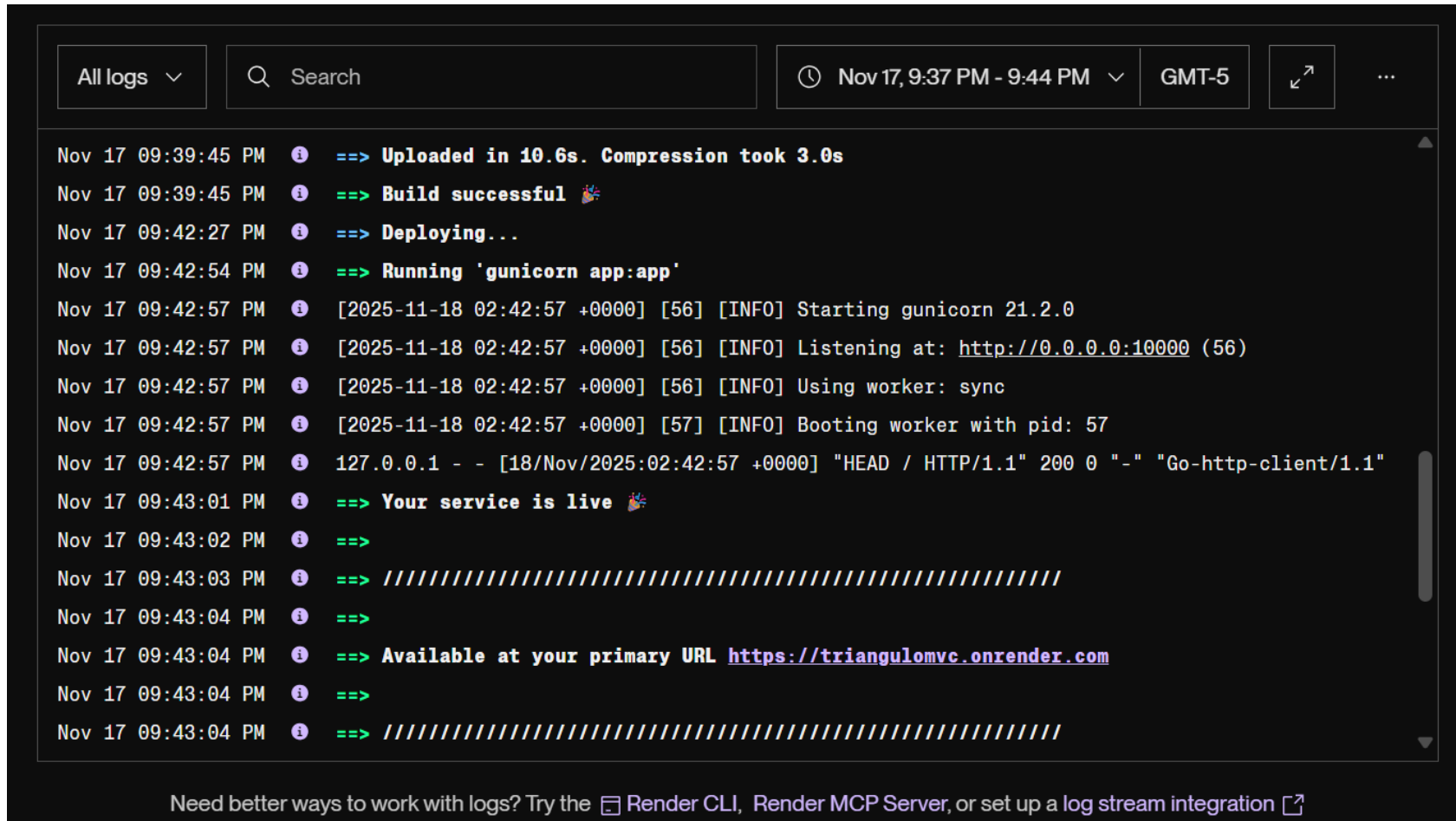
Upgrade to enable more features
Free instances spin down after periods of inactivity. They do not support SSH access, scaling, one-off jobs, or persistent disks. Select any paid instance type to enable these features.

 Deploy Web Service

Free
\$0 / month

Despliegue de la aplicación

Paso 07: Construcción del despliegue en el servidor.



```
Nov 17 09:39:45 PM ⓘ ==> Uploaded in 10.6s. Compression took 3.0s
Nov 17 09:39:45 PM ⓘ ==> Build successful 🎉
Nov 17 09:42:27 PM ⓘ ==> Deploying...
Nov 17 09:42:54 PM ⓘ ==> Running 'gunicorn app:app'
Nov 17 09:42:57 PM ⓘ [2025-11-18 02:42:57 +0000] [56] [INFO] Starting gunicorn 21.2.0
Nov 17 09:42:57 PM ⓘ [2025-11-18 02:42:57 +0000] [56] [INFO] Listening at: http://0.0.0.0:10000 (56)
Nov 17 09:42:57 PM ⓘ [2025-11-18 02:42:57 +0000] [56] [INFO] Using worker: sync
Nov 17 09:42:57 PM ⓘ [2025-11-18 02:42:57 +0000] [57] [INFO] Booting worker with pid: 57
Nov 17 09:42:57 PM ⓘ 127.0.0.1 - - [18/Nov/2025:02:42:57 +0000] "HEAD / HTTP/1.1" 200 0 "-" "Go-http-client/1.1"
Nov 17 09:43:01 PM ⓘ ==> Your service is live 🎉
Nov 17 09:43:02 PM ⓘ ==>
Nov 17 09:43:03 PM ⓘ ==> //////////////////////////////////////
Nov 17 09:43:04 PM ⓘ ==>
Nov 17 09:43:04 PM ⓘ ==> Available at your primary URL https://triangulomvc.onrender.com
Nov 17 09:43:04 PM ⓘ ==>
Nov 17 09:43:04 PM ⓘ ==> //////////////////////////////////////
```

Need better ways to work with logs? Try the [Render CLI](#), [Render MCP Server](#), or set up a log stream integration [↗](#)

Despliegue de la aplicación

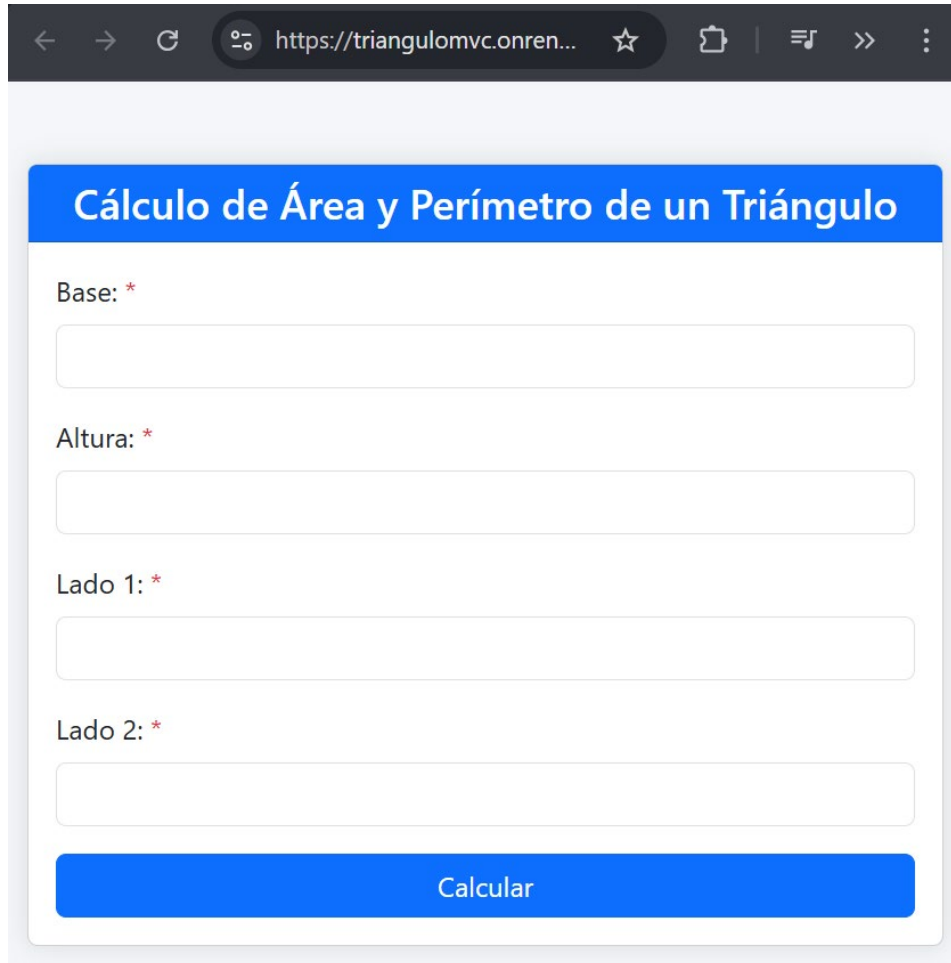
Paso 08: Acceder a la aplicación.

```
22:41:47 INCOMING HTTP REQUEST DETECTED ...  
22:41:50 SERVICE WAKING UP ...  
  
[Progress bar showing 100% completion]  
  
22:41:54 ALLOCATING COMPUTE RESOURCES ...  
22:41:57 PREPARING INSTANCE FOR INITIALIZATION ...  
22:42:01 STARTING THE INSTANCE ...  
22:42:07 ENVIRONMENT VARIABLES INJECTED ...  
22:42:09 FINALIZING STARTUP ...  
22:42:11 OPTIMIZING DEPLOYMENT ...  
22:42:13 STEADY HANDS. CLEAN LOGS. YOUR APP IS ALMOST LIVE ...  
START BUILDING ON RENDER TODAY →
```

○ APPLICATION LOADING

Despliegue de la aplicación

Paso 08: Acceder a la aplicación.



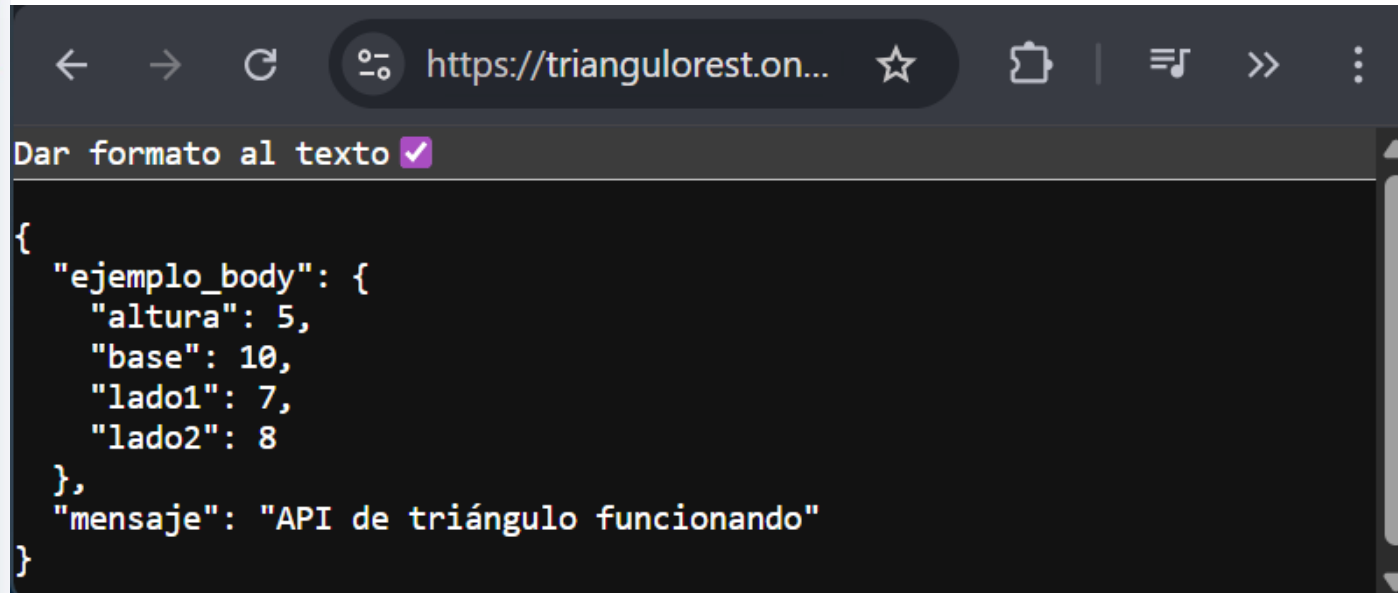
Base: *

Altura: *

Lado 1: *

Lado 2: *

Calcular



```
{
  "ejemplo_body": {
    "altura": 5,
    "base": 10,
    "lado1": 7,
    "lado2": 8
  },
  "mensaje": "API de triángulo funcionando"
}
```



GRACIAS

Presentó: Alvaro Pérez Niño
Instructor Técnico

Correo: aperezn@misena.edu.co

<http://centrodeserviciosygestionempresarial.blogspot.com/>

Línea de atención al ciudadano: 01 8000 910270

Línea de atención al empresario: 01 8000 910682



www.sena.edu.co