05 – Despliegue

Centro de Servicios y Gestión Empresarial
SENA Regional Antioquia

@SENAComunica

www.sena.edu.co

Despliegue

Alvaro Pérez Niño - Instructor

Despliegue

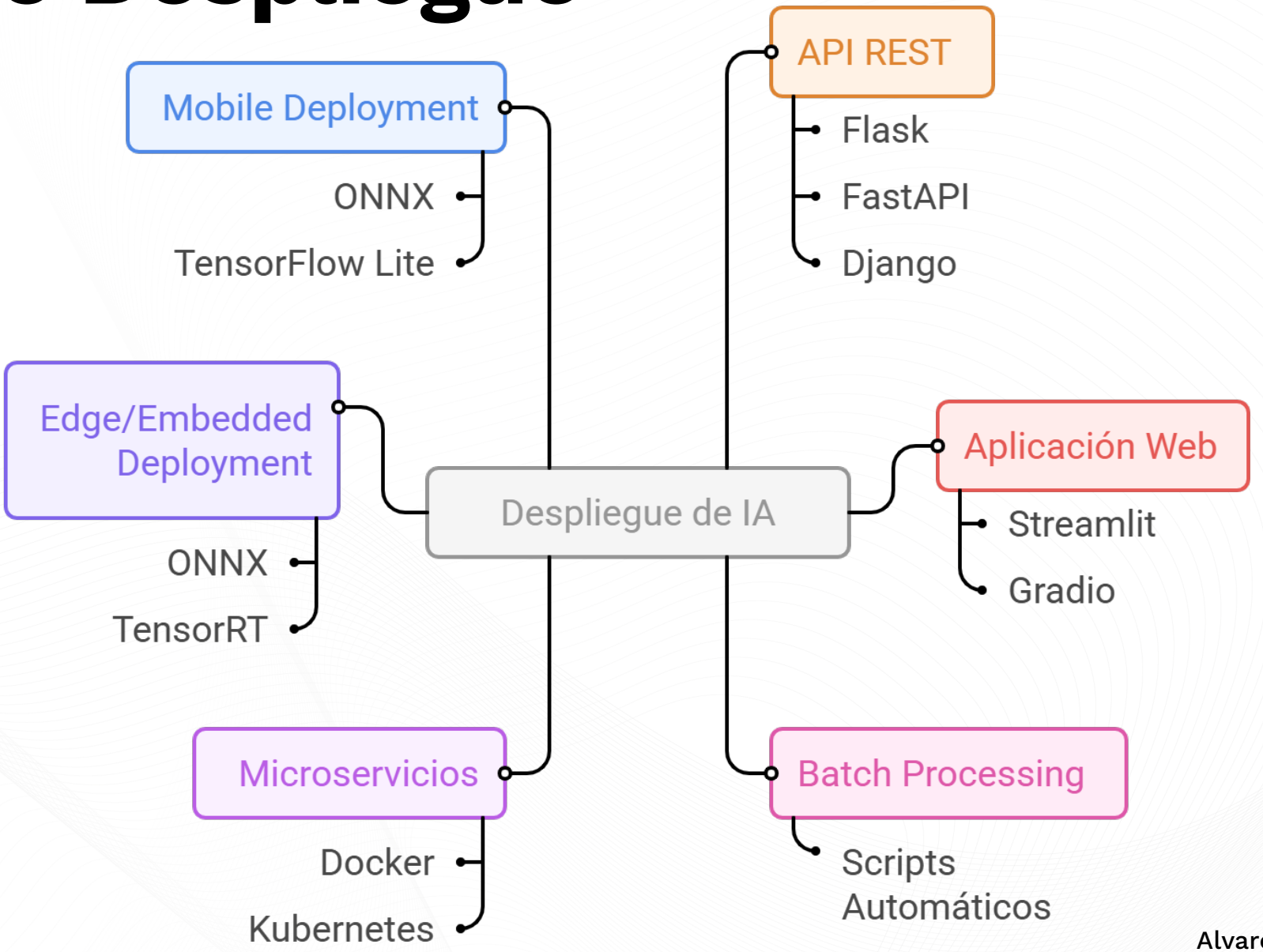
En esta fase el modelo se implementa en el entorno real donde será utilizado. Se integran los resultados a los procesos de la organización y se establecen mecanismos de monitoreo, mantenimiento y actualización para asegurar su funcionamiento continuo.

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

Elementos de Ingeniería de Software

Alvaro Pérez Niño - Instructor

Tipos de Despliegue



Alvaro Pérez Niño - Instructor

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**

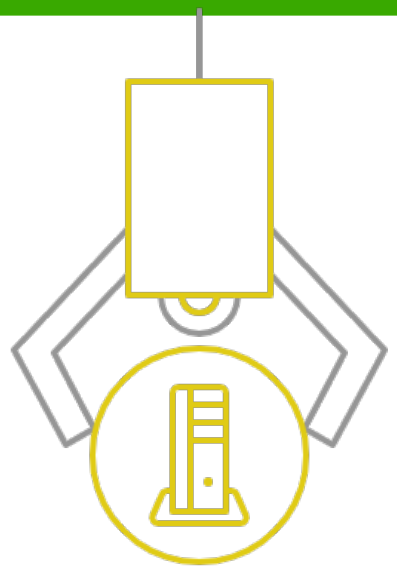
Rutas: Recibir solicitudes y entregarlas a servicios

Servicios: Validación, cálculos, reglas de negocio

Repositorios: Interactuar con la base de datos o IA

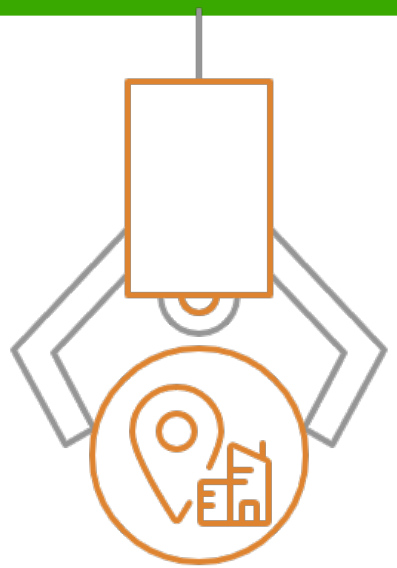
Alvaro Pérez Niño - Instructor

Infraestructura necesaria



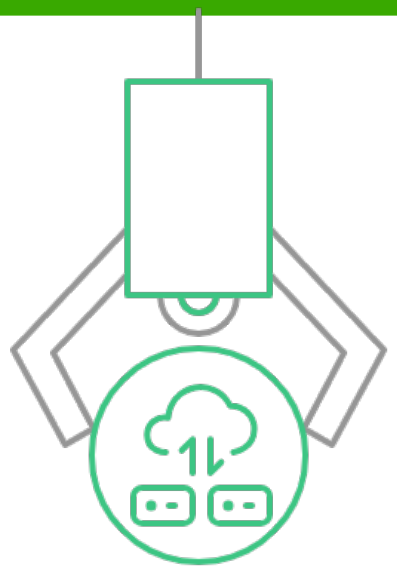
Servidor local

Alojamiento de modelos en un servidor local.



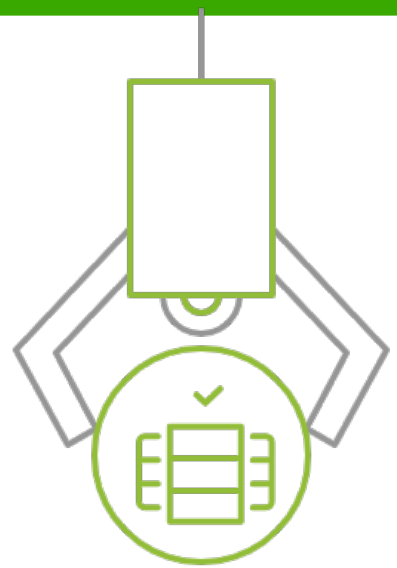
On-Premise

Alojamiento de modelos en la infraestructura propia.



Cloud

Alojamiento de modelos en la nube.



CI/CD pipelines

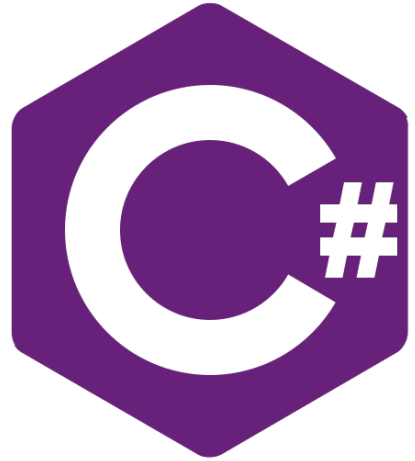
Uso de pipelines CI/CD para actualizaciones automáticas.

Alvaro Pérez Niño - Instructor

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.



Alvaro Pérez Niño - Instructor

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

Alvaro Pérez Niño - Instructor

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

- Fácil de aprender
- Ideal para APIs
- Flexible
- Compatible con ML
- Comunidad grande

VS

Desventajas

- Falta de componentes
- Arquitectura dependiente
- Configuración manual
- Sin autenticación

Alvaro Pérez Niño - Instructor

Proyecto en Flask

Primeros Pasos - MVC

Alvaro Pérez Niño - Instructor

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**

```python
# 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

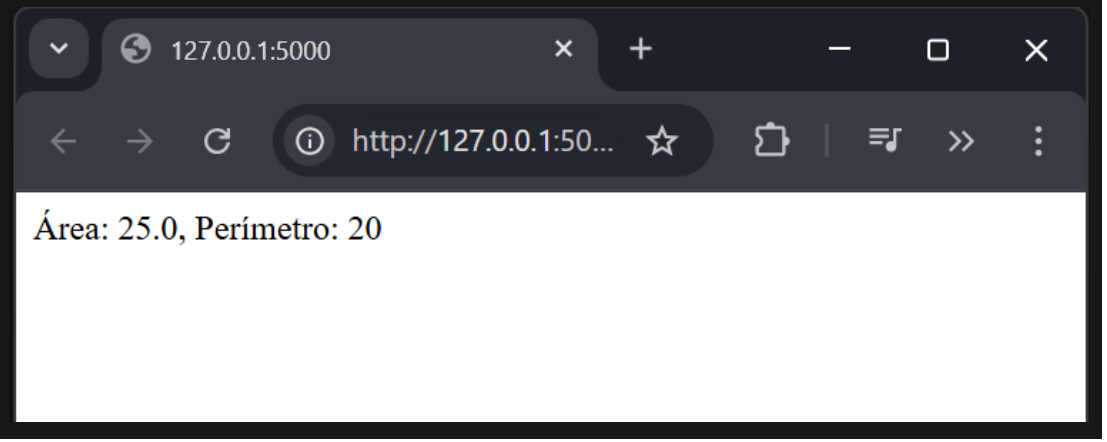
Paso 05: Crear los modelos – lógica del negocio

```python
# 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
```

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
```



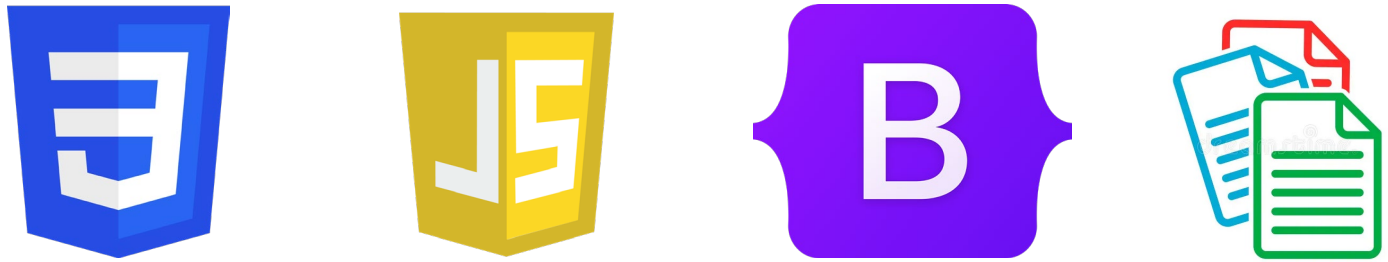
Desarrollo de la aplicación

Paso 07: Construir las vistas de la aplicación.

Alvaro Pérez Niño - Instructor

Desarrollo de la aplicación

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



Paso 09: Ejecución *final* de la aplicación

```
python app.py
```

Proyecto en Flask

Primeros Pasos - Rest

Alvaro Pérez Niño - Instructor

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

```python
# 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

```python
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)
```

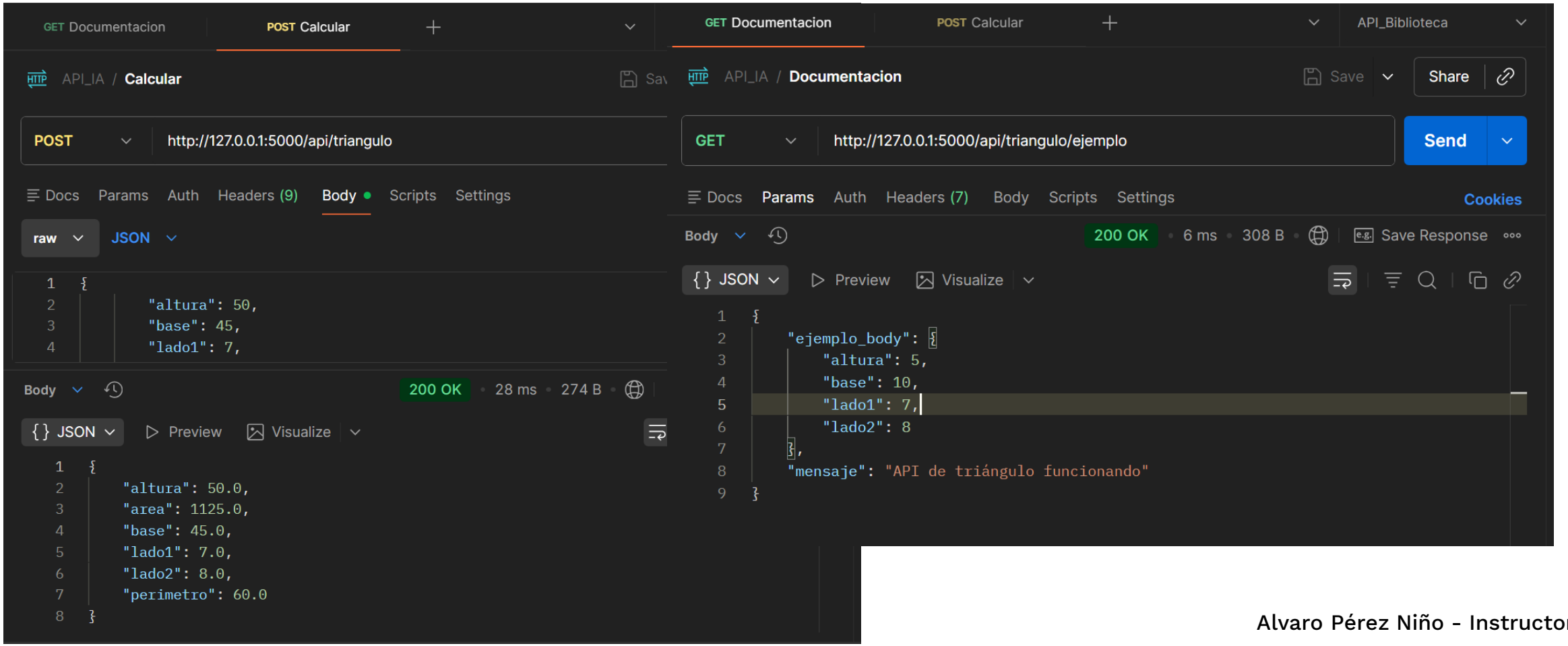
Desarrollo de la aplicación

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



Alvaro Pérez Niño - Instructor

Alojamiento Cloud

Alvaro Pérez Niño - Instructor

Despliegue en Render

https://render.com

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

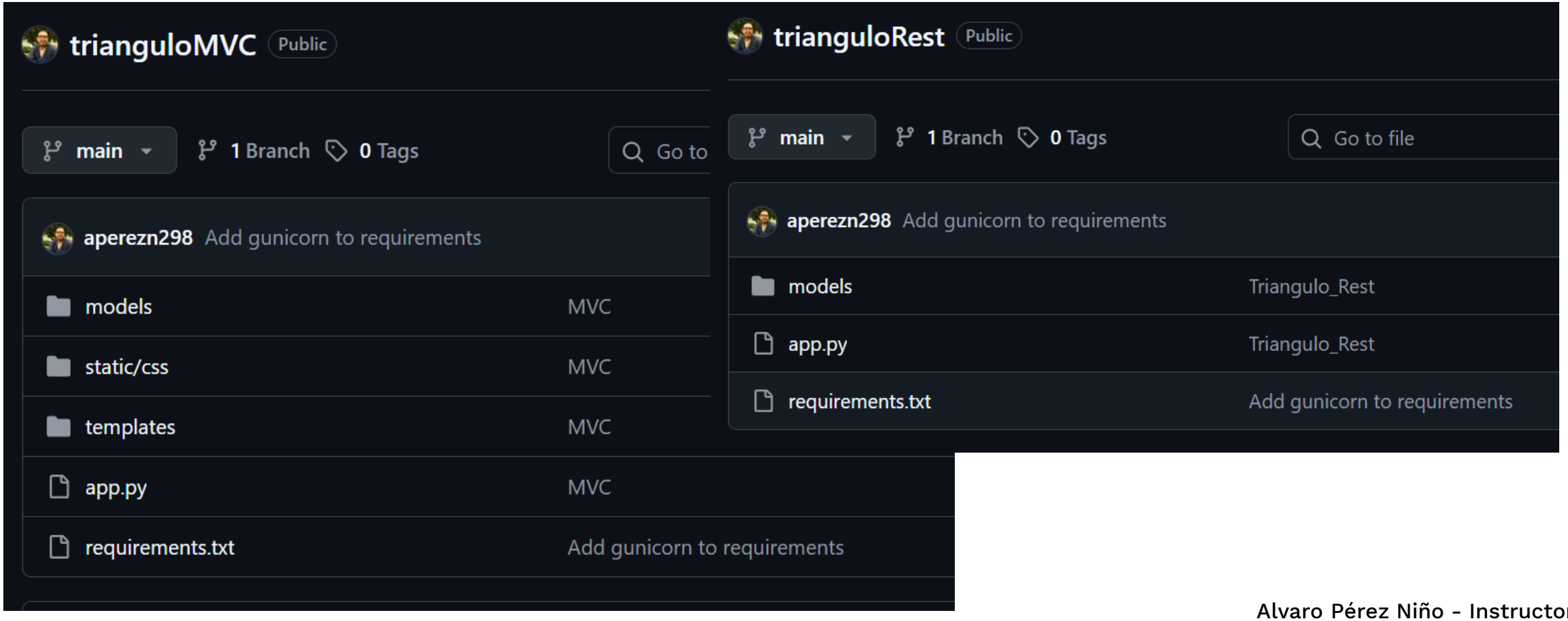
```
pip freeze > requirements.txt
```

```
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.



Despliegue de la aplicación

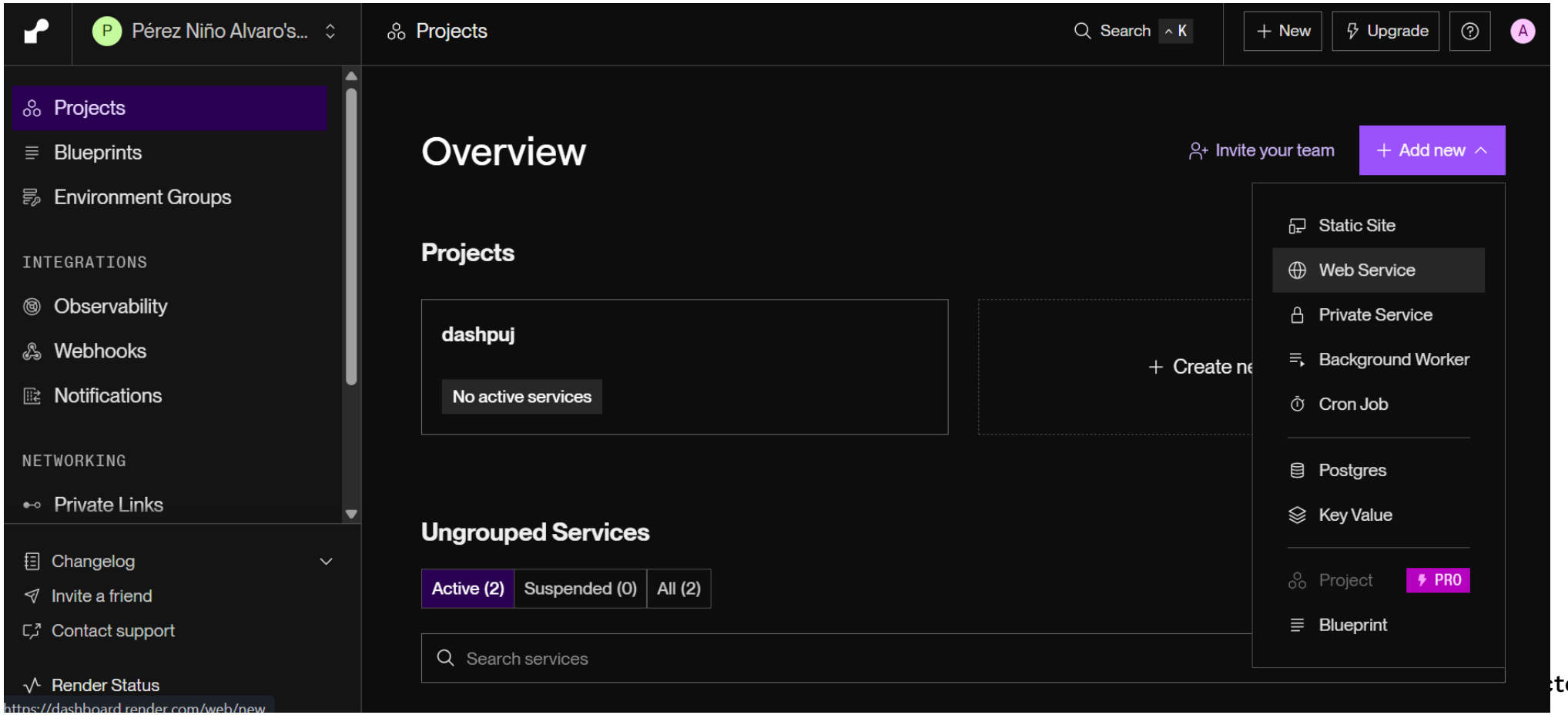
Paso 03: Subir el proyecto a GitHub.

- 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
```

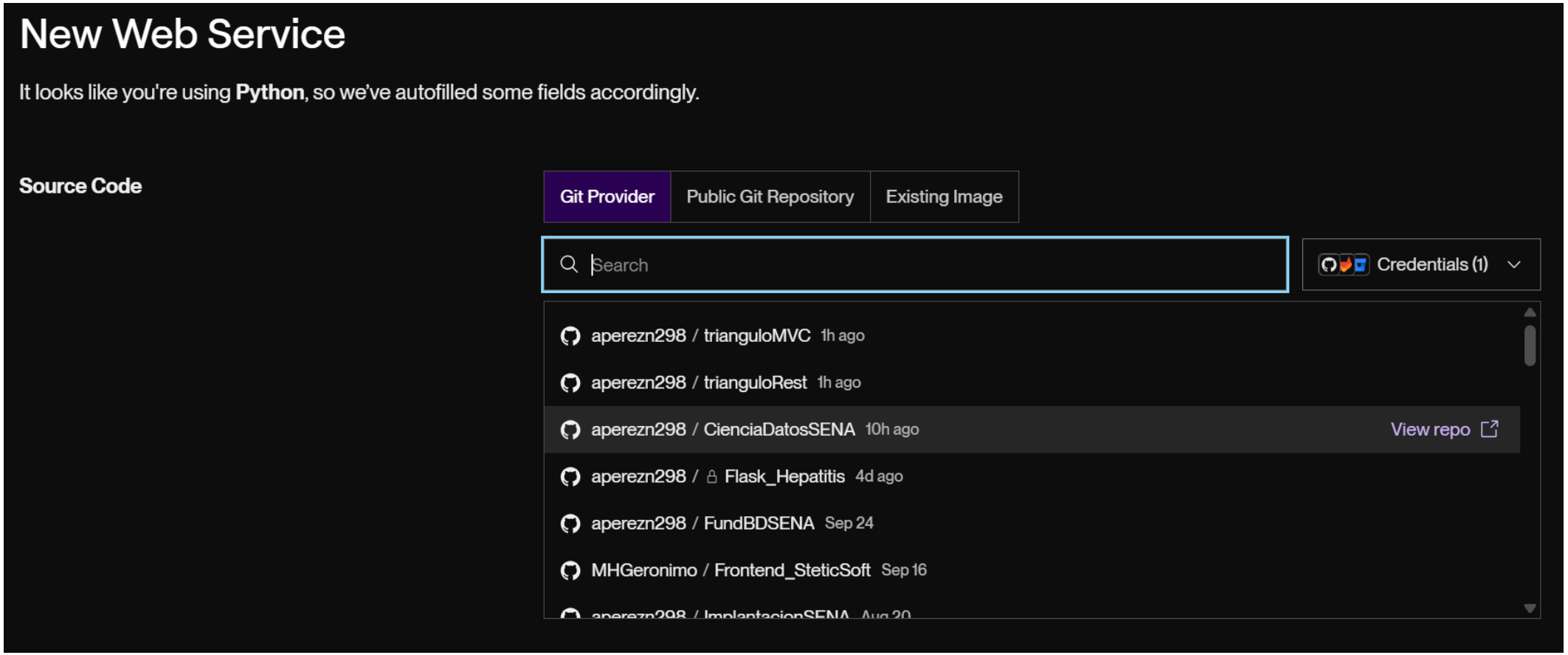
Despliegue de la aplicación

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



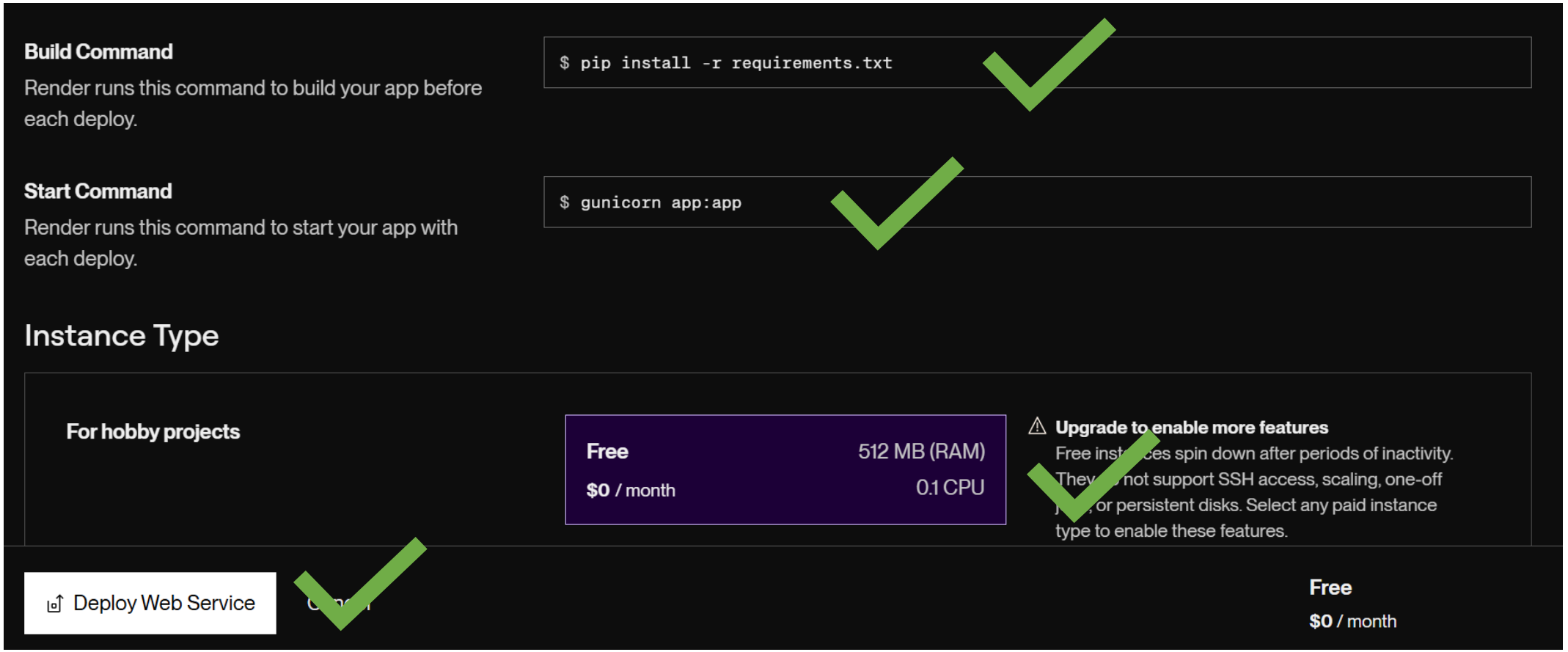
Despliegue de la aplicación

Paso 05: Seleccionar el repositorio a desplegar.



Despliegue de la aplicación

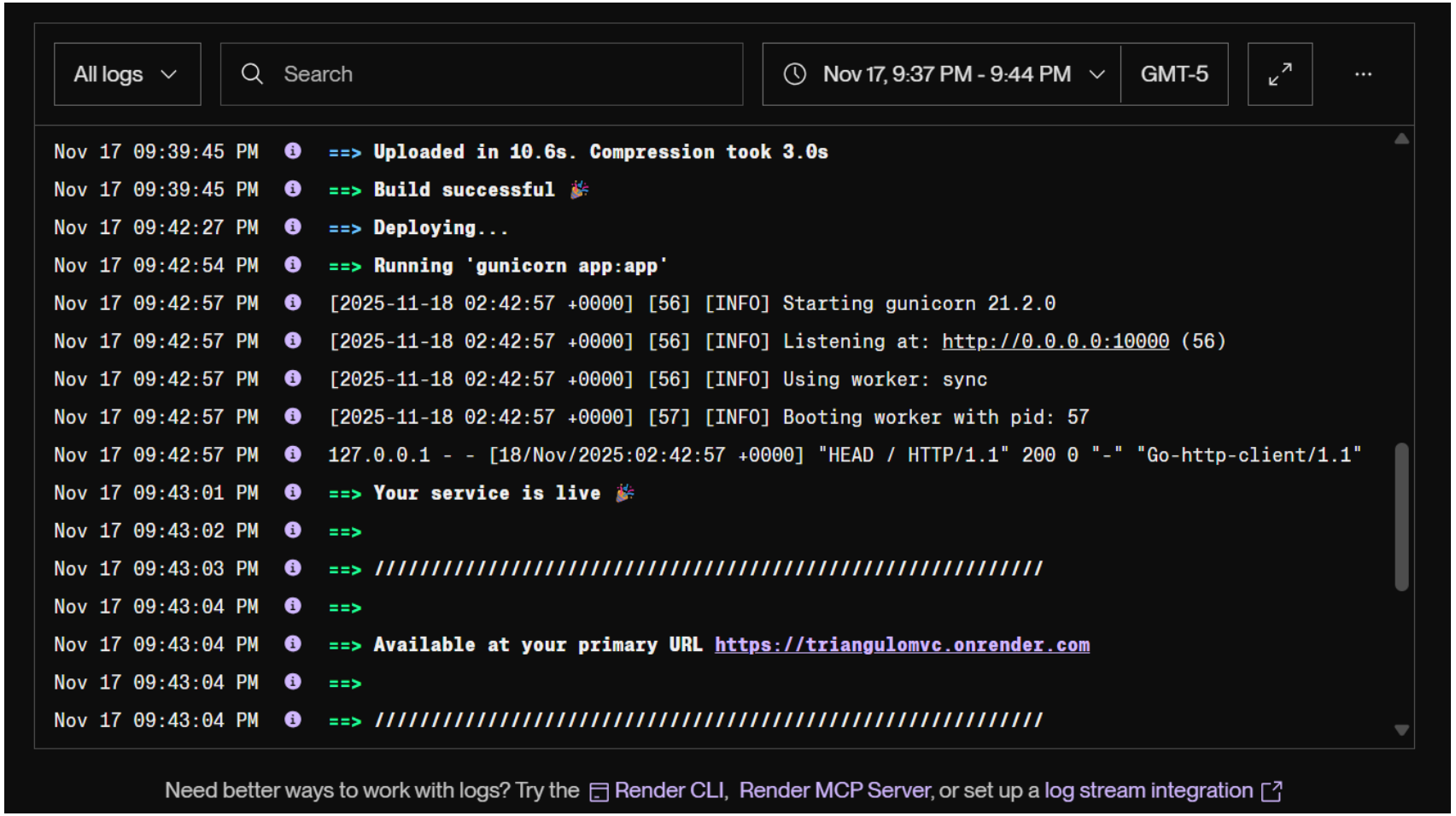
Paso 06: Configurar el render y Deplegar



Alvaro Pérez Niño - Instructor

Despliegue de la aplicación

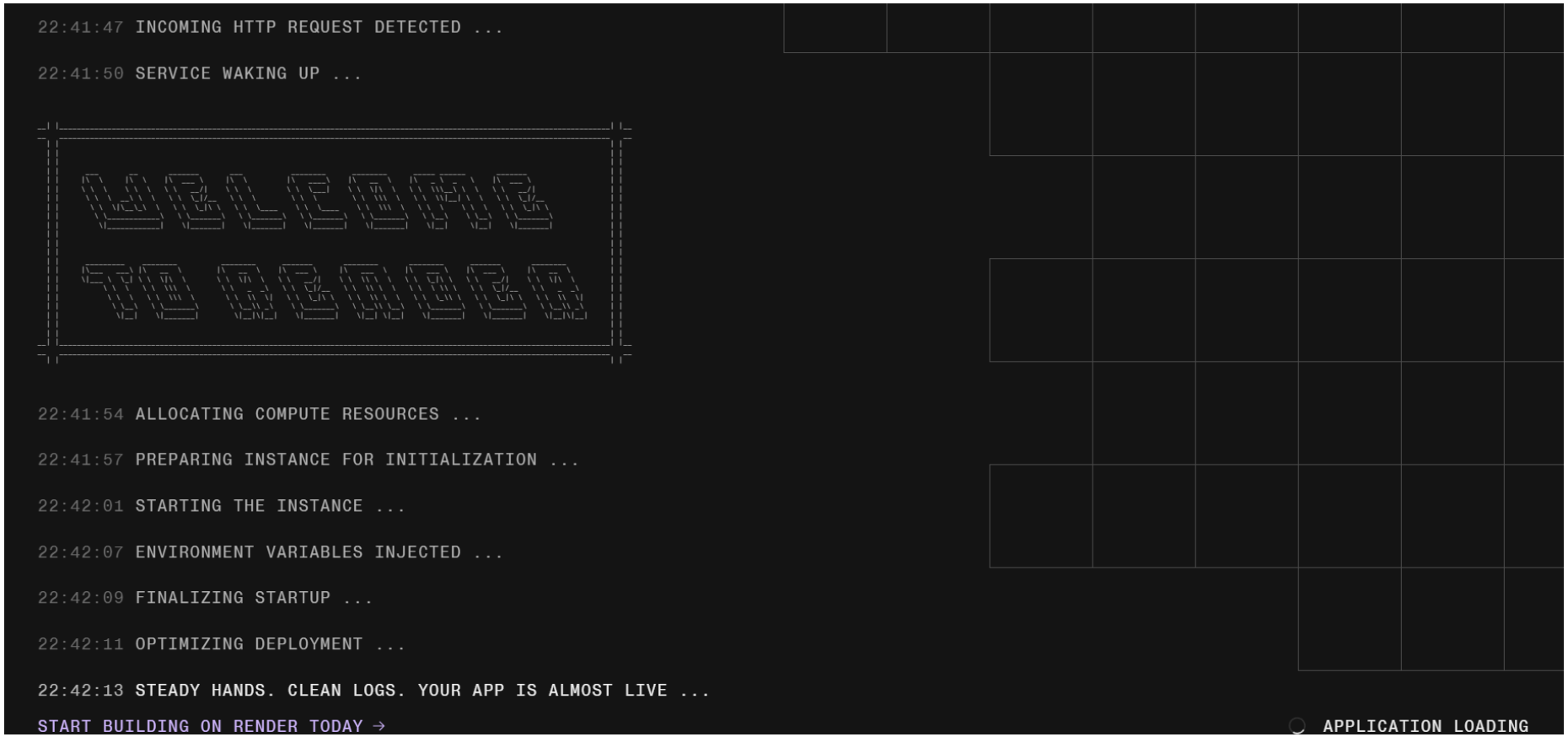
Paso 07: Construcción del despliegue en el servidor.



Alvaro Pérez Niño - Instructor

Despliegue de la aplicación

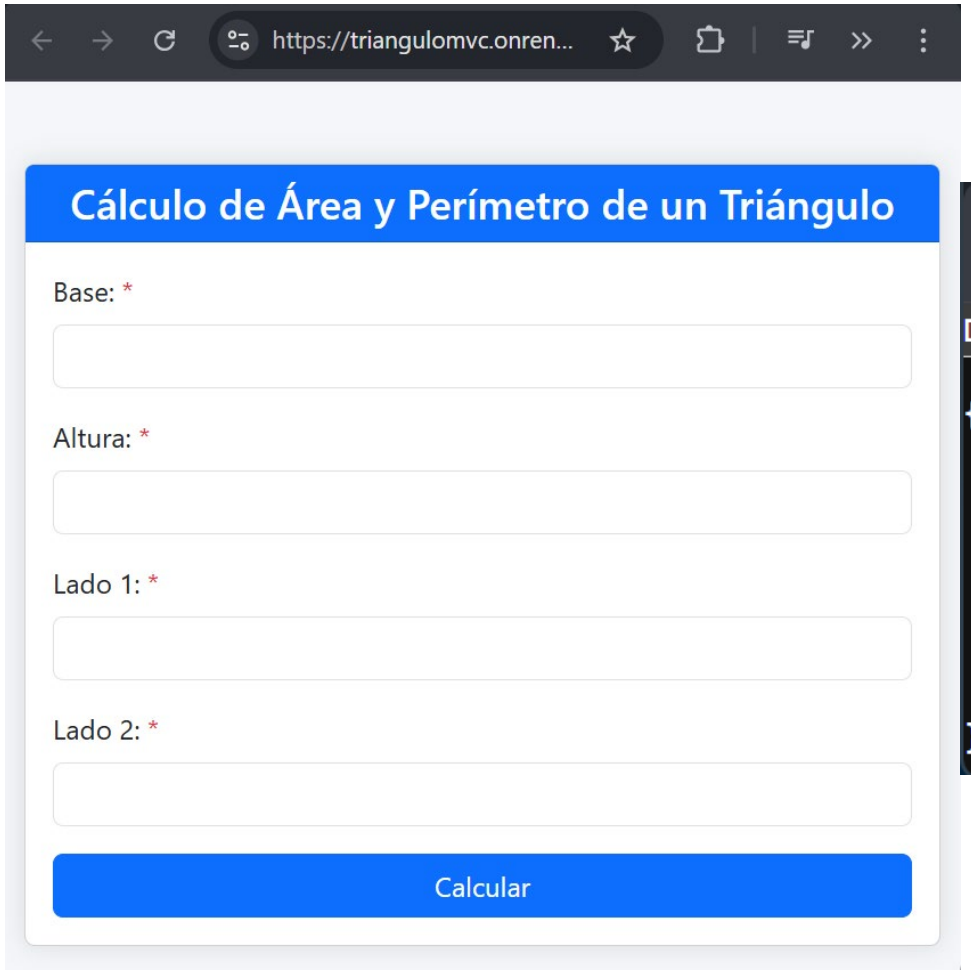
Paso 08: Acceder a la aplicación.



Alvaro Pérez Niño - Instructor

Despliegue de la aplicación

Paso 08: Acceder a la aplicación.



G R A C I A S

Presentó: Alvaro Pérez Niño
Instructor Técnico
Correo: aperezn@misena.edu.co
http://centrodeserviciosygestionempresarial.blogspot.com/