**DEFINICIÓN DE PROYECTO: PLATAFORMA "PRE-INCUBADORA AI"**

Desarrollado por:

**Nicolás Navarro Garrido**

[**1.0 Resumen Ejecutivo 3**](#)

[1.1 Objetivos Estratégicos 3](#)

[**2.0 Alcance Funcional y Lógica de Negocio 4**](#)

[2.1 Flujo de Valor (The "Hook") 4](#)

[**3.0 Arquitectura Técnica (Monolito Self-Hosted) 5**](#)

[3.1 Stack Tecnológico Detallado 5](#)

[**4.0 Estrategia de Inteligencia Artificial 6**](#)

[4.1 Selección del Modelo: Gemini 1.5 Flash (Google) 6](#)

[4.2 Análisis de Costos (IA) 6](#)

[**5.0 Análisis Financiero (OpEx) 7**](#)

[5.1 Estructura de Costos Mensuales (Proyección) 7](#)

[**6.0 Plan de Implementación (Roadmap) 8**](#)

[Fase 1: Core Monolítico (Semana 1) 8](#)

[Fase 2: Lógica de Negocio & IA (Semana 2) 8](#)

[Fase 3: Chat & Interacción (Semana 2-3) 8](#)

[Fase 4: Despliegue (Semana 4) 8](#)

[**7.0 Análisis de Riesgos y Mitigación 9**](#)

[**8.0 Conclusión 9**](#)

## 1.0 Resumen Ejecutivo

El proyecto consiste en el desarrollo de una plataforma SaaS (*Software as a Service*) propietaria diseñada como un **Embudo de Conversión Automatizado (Lead Magnet)**. Utilizando Inteligencia Artificial Generativa, la plataforma transforma ideas de negocio abstractas en esquemas empresariales estructurados, ofreciendo valor inmediato al usuario final a cambio de calificación comercial.

### 1.1 Objetivos Estratégicos

1. **Captación de Leads Cualificados:** Automatizar la primera fase de consultoría para filtrar curiosos de clientes potenciales reales.
2. **Soberanía Tecnológica:** Implementar una arquitectura 100% autogestionada (*Self-Hosted*) para garantizar el control total de los datos.
3. **Velocidad de Desarrollo (Rapid Prototyping):** Utilizar una arquitectura **Monolítica (Full-Stack Flask)** para eliminar la complejidad de separar Frontend/Backend, acelerando el *Time-to-Market*.

## 2.0 Alcance Funcional y Lógica de Negocio

### 2.1 Flujo de Valor (The "Hook")

El sistema opera bajo un modelo *Freemium* restrictivo diseñado psicológicamente para la conversión:

1. **Input Inicial:** El usuario se registra (Email/Password) e ingresa su idea en lenguaje natural.
2. **Refinamiento Activo (Ambiguity Check):**
   * El sistema analiza la densidad de información del input.
   * **Condicional:** Si la idea es vaga, la IA interrumpe y lanza una micro-entrevista (máximo 3 preguntas) vía chat para extraer contexto crítico.
3. **Procesamiento (The Blueprint):** Una vez validado el contexto, el sistema genera el reporte de viabilidad basado en 5 tópicos inmutables:
   * **Problema Real y Propuesta de Valor.**
   * **Mercado y Modelo de Ingresos.**
   * **Costos y Recursos.**
   * **Viabilidad y Riesgos.**
   * **Roadmap para Iniciar.**
4. **Consultoría Limitada:** Se habilita un chat contextual con la IA sobre el reporte generado.
   * *Restricción:* Límite duro (*Hard Cap*) de **10 mensajes**.
5. **Cierre (Call to Action):** Al agotar los créditos o detectar intención de compra, el sistema bloquea el input y presenta la única salida: **"Agendar Reunión Estratégica"** (Link Google Calendar).

## 3.0 Arquitectura Técnica (Monolito Self-Hosted)

Se ha optimizado la arquitectura unificando Frontend y Backend en una sola aplicación Flask (Server Side Rendering). Esto reduce la complejidad de mantenimiento y despliegue a la mitad.

### 3.1 Stack Tecnológico Detallado

| **Capa** | **Tecnología** | **Justificación Técnica** |
| --- | --- | --- |
| **Core Framework** | **Flask (Python 3.11)** | Maneja Rutas, Lógica, Auth y Renderizado HTML. Un solo repositorio. |
| **Frontend** | **Jinja2 + Tailwind CSS** | **Server Side Rendering (SSR).** Elimina la necesidad de construir una API REST compleja. Tailwind vía CDN o build simple para estilos rápidos. |
| **Interactividad** | **JavaScript (Vanilla) / HTMX** | Para el chat en tiempo real sin la complejidad de React/Redux. |
| **Auth** | **Flask-Login** | Gestión de sesiones nativa (Cookies), más simple y segura que JWT para monolitos. |
| **Base de Datos** | **PostgreSQL 16** | Estándar relacional robusto. Alojado en contenedor Docker local. |
| **Infraestructura** | **Docker Compose** | Orquestación de servicios (App + DB) reproducible. |

## 4.0 Estrategia de Inteligencia Artificial

### 4.1 Selección del Modelo: Gemini 1.5 Flash (Google)

Se selecciona **Gemini 1.5 Flash** como el motor de inferencia por su optimización específica para tareas de alto volumen y baja latencia.

* **Ventana de Contexto (1M Tokens):** Permite mantener en memoria todo el historial del chat y el reporte inicial sin pérdida de coherencia.
* **Velocidad:** Latencia de primer token (TTFT) extremadamente baja, ideal para chat.

### 4.2 Análisis de Costos (IA)

| **Escenario** | **Modelo** | **Costo Mensual Est.** | **Capacidad** |
| --- | --- | --- | --- |
| **Fase 1 (Validación)** | **Google Free Tier** | \*\*$0.00 USD\*\* | ~50 Usuarios Concurrentes (15 RPM) |
| **Fase 2 (Escala)** | **Gemini Flash Pay-as-you-go** | ~$0.35 USD / 1M Tokens | Ilimitada |

## 5.0 Análisis Financiero (OpEx)

La arquitectura monolítica mantiene los costos fijos de infraestructura al mínimo.

### 5.1 Estructura de Costos Mensuales (Proyección)

| **Ítem** | **Proveedor Sugerido** | **Costo Mensual** | **Observación** |
| --- | --- | --- | --- |
| **Servidor VPS** | Hetzner (CPX11) / DigitalOcean | **$7.00 - $12.00** | 2 vCPU, 4GB RAM. |
| **Dominio** | Namecheap | **~$1.00** | Prorrateado anual ($12/año). |
| **Certificados SSL** | Let's Encrypt | \*\*$0.00\*\* | Renovación automática. |
| **Base de Datos** | Local (Docker) | **$0.00** | Incluido en el VPS. |
| **API IA** | Google AI Studio | **$0.00** | Gratuito hasta superar 1,500 req/día. |
| **Backups** | AWS S3 Glacier | **~$0.50** | Copias de seguridad frías. |
| **TOTAL OPEX** |  | **~$8.50 - $13.50 USD** | **Margen de operación > 90%** |

## 6.0 Plan de Implementación (Roadmap)

Al unificar el stack, el desarrollo se paraleliza y acelera.

### Fase 1: Core Monolítico (Semana 1)

* Configuración de Flask + SQLAlchemy + PostgreSQL (Docker).
* Implementación de **Flask-Login** (Registro/Login con plantillas HTML).
* Estructura base de plantillas Jinja2 con Tailwind CSS.

### Fase 2: Lógica de Negocio & IA (Semana 2)

* Integración SDK Gemini (ai\_service.py).
* Desarrollo de la lógica de "Ambiguity Check" y generación de los 5 Tópicos.
* Renderizado de vistas del reporte (Dashboard de usuario).

### Fase 3: Chat & Interacción (Semana 2-3)

* Implementación del Chatbot usando JavaScript simple (Fetch API) para comunicarse con Flask.
* Lógica de conteo de mensajes y bloqueo (Hard Cap).
* Integración visual de los componentes de chat en las plantillas.

### Fase 4: Despliegue (Semana 4)

* Configuración de Gunicorn como servidor de producción.
* Provisionamiento de VPS, Nginx y Certbot.
* Puesta en producción con Docker Compose.

## 

## 7.0 Análisis de Riesgos y Mitigación

1. **Escalabilidad UI:** Jinja2 es menos flexible que React para interfaces muy complejas.
   * *Mitigación:* El alcance actual (Formulario + Chat + Texto) es perfectamente manejable con HTML/JS estándar. Si crece mucho, se puede migrar a React progresivamente (Hybrid App).
2. **Bloqueo del Hilo Principal:** Flask es síncrono por defecto.
   * *Mitigación:* Usar workers asíncronos en Gunicorn o colas de tareas (Celery/Redis) si la generación de IA toma más de 30 segundos, para no congelar la interfaz del usuario.

## 8.0 Conclusión

El uso de una arquitectura **Monolítica en Flask** elimina la sobrecarga de gestionar dos bases de código separadas y simplifica el despliegue a un solo comando. Es la estrategia óptima para un desarrollador en solitario que busca eficiencia máxima y bajo mantenimiento.