# .Objetivo general:

# Desarrollar una plataforma web llamada Workspace que funcione como un intermediario digital eficiente entre empresas y usuarios que buscan proyectos temporales, permitiendo la publicación, búsqueda y gestión de oportunidades laborales de corta duración sin necesidad de establecer relaciones contractuales formales. El objetivo es ofrecer un sistema accesible, intuitivo y seguro que facilite la conexión entre oferta y demanda de proyectos, mejore la visibilidad de oportunidades para los usuarios, optimice los procesos de búsqueda para las empresas y garantice una experiencia confiable mediante herramientas de comunicación, perfiles profesionales y filtros de coincidencia basados en habilidades, duración y tipo de actividad.

# Objetivos específicos:

# Objetivos técnicos

# Diseñar e implementar una interfaz web intuitiva y accesible que permita a los usuarios registrarse, buscar proyectos y postularse de manera sencilla.

# Desarrollar el backend del sistema utilizando lenguajes y tecnologías apropiadas para garantizar eficiencia, escalabilidad y seguridad en el manejo de datos.

# Configurar una base de datos que almacene de forma segura información de usuarios, empresas y proyectos publicados.

# Integrar mecanismos de autenticación y verificación para garantizar la protección de datos y la confiabilidad de los perfiles dentro de la plataforma.

# Implementar servicios de almacenamiento y despliegue web en proveedores de hosting compatibles con las tecnologías empleadas. Objetivos funcionales

# Permitir que las empresas publiquen proyectos temporales con descripciones detalladas, requisitos y tiempo estimado.

# Brindar a los usuarios herramientas de filtrado y búsqueda avanzada para encontrar oportunidades acordes a sus habilidades y disponibilidad.

# Facilitar un sistema de comunicación interna entre usuarios y empresas para aclarar dudas y gestionar acuerdos.

# Integrar un sistema de reputación o evaluación que permita verificar el desempeño de los usuarios en proyectos anteriores.

# Objetivos administrativos y operativos

# Establecer un modelo de costos basado en técnicas de estimación paramétrica que permita proyectar el costo operativo mensual del servicio.

# Analizar la viabilidad económica de Workspace mediante el cálculo de costos e ingresos potenciales según el número de usuarios y proyectos mensuales.

# Definir políticas de uso, privacidad y términos del servicio para garantizar el cumplimiento de parámetros legales y éticos.

# Objetivos de impacto y alcance

# Fomentar el acceso a oportunidades laborales flexibles para estudiantes, trabajadores independientes y profesionales en general.

# Reducir el tiempo y complejidad del proceso de conexión entre empresas y colaboradores temporales.

# Crear un espacio digital confiable que impulse la colaboración, el trabajo temporal y la economía de proyectos.

# Requisitos Funcionales:

# RF1. Registro de usuarios

# El sistema deberá permitir que los usuarios (freelancers y empresas) puedan registrarse mediante correo electrónico y contraseña.

# RF2. Inicio de sesión

# El sistema permitirá que los usuarios ingresen con sus credenciales para acceder a sus perfiles y funciones.

# RF3. Recuperación de contraseña

# El usuario podrá recuperar su contraseña mediante correo electrónico.

# RF4. Creación y gestión de perfiles

# Los usuarios podrán crear y editar su perfil, incluyendo datos como nombre, habilidades, áreas profesionales e información de contacto.

# RF5. Publicación de proyectos por parte de empresas

# Las empresas podrán crear publicaciones donde especifiquen:

# nombre del proyecto,

# duración,

# pago (si aplica),

# tipo de actividad,

# habilidades requeridas.

# RF6. Búsqueda de proyectos por parte de usuarios

# Los usuarios podrán buscar proyectos mediante filtros como duración, categoría, habilidades requeridas o rango de pago.

# RF7. Sistema de aplicación a proyectos

# Los usuarios podrán enviar su solicitud a una vacante temporal o proyecto publicado.

# RF8. Panel administrativo básico

# Los administradores podrán gestionar usuarios, aprobar registros y revisar publicaciones sospechosas.

# RF10. Visualización del historial de proyectos

# Los usuarios podrán visualizar los proyectos a los que han aplicado o en los que han participado.

# RF11. Notificaciones

# El sistema enviará notificaciones sobre:

# nuevas publicaciones,

# respuesta a solicitudes,

# mensajes nuevos.

# Requisitos no Funcionales:

# RNF1. Escalabilidad

# El sistema deberá ser capaz de soportar un crecimiento progresivo de usuarios sin degradar el rendimiento.

# RNF2. Usabilidad

# La interfaz deberá ser intuitiva, clara y fácil de usar para personas sin conocimientos técnicos.

# RNF3. Disponibilidad

# El sistema deberá estar disponible el 99% del tiempo, considerando posibles fallos de hosting o mantenimiento.

# RNF4. Seguridad de datos

# Los datos personales deberán encriptarse y mantenerse bajo políticas de privacidad que protejan la información del usuario.

# RNF5. Rendimiento

# Las páginas deberán cargar en un tiempo promedio menor a 3 segundos bajo condiciones normales de conexión.

# RNF6. Compatibilidad multiplataforma

# La plataforma deberá funcionar correctamente en:

# navegadores web modernos,

# sistemas Windows, Linux y macOS,

# dispositivos móviles mediante diseño responsive.

# RNF7. Mantenibilidad

# El código deberá desarrollarse siguiendo buenas prácticas para facilitar futuras actualizaciones.

# RNF8. Confiabilidad

# El sistema deberá manejar adecuadamente errores como solicitudes incompletas, formularios inválidos o caídas parciales del servicio.

# RNF9. Protección ante fallos externos

# El sistema deberá soportar y recuperarse de fallos como:

# caída del proveedor cloud,

# interrupción temporal del internet del usuario,

# sobrecarga de tráfico.

# B)MODELO LOGICO

MODELO LOGICO

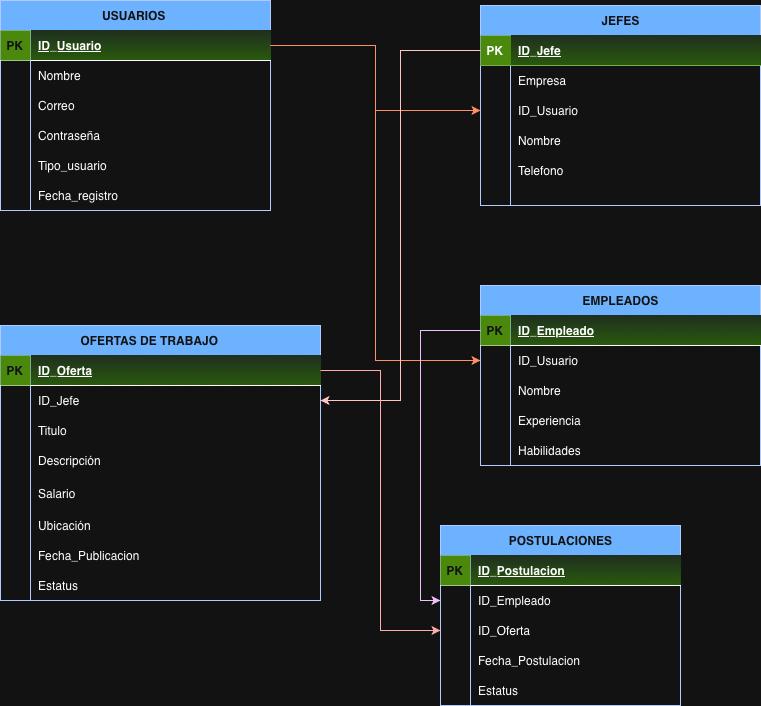


DIAGRAMA DE CASO DE USO

# 

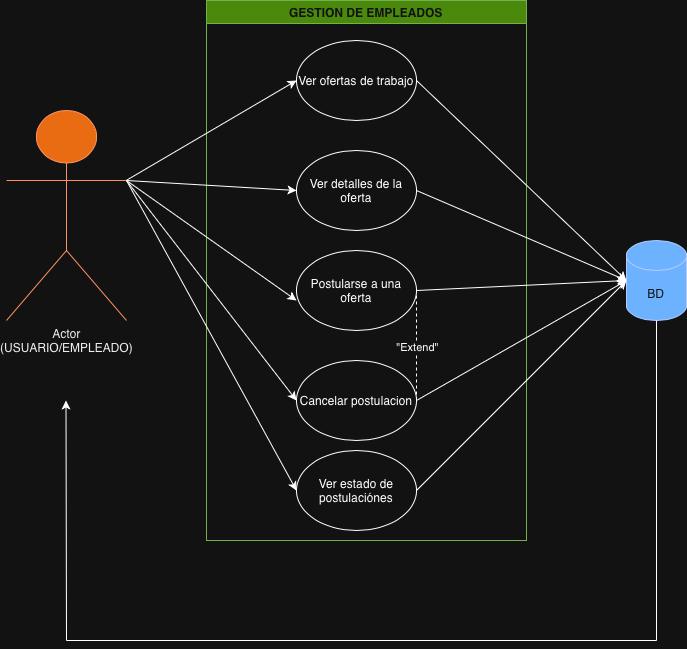
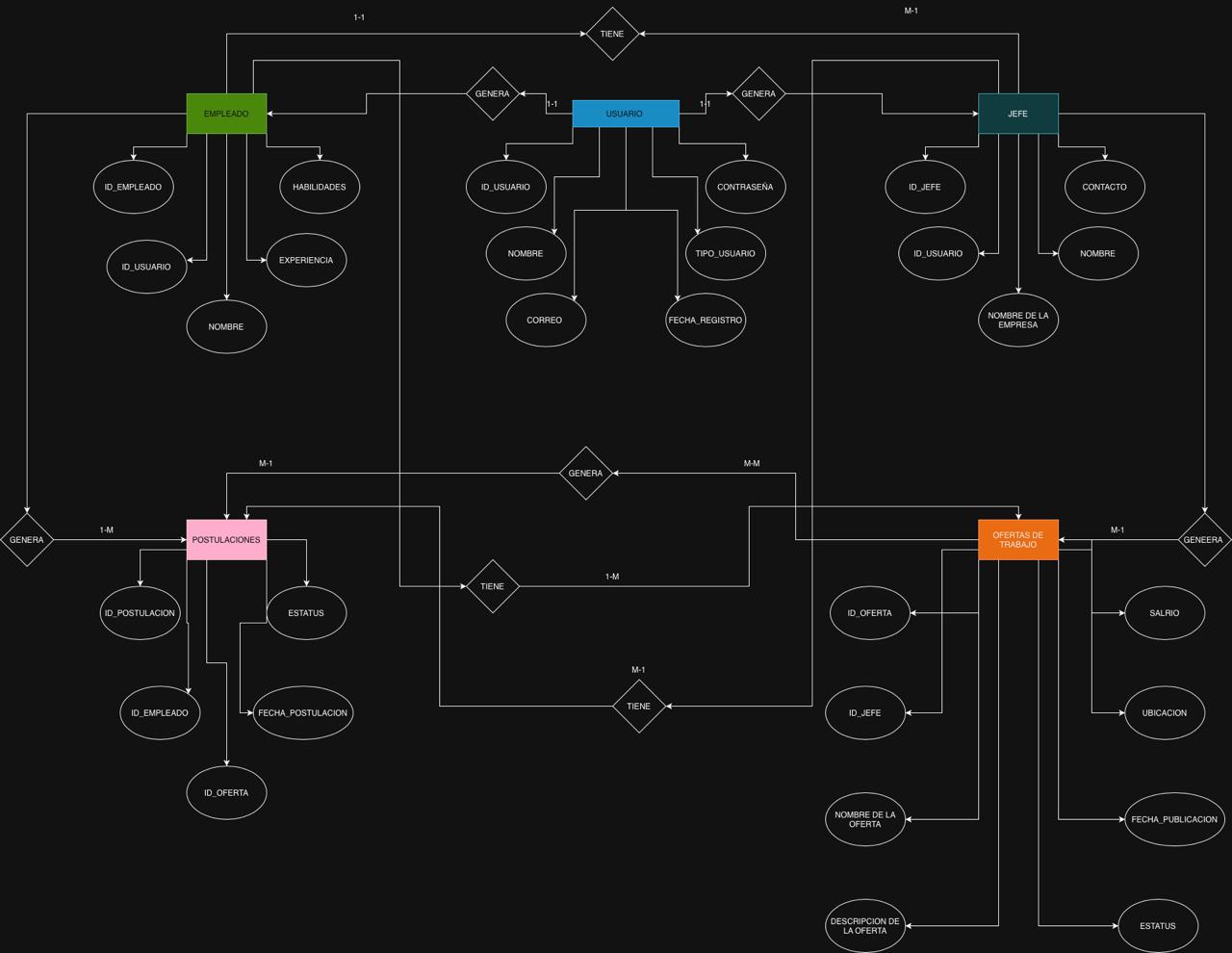


DIAGRAMA ENTIDAD RELACION



# C) IMPLEMENTACIÓN DEL SISTEMA WORKSPACE

## 1. Introducción

La fase de implementación de **WORKSPACE** consistió en la transformación de los requisitos y diseños previos en un sistema de software funcional. Se utilizó una arquitectura Modelo-Vista-Controlador (MVC), aprovechando la robustez de **Python** y **Flask** para el *backend*, y la flexibilidad de **PostgreSQL** para la persistencia de datos. El desarrollo se centró en la seguridad, la escalabilidad y la experiencia de usuario, implementando roles diferenciados y flujos de trabajo estrictos.

## 2. Configuración del Entorno de Desarrollo

Para garantizar la reproducibilidad y el aislamiento de dependencias, se configuró el siguiente entorno técnico:

* **Lenguaje:** Python 3.13.
* **Gestor de Paquetes:** pip.
* **Dependencias Clave (requirements.txt):**
  + Flask: Framework web principal.
  + Flask-SQLAlchemy: ORM para manejo de base de datos.
  + Flask-Login: Gestión de sesiones de usuario.
  + psycopg2-binary: Conector para PostgreSQL.
  + Werkzeug: Utilidades de seguridad (*hashing*).
  + gunicorn: Servidor de aplicación para despliegue en Render.

## 3. Desarrollo del Backend (Lógica del Servidor)

El núcleo del sistema reside en app.py.

### 3.1. Sistema de Autenticación

Se utilizó Werkzeug.security para no almacenar contraseñas en texto plano.

* **Registro:** Se genera un *hash* (generate\_password\_hash) antes de guardar en la BD.
* **Login:** Se compara el *hash* almacenado con la entrada del usuario (check\_password\_hash).

### 3.2. Lógica de Negocio (Rutas y Flujos Clave)

Se desarrollaron controladores específicos para cada flujo, asegurando la integridad transaccional:

* **Publicación de Ofertas:** Validación de formularios y creación de registros en la tabla job\_offers.
* **Postulación:** Verificación de duplicidad (un Worker no puede postularse dos veces al mismo proyecto) e inserción en la tabla applications.
* **Gestión de Estados:** Lógica para transicionar una solicitud de pending a accepted, rejected o completed. Se implementó el cierre automático de la oferta (job.status = 'closed') al aceptar un candidato, eliminando la publicación del *marketplace* público.
* **Protección de Rutas:** Se implementaron los decoradores @login\_required, @boss\_required y @worker\_required para aplicar seguridad a nivel de rol, evitando accesos no autorizados.

## 4. Implementación de la Base de Datos (ORM)

Se utilizó **SQLAlchemy** en el archivo models.py para mapear las clases de Python a tablas relacionales.

### 4.1. Estructura de Datos (Modelo Relacional)

Se implementó un modelo relacional normalizado:

* User: Tabla padre que contiene credenciales y el discriminador user\_type.
* Boss / Employee: Tablas hijas vinculadas por user\_id (Relación 1:1).
* JobOffer: Tabla de ofertas vinculada al Boss (Relación 1:N).
* Application: Tabla intermedia que vincula Employee y JobOffer (Relación N:M).

### 4.2. Integridad de Datos y Migración

Se añadieron restricciones para garantizar la coherencia:

* UniqueConstraint: Evita postulaciones duplicadas.
* CheckConstraint: Asegura que el campo status solo contenga valores válidos (pending, accepted, rejected, completed).
* **Despliegue Inicial (Solución Técnica):** Debido a las limitaciones de Render, la creación inicial y la migración de la base de datos se automatizó mediante un *script* (fix\_db.py) que fue ejecutado a través del **Start Command** de Gunicorn, asegurando que el *CheckConstraint* incluyera el estado 'completed'.

## 5. Desarrollo del Frontend (Interfaz de Usuario)

La implementación visual se realizó utilizando Jinja2 (motor de plantillas de Flask) y CSS3.

### 5.1. Diseño Adaptativo y Tematización

Se implementó un sistema visual basado en colores para mejorar la usabilidad y diferenciar los roles:

* **Tema Naranja:** Aplicado dinámicamente a las vistas del **Boss**.
* **Tema Azul:** Aplicado dinámicamente a las vistas del **Worker**.

Esto se logró mediante lógica condicional en las plantillas Jinja2 y CSS.

### 5.2. Feedback al Usuario

Se implementó el sistema de *flash messages* de Flask para notificar al usuario sobre el éxito o fracaso de sus acciones (ej. "Postulación enviada con éxito", "Credenciales incorrectas") sin necesidad de recargar la página completamente.

## 6. Pruebas Funcionales y Depuración (Ejecución Vía CMD)

Durante la implementación, la validación del sistema se realizó mediante pruebas funcionales de extremo a extremo, ejecutando el servidor de desarrollo directamente desde la **línea de comandos (CMD)** con python app.py y simulando las interacciones de los usuarios en el navegador.

| **Tipo de Prueba** | **Descripción del Procedimiento (Simulado en CMD/Navegador)** | **Validación de Éxito** |
| --- | --- | --- |
| **Prueba de Roles y Redirección** | Intentos de acceso directo a /perfilb siendo Worker, y acceso a /login estando ya autenticado. | El sistema forzó la redirección de los usuarios a su perfil correcto o a la página de inicio de sesión. |
| **Prueba de Integridad Referencial** | Simulación de la eliminación de un usuario Boss para verificar la respuesta de sus proyectos. | Los proyectos (job\_offers) del Boss eliminado fueron gestionados según las reglas relacionales (ON DELETE SET NULL). |
| **Prueba de Restricciones (Duplicidad)** | Intento de postular dos veces al mismo proyecto con el mismo Worker. | El sistema lanzó el IntegrityError (manejado en app.py) y mostró el mensaje amigable "Ya te has postulado". |
| **Prueba de Flujo de Estados** | Se validó el ciclo completo: Creación (Boss) → Postulación (Worker) → Aceptación (Boss) → Finalización (Worker) [cite: app.py]. | Se confirmó que el proyecto desapareciera de la búsqueda pública al ser aceptado, y que el estado final completed se guardara sin error, validando la corrección del *CheckConstraint*. |

## 7. Conclusión de la Implementación

La implementación de **WORKSPACE** resultó en una aplicación web robusta y segura. La separación de responsabilidades entre el *Backend* (lógica y seguridad) y el *Frontend* (experiencia de usuario) permitió crear un sistema escalable. El uso de tecnologías estándar de la industria (Flask, PostgreSQL) y la superación de los retos de despliegue en la nube aseguran que el proyecto cumple con los requisitos de calidad esperados para un entorno de producción.

# d) Gestión del Proyecto

## Cronograma de Actividades (Diagrama de Gantt)

El desarrollo del proyecto **WORKSPACE** se ha planificado y ejecutado en un periodo de 8 semanas, siguiendo una metodología de desarrollo iterativa. A continuación, se detalla el cronograma de actividades dividido por fases.

### Tabla de Planificación Temporal

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Fase** | **Tarea / Actividad** | **Duración** | **Semanas** |
| **1** | **Análisis y Requisitos** | **Definición del Alcance** | **1 semana** | **Semana 1** |
| 1.1 |  | Identificación de roles (Boss/Worker) y Requisitos Funcionales | 3 días | S1 |
| 1.2 |  | Especificación de Requisitos No Funcionales | 2 días | S1 |
| **2** | **Diseño del Sistema** | **Modelado de Datos y Arquitectura** | **1 semana** | **Semana 2** |
| 2.1 |  | Diseño del Diagrama Entidad-Relación (BD) | 2 días | S2 |
| 2.2 |  | Diseño de Diagramas de Casos de Uso y Flujo | 3 días | S2 |
| 2.3 |  | Diseño de Interfaz de Usuario (Mockups/HTML Estático) | 2 días | S2 |
| **3** | **Implementación Backend** | **Desarrollo del Servidor (Flask)** | **2 semanas** | **Semana 3-4** |
| 3.1 |  | Configuración del entorno (Virtualenv, Config.py) | 1 día | S3 |
| 3.2 |  | Implementación de Modelos BD (SQLAlchemy) | 3 días | S3 |
| 3.3 |  | Desarrollo del sistema de Autenticación (Login/Registro) | 3 días | S3 |
| 3.4 |  | Lógica de negocio: Rutas para Boss (Crear proyectos) | 3 días | S4 |
| 3.5 |  | Lógica de negocio: Rutas para Worker (Postulaciones) | 2 días | S4 |
| **4** | **Implementación Frontend** | **Integración y Vistas** | **2 semanas** | **Semana 5-6** |
| 4.1 |  | Desarrollo de Plantillas Base (Jinja2) y Estilos CSS | 3 días | S5 |
| 4.2 |  | Integración de formularios de Registro y Login | 2 días | S5 |
| 4.3 |  | Vistas de Perfiles (Boss/Worker) y Dashboards | 3 días | S6 |
| 4.4 |  | Vistas de Gestión: Detalles, Aceptar/Rechazar solicitudes | 2 días | S6 |
| **5** | **Pruebas y Depuración** | **Control de Calidad (QA)** | **1 semana** | **Semana 7** |
| 5.1 |  | Pruebas Unitarias (Modelos y Base de Datos) | 2 días | S7 |
| 5.2 |  | Pruebas de Integración (Flujo completo Boss -> Worker) | 2 días | S7 |
| 5.3 |  | Corrección de errores (Bugs de redirección, Constraints BD) | 3 días | S7 |
| **6** | **Despliegue y Cierre** | **Puesta en Producción** | **1 semana** | **Semana 8** |
| 6.1 |  | Configuración de Servidor (Gunicorn, Requirements.txt) | 1 día | S8 |
| 6.2 |  | Despliegue en Render.com y Configuración de PostgreSQL | 1 día | S8 |
| 6.3 |  | Elaboración de Manuales y Documentación Final | 3 días | S8 |

# e) Documentación del Proyecto WORKSPACE

## 1. Manual de Usuario

### 1.1. Introducción

**WORKSPACE** es una plataforma web diseñada para facilitar la conexión entre profesionales ("Workers") y gestores de proyectos ("Bosses"). Este manual guía al usuario a través de las funcionalidades principales del sistema, desde el registro hasta la finalización de un trabajo.

### 1.2. Acceso y Registro

El sistema cuenta con un sistema de autenticación seguro.

1. **Ingreso al Sistema:**
   * Acceda a la página principal (Index).
   * Haga clic en el botón "Login" o "Iniciar Sesión".
   * Ingrese su correo electrónico y contraseña.
2. **Registro de Nuevo Usuario:**
   * En la pantalla de Login, seleccione "¿No tienes cuenta? Crear cuenta".
   * **Selección de Rol:** Se le presentará una pantalla para elegir su perfil:
     + **Worker:** Para quienes buscan trabajo.
     + **Boss:** Para quienes ofrecen proyectos.
   * Complete el formulario correspondiente con sus datos personales y credenciales.

### 1.3. Perfil Boss (Jefe de Proyecto)

Al iniciar sesión como Boss, tendrá acceso a las siguientes herramientas:

* **Panel de Control (Perfil):** Visualización de sus datos de contacto y empresa.
* **Crear Proyecto:**
  1. Navegue a la opción "Crear Proyecto".
  2. Complete el formulario con el Título, Descripción, Ubicación y Presupuesto.
  3. Al guardar, el proyecto será visible inmediatamente para todos los Workers.
* **Gestión de Proyectos:**
  1. En "Mis Proyectos", verá una lista de sus ofertas publicadas.
  2. Haga clic en **"Ver / Administrar"** para ver los detalles.
  3. **Aceptar/Rechazar Candidatos:** En la vista de detalle, verá la lista de Workers que se han postulado. Puede "Aceptar" para contratar o "Rechazar" la solicitud.

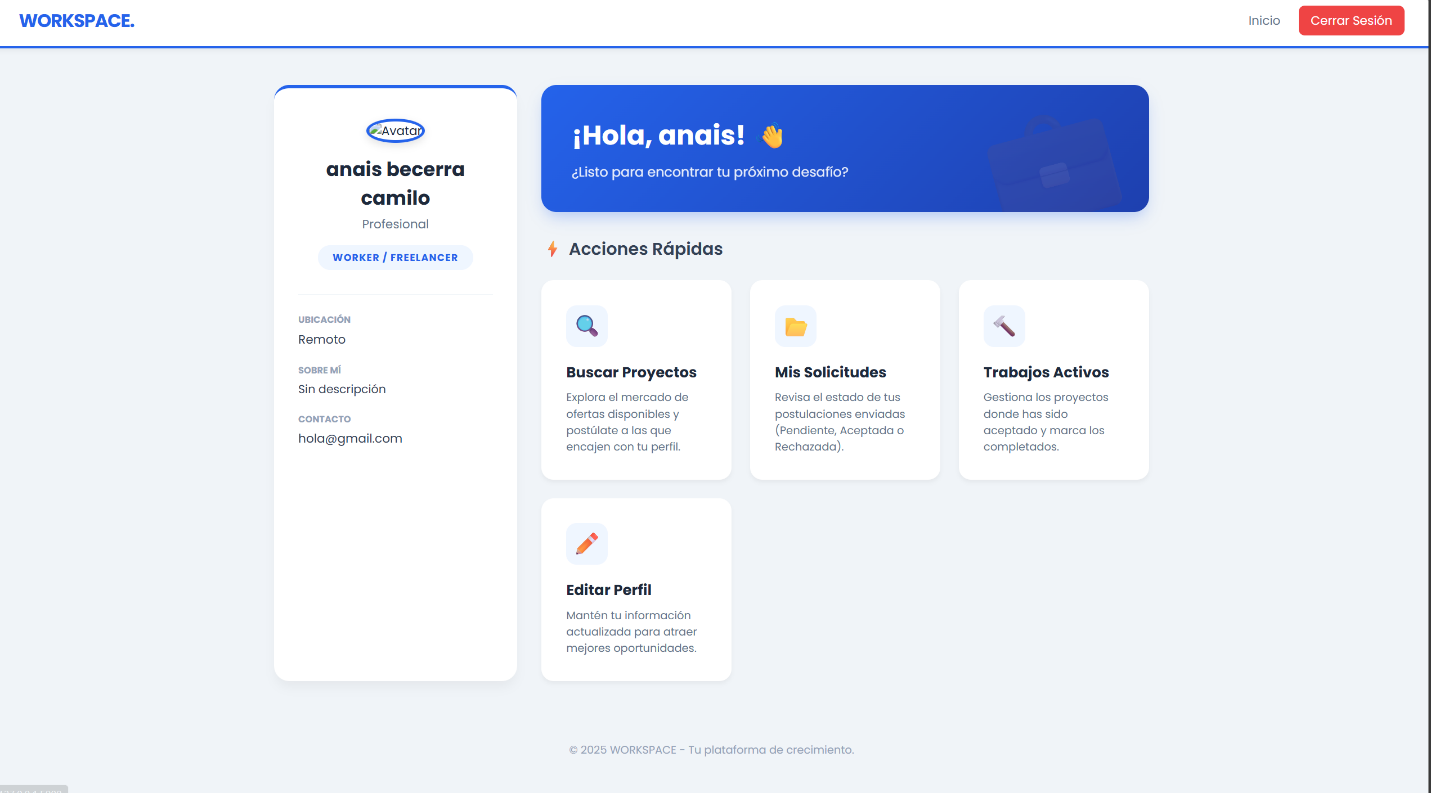
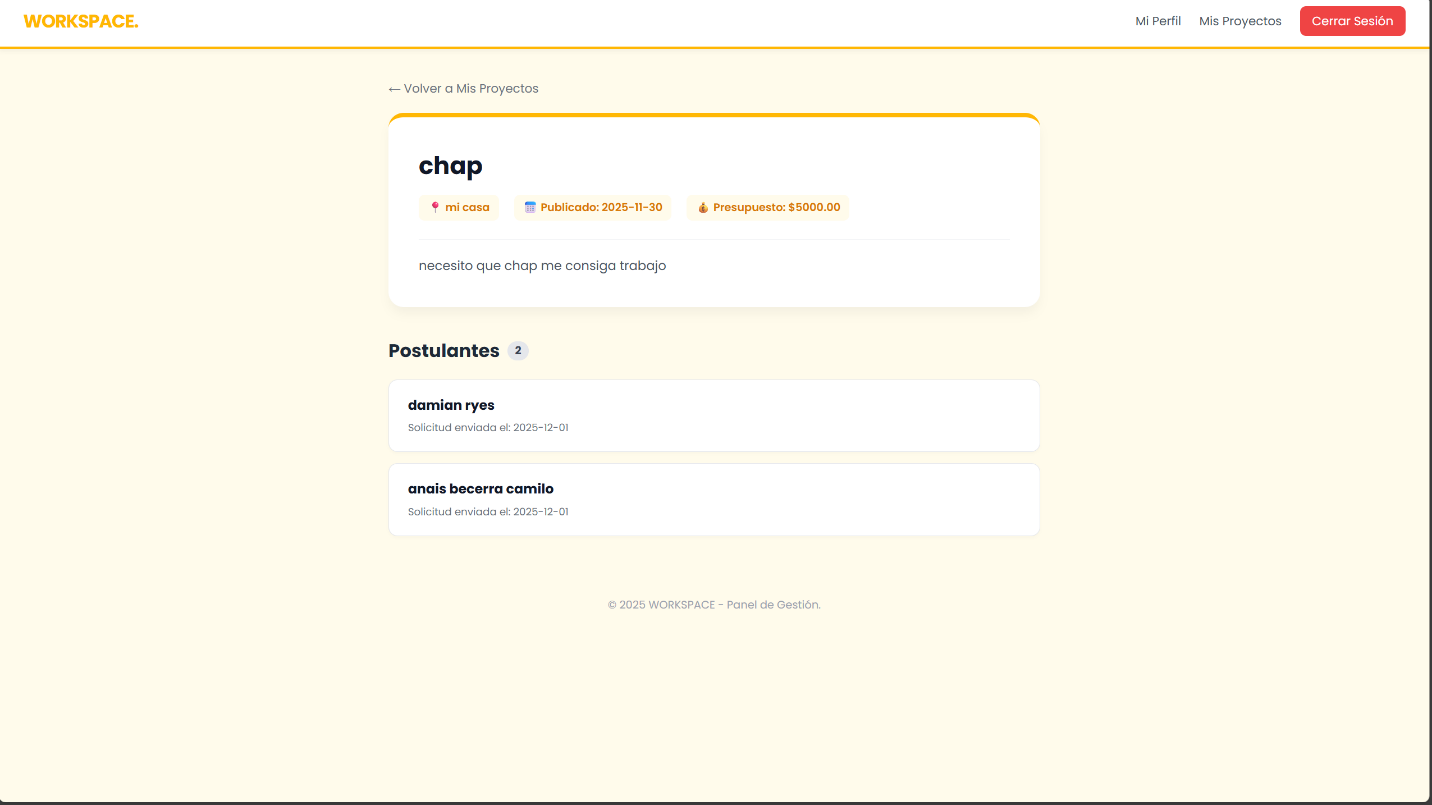
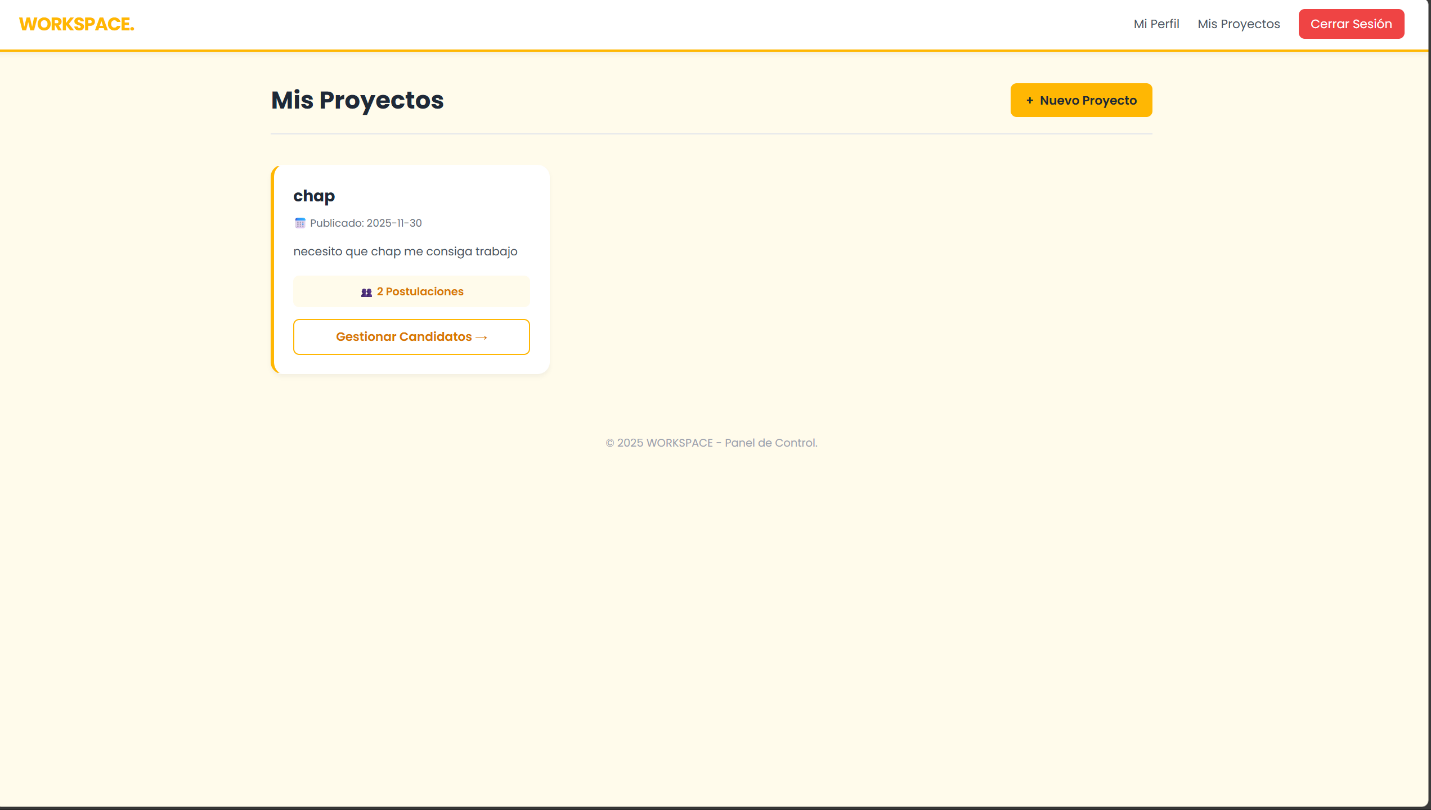
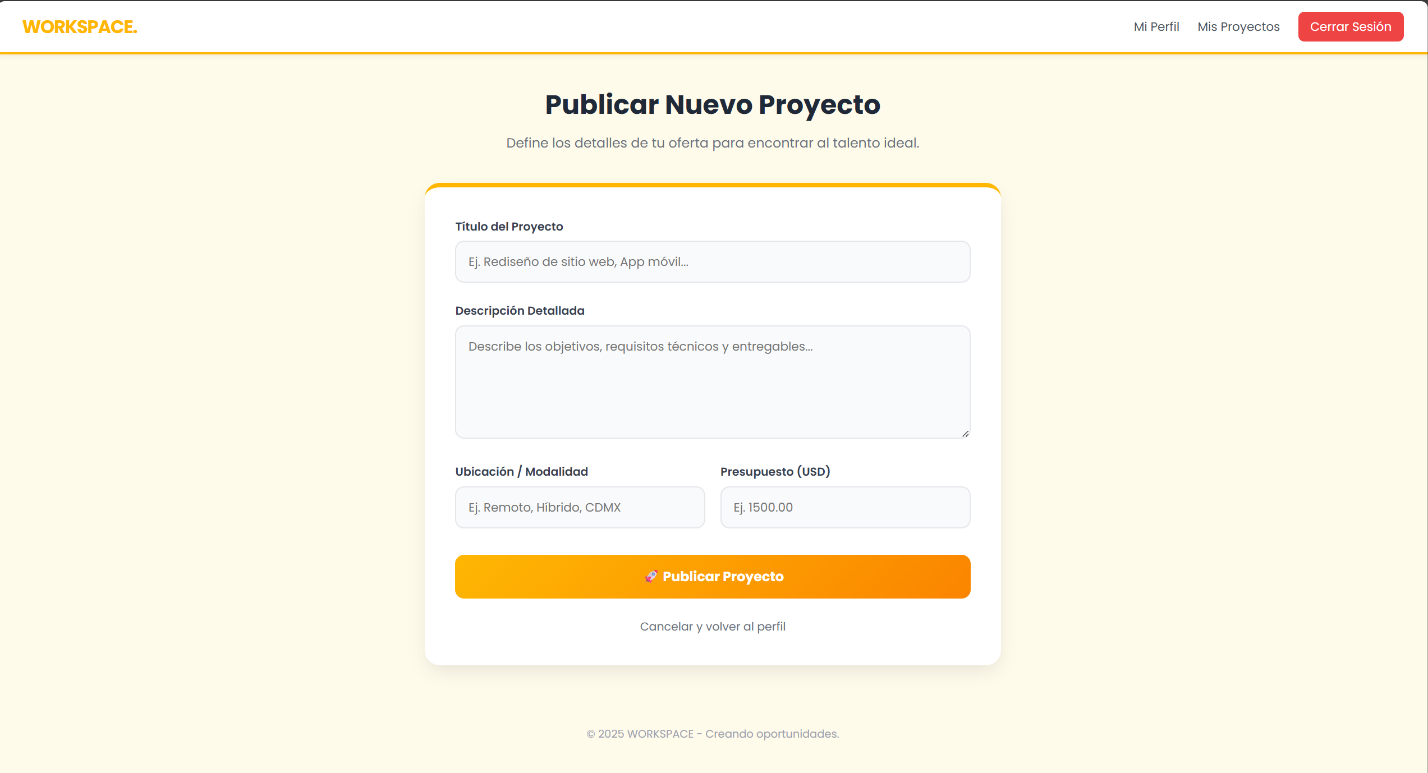
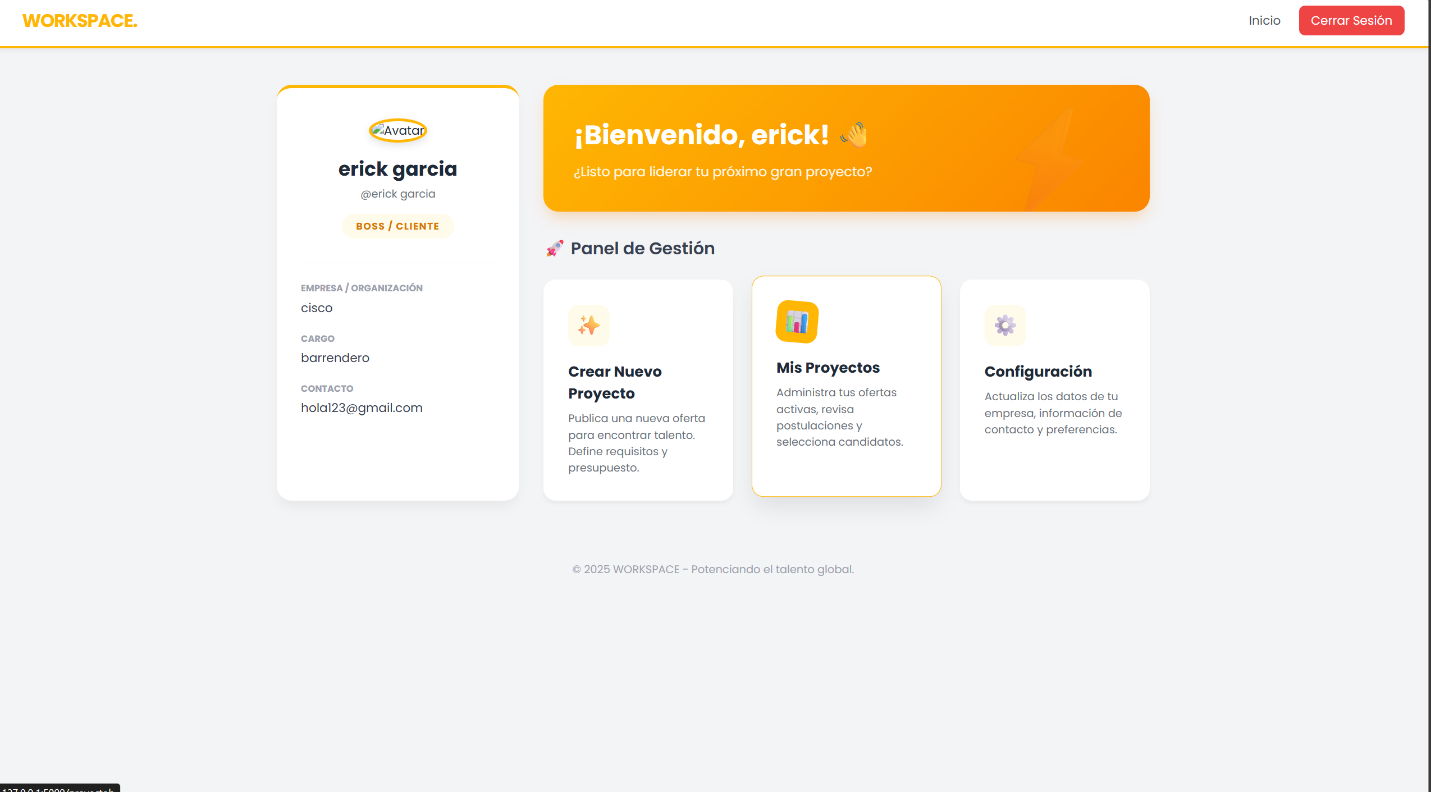
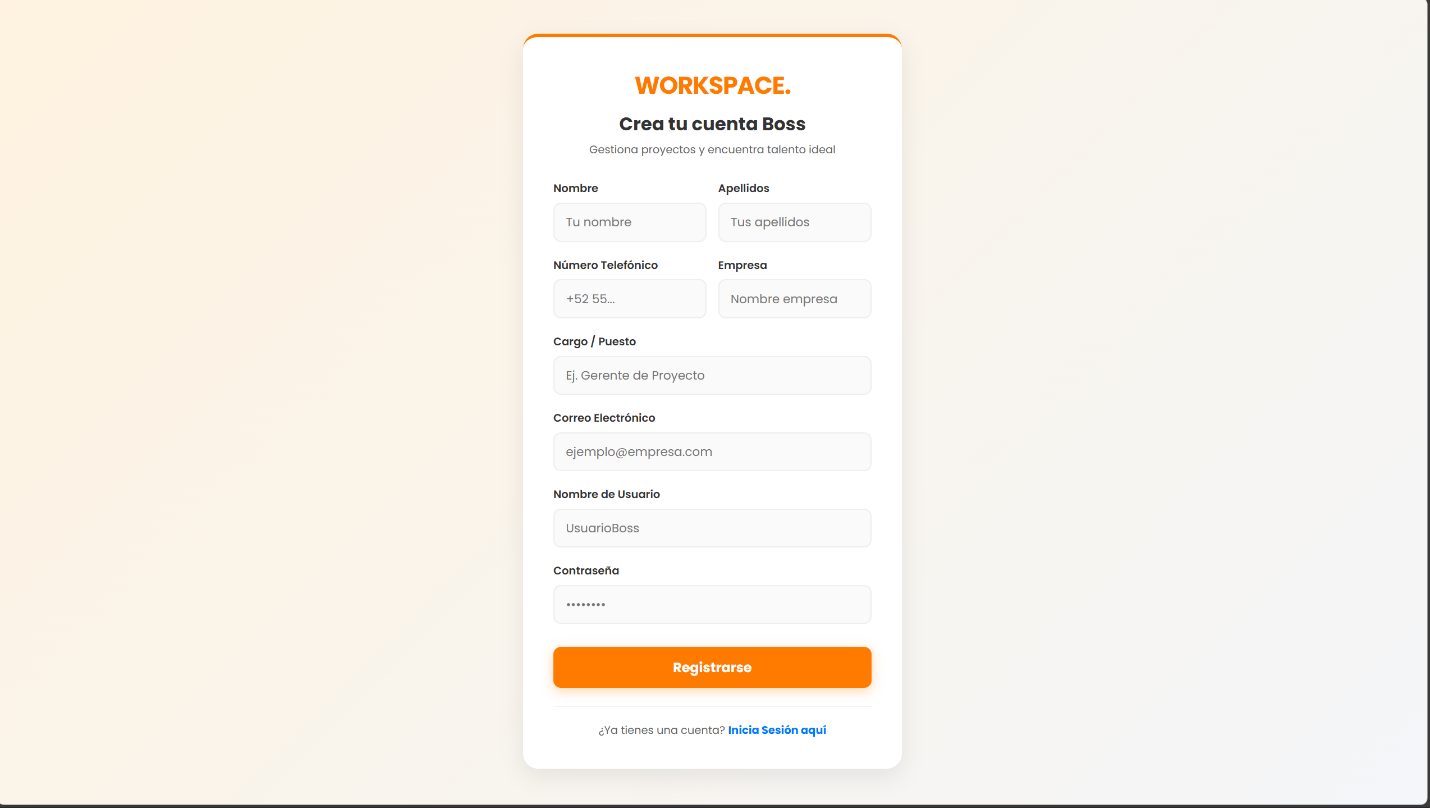
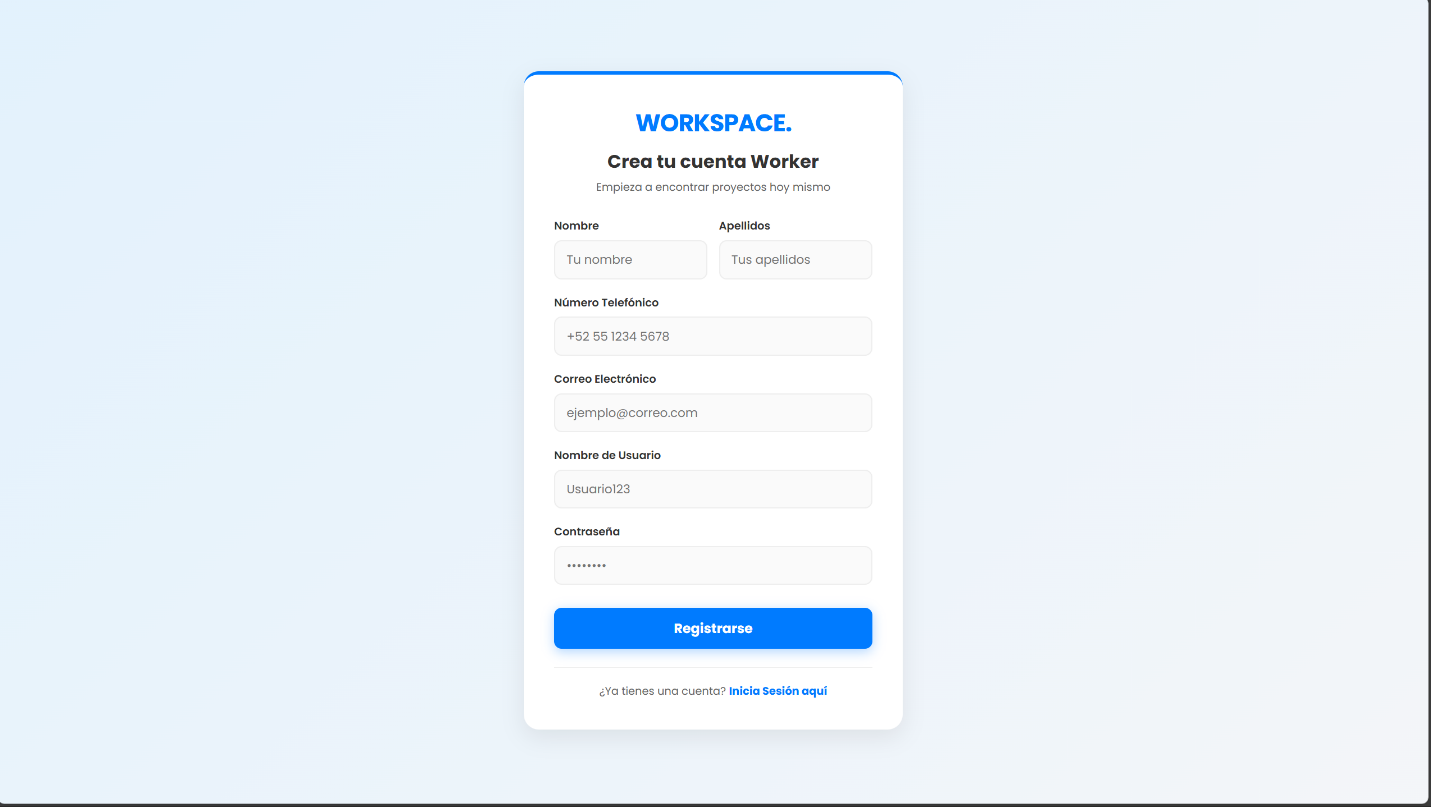
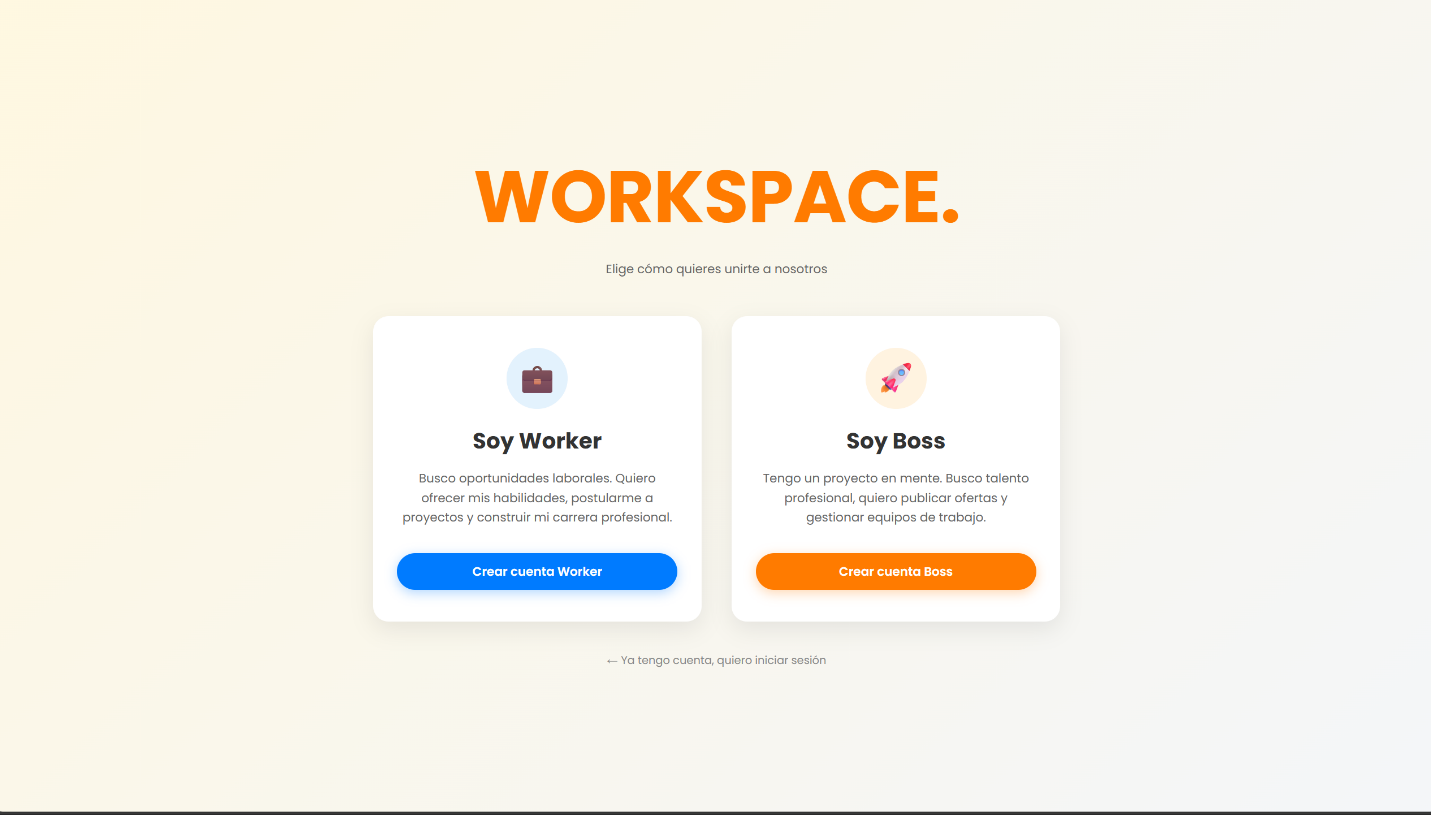
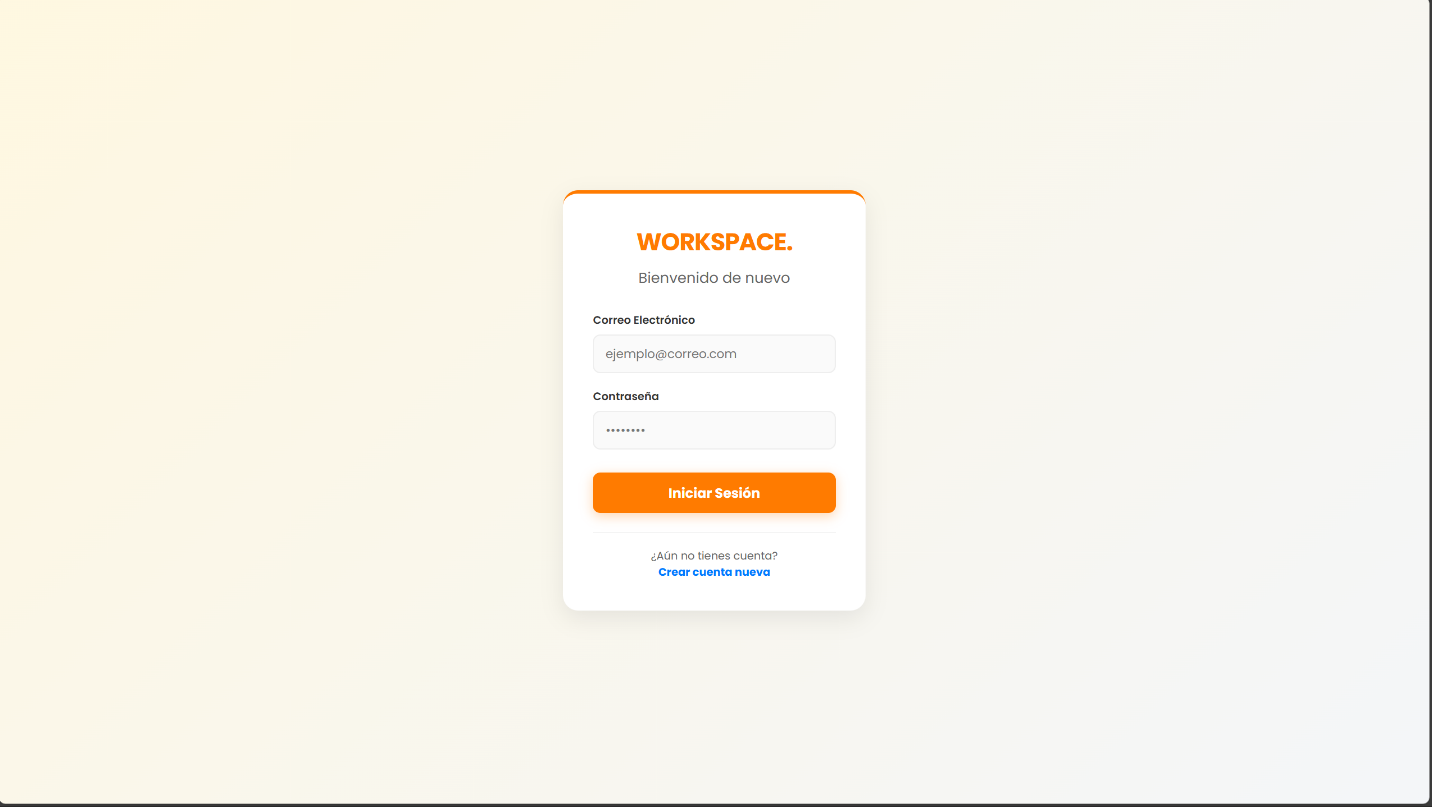
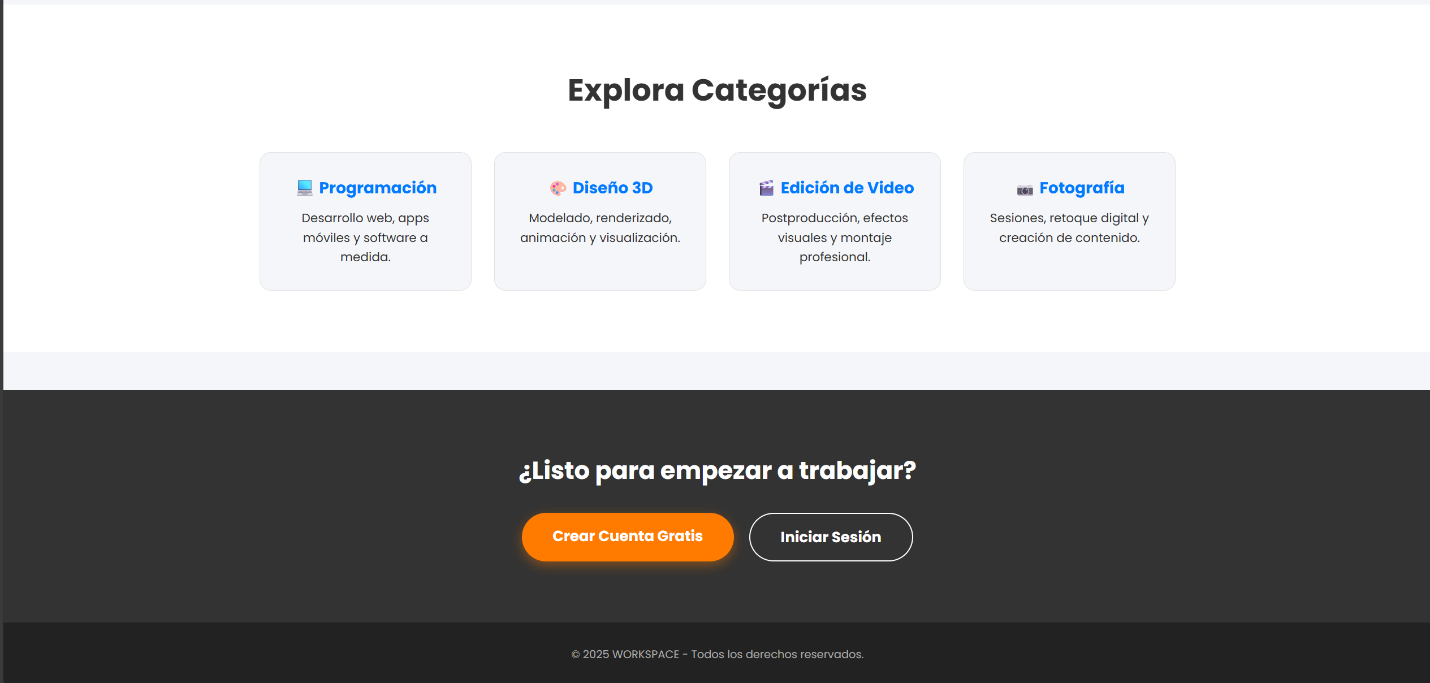
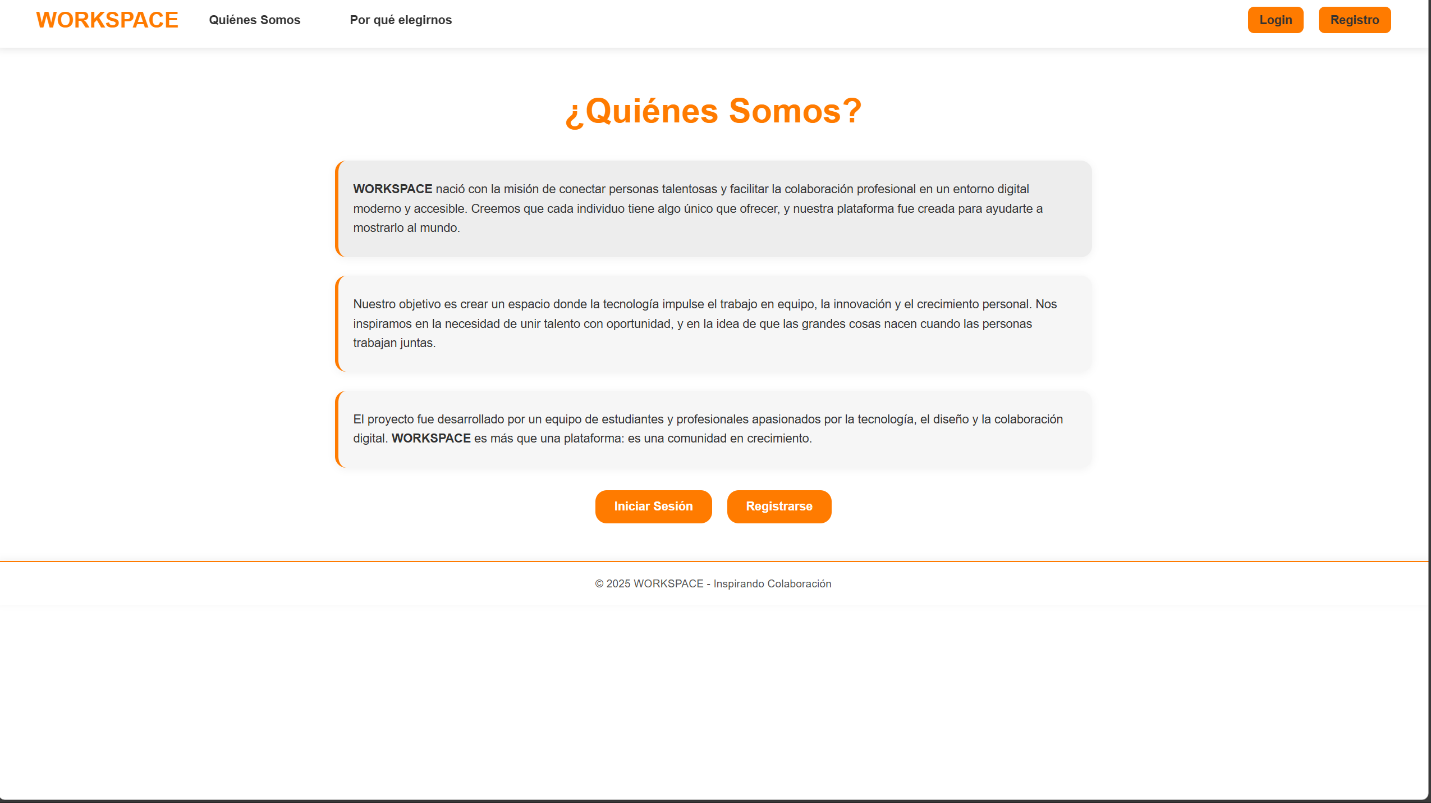
### 1.4. Perfil Worker (Trabajador)

Al iniciar sesión como Worker, podrá realizar las siguientes acciones:

* **Buscar Proyectos:**
  1. Acceda a la sección "Buscar Proyectos".
  2. Verá todas las ofertas disponibles con estado "Abierto".
  3. Haga clic en **"Postularme"** para enviar su interés al Boss.
* **Mis Solicitudes:**
  + Permite ver el estado de sus postulaciones (Pendiente, Aceptado o Rechazado).
* **Trabajos Pendientes:**
  + Aquí aparecen únicamente los proyectos donde ha sido **Aceptado**.
  + Esta sección se divide en:
    - **En Curso:** Trabajos activos. Puede ver detalles y contactar al cliente.
    - **Historial:** Trabajos que usted ha marcado como "Completados".
* **Finalizar Trabajo:** Dentro de los detalles de un trabajo en curso, utilice el botón "Marcar como Completado" para notificar la finalización.

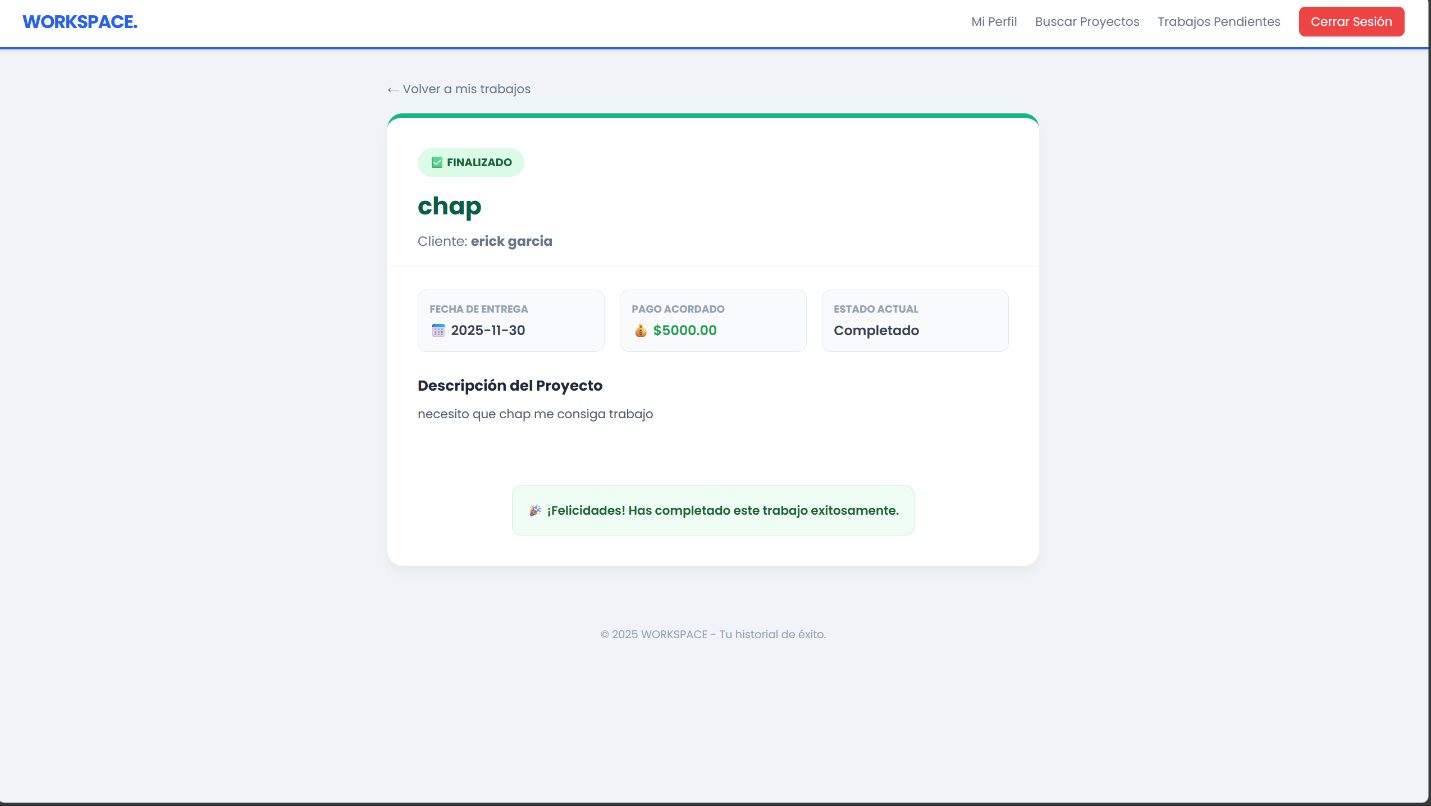
## 2. Manual del Sistema (Técnico)

### 2.1. Descripción General



Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

### 2.2. Arquitectura Tecnológica

* **Backend:** Python 3.13 + Flask.
* **Base de Datos:** PostgreSQL (Producción) / SQLite (Desarrollo).
* **ORM:** SQLAlchemy para el mapeo objeto-relacional.
* **Frontend:** HTML5, CSS3, Jinja2 (Motor de plantillas).
* **Autenticación:** Flask-Login con hashing de contraseñas (Werkzeug Security).

### 2.3. Estructura del Proyecto

El código fuente se organiza de la siguiente manera:

* app.py: Controlador principal. Contiene la configuración de la app, inicialización de extensiones y todas las rutas (endpoints) del sistema.
* models.py: Define las clases de la base de datos (User, Employee, Boss, JobOffer, Application).
* config.py: Variables de configuración y conexión a base de datos.
* templates/: Contiene los archivos HTML dinámicos.2.4. Modelo de Datos (Base de Datos)

El sistema utiliza un modelo relacional con las siguientes tablas principales:

1. **Users:** Tabla maestra de credenciales (email, password hash, tipo de usuario).
2. **Employees / Bosses:** Tablas de perfil vinculadas 1:1 con Users mediante Foreign Key.
3. **Job\_Offers:** Almacena las ofertas publicadas por los Bosses.
4. **Applications:** Tabla intermedia que registra la relación N:M entre Employees y Job\_Offers, controlando el estado de la postulación (pending, accepted, rejected, completed).

### 2.5. Requisitos de Instalación y Despliegue

Para ejecutar el sistema en un entorno nuevo:

1. **Instalar Dependencias:**  
   pip install -r requirements.txt
2. **Configurar Variables de Entorno:**
   * Definir DATABASE\_URL y SECRET\_KEY.

Texto

El contenido generado por IA puede ser incorrecto.

1. Captura de pantalla de un celular

   El contenido generado por IA puede ser incorrecto.
2. **Inicializar Base de Datos:**  
   flask init-db
3. Captura de pantalla de computadora

   El contenido generado por IA puede ser incorrecto.  
   *(Este comando crea automáticamente las tablas definidas en models.py).*
4. **Ejecución:**
   * Desarrollo: python app.py
5. Captura de pantalla de un celular

   El contenido generado por IA puede ser incorrecto.
   * Producción: gunicorn app:app

### 2.6. Seguridad

* **Contraseñas:** No se almacenan en texto plano. Se utiliza generate\_password\_hash con algoritmo pbkdf2:sha256.
* **Control de Acceso:** Uso de decoradores @login\_required, @boss\_required y @worker\_required para proteger rutas sensibles y evitar escalada de privilegios.
* **Protección SQL:** Uso de ORM (PostgresSQL) para prevenir inyecciones SQL.