## **Guía de Despliegue y Depuración para SOUP Emprendimientos**

Esta guía proporciona instrucciones detalladas para desplegar tanto el backend FastAPI como el frontend React de la aplicación SOUP Emprendimientos, así como un compendio de herramientas y técnicas de depuración utilizadas durante el desarrollo.

### **1. Despliegue del Servidor Backend (FastAPI)**

El backend de SOUP Emprendimientos está construido con FastAPI y Python, utilizando PostgreSQL como base de datos.

#### **1.1. Prerrequisitos**

* **Python 3.11+**: Asegúrate de tener Python instalado en tu sistema.
* **pip**: Gestor de paquetes de Python (viene con Python).
* **PostgreSQL**: Servidor de base de datos instalado y en ejecución (versión 14+ recomendada).
  + **Usuario de DB**: soupuser
  + **Contraseña de DB**: souppass
  + **Base de Datos**: soup\_app\_db
  + **Puerto de DB**: 5432
* **Dependencias de Python**: Asegúrate de que psycopg2-binary (para la conexión a PostgreSQL), FastAPI, Uvicorn, SQLAlchemy, Pydantic, python-dotenv, passlib[bcrypt] y python-jose[cryptography] estén instaladas.

#### **1.2. Configuración del Entorno (.env)**

El backend utiliza un archivo .env para la configuración sensible.

1. Navega a la carpeta backend/.
2. Crea un archivo llamado .env si no existe.
3. Añade las siguientes variables (ajusta si tus credenciales de DB son diferentes):  
   # Database configuration  
   DATABASE\_URL="postgresql+psycopg2://soupuser:souppass@localhost:5432/soup\_app\_db"  
     
   # JWT Secret Key (generate a strong, random key for production)  
   SECRET\_KEY="tu\_clave\_secreta\_jwt\_muy\_larga\_y\_segura"  
   ALGORITHM="HS256"  
   ACCESS\_TOKEN\_EXPIRE\_MINUTES=30  
   * Importante: Para producción, SECRET\_KEY debe ser una cadena aleatoria muy larga y compleja. Puedes generarla con Python:  
     import secrets; print(secrets.token\_hex(32))

#### **1.3. Inicialización de la Base de Datos**

Antes de iniciar el servidor, la base de datos soup\_app\_db y sus tablas deben existir.

1. **Detén cualquier instancia previa del servidor FastAPI.**
2. **Conéctate a PostgreSQL como el superusuario postgres**:
   * Abre una terminal de Command Prompt.
   * Ejecuta: psql -U postgres -h localhost -p 5432 (ingresa la contraseña de postgres).
   * Si tienes problemas de acceso, asegúrate de que estás usando la ruta correcta a psql.exe (C:\Program Files\PostgreSQL\<VERSION>\bin\psql.exe) o que el usuario postgres tiene privilegios adecuados.
3. **Ejecuta los siguientes comandos SQL en el prompt postgres=#**:  
   -- Eliminar la base de datos si existe (¡Cuidado en producción!)  
   DROP DATABASE IF EXISTS soup\_app\_db;  
   -- Crear la base de datos y asignarle el propietario  
   CREATE DATABASE soup\_app\_db OWNER soupuser;  
   -- Otorgar todos los privilegios al usuario sobre la base de datos  
   GRANT ALL PRIVILEGES ON DATABASE soup\_app\_db TO soupuser;
4. **Sal de psql**: \q
5. **Inicia el servidor FastAPI** (ver sección 1.4). Durante el inicio, app.main:app llamará a create\_db\_tables() que creará todas las tablas definidas en app/models.py.
   * Verifica los logs: Deberías ver Creating database tables... seguido de INFO sqlalchemy.engine.Engine CREATE TABLE... para cada tabla y Database tables created..

#### **1.4. Ejecución del Servidor Backend**

1. Navega a la carpeta backend/ en tu terminal.
2. **Activa el entorno virtual**: .\venv\Scripts\activate (Windows) o source venv/bin/activate (Linux/macOS).
3. **Inicia el servidor Uvicorn**:  
   uvicorn app.main:app --reload  
   * --reload: Útil para desarrollo, recarga el servidor automáticamente al detectar cambios en el código. **No usar en producción.**
   * El servidor estará disponible en http://127.0.0.1:8000.

#### **1.5. Notas Importantes para Producción**

* **Uvicorn sin --reload**: En producción, no uses --reload.
* **Gunicorn/Nginx**: Para un despliegue robusto en producción, es común usar Gunicorn (servidor WSGI) para manejar Uvicorn y Nginx (servidor web) como proxy inverso y para servir archivos estáticos.
* **Variables de Entorno**: No almacenes credenciales directamente en el código. Usa variables de entorno del sistema o un sistema de gestión de secretos.
* **Base de Datos en la Nube**: Para producción, considera servicios de base de datos gestionados (AWS RDS, Google Cloud SQL, Heroku Postgres).

### **2. Despliegue del Frontend (React)**

El frontend de SOUP Emprendimientos está construido con React y utiliza Tailwind CSS.

#### **2.1. Prerrequisitos**

* **Node.js**: Asegúrate de tener Node.js instalado (incluye npm).
* **npm**: Gestor de paquetes de Node.js.

#### **2.2. Configuración de la API del Backend**

1. Navega a la carpeta frontend/src/api/.
2. Abre authApi.js.
3. Verifica que const API\_BASE\_URL = 'http://localhost:8000'; apunte a la URL correcta de tu backend. Para despliegue, esto cambiará a la URL pública de tu API.

#### **2.3. Instalación de Dependencias**

1. Navega a la carpeta frontend/ en tu terminal.
2. Instala las dependencias:  
   npm install
3. **Componentes Shadcn UI**: Si los componentes no se han generado, asegúrate de que los archivos de configuración existan y luego generarlos:
   * frontend/components.json
   * frontend/jsconfig.json
   * Luego: npx shadcn@latest add button card input label select textarea
   * Asegúrate de que los componentes estén en frontend/src/components/ui/ y que frontend/src/lib/utils.js exista y contenga la función cn.

#### **2.4. Ejecución del Frontend en Desarrollo**

1. Navega a la carpeta frontend/ en tu terminal.
2. **Inicia el servidor de desarrollo de React**:  
   npm start  
   * El frontend estará disponible en http://localhost:3000.

#### **2.5. Creación de la Build para Producción**

1. Navega a la carpeta frontend/ en tu terminal.
2. **Crea la build optimizada para producción**:  
   npm run build  
   * Esto generará una carpeta build/ con todos los archivos estáticos listos para ser servidos.

#### **2.6. Despliegue del Frontend en Producción**

* **Servicio de Archivos Estáticos**: La carpeta build/ puede ser servida por cualquier servidor web (Nginx, Apache, Caddy) o un servicio de hosting estático (Netlify, Vercel, Firebase Hosting, AWS S3 + CloudFront).
* **Nginx/Apache**: Configura tu servidor web para servir los archivos de la carpeta build/.
* **serve (para pruebas rápidas de producción local)**:  
  npm install -g serve  
  serve -s build -l 3000  
    
  Esto servirá tu aplicación de producción en el puerto 3000.

### **3. Utilidades y Técnicas de Depuración**

Durante el desarrollo, hemos encontrado varios problemas. Aquí se resumen las herramientas y estrategias utilizadas para solucionarlos.

#### **3.1. Depuración del Backend (FastAPI/Python/PostgreSQL)**

* **Terminal de Uvicorn**:
  + **Logs INFO**: Indican el estado del servidor (inicio, reinicios, solicitudes, creación de tablas).
  + **Logs ERROR y Tracebacks**: **Cruciales para identificar errores en el código Python del backend.** Un 500 Internal Server Error en el navegador casi siempre corresponde a un traceback en esta terminal.
  + **SQLAlchemy Logs (INFO sqlalchemy.engine.Engine)**: Muestran las consultas SQL que se están ejecutando. Muy útil para verificar si las tablas se están creando (CREATE TABLE) o si las consultas a la base de datos están fallando.
* **FastAPI Swagger UI (/docs)**:
  + Acceso en http://127.0.0.1:8000/docs.
  + Permite probar los endpoints de la API directamente (enviar solicitudes POST, GET, etc.) y ver los esquemas de Pydantic esperados. Ideal para aislar si el problema es del frontend o del backend.
* **Terminal psql (PostgreSQL Command Line Client)**:
  + psql -U <usuario> -d <db\_nombre> -h <host> -p <puerto>: Para conectarse a la base de datos con un usuario específico (ej. psql -U soupuser -d soup\_app\_db).
  + \dt: Lista las tablas en la base de datos actual. **Esencial para verificar si las tablas han sido creadas.**
  + DROP DATABASE <nombre\_db>; / CREATE DATABASE <nombre\_db>;: Para recrear la base de datos y asegurar un estado limpio en desarrollo.
  + GRANT ALL PRIVILEGES ON DATABASE <nombre\_db> TO <usuario>;: Para asegurar que el usuario tenga los permisos necesarios.
* **Intérprete Interactivo de Python (REPL)**:
  + Inicia con python en la terminal del backend (con venv activado).
  + from app.database import Base, engine: Importa la instancia Base de SQLAlchemy.
  + import app.models: Carga todos tus modelos.
  + print(Base.metadata.tables): **Muestra todas las tablas que SQLAlchemy ha reconocido y registrado.** Si esta salida es FacadeDict({}), significa que los modelos no están siendo registrados correctamente, lo que impide la creación de tablas. Esto fue clave para depurar el error UndefinedTable.
* **Archivo .env**: Verificar que las credenciales y URLs de la base de datos sean correctas y que no haya errores de sintaxis o caracteres ocultos.

#### **3.2. Depuración del Frontend (React)**

* **Consola del Navegador (Herramientas de Desarrollador - F12)**:
  + **Mensajes de Error (en rojo)**: Proporcionan información sobre errores de JavaScript en el código React.
  + **Failed to fetch**: Indica que la solicitud HTTP no pudo completarse. Generalmente es un problema de conectividad con el backend o CORS.
  + **Códigos de Estado HTTP**:
    - 400 Bad Request: Datos enviados incorrectos.
    - 401 Unauthorized: Autenticación fallida (ej., token inválido).
    - 403 Forbidden: El usuario no tiene permisos para la acción.
    - 422 Unprocessable Entity: Datos válidos, pero no cumplen las restricciones del backend (validación Pydantic).
    - 500 Internal Server Error: Algo falló en el servidor (backend).
* **Pestaña "Red" (Network) del Navegador**:
  + Permite inspeccionar cada solicitud HTTP que el frontend envía y recibe.
  + **Headers**: Verificar el token de autorización (si es un endpoint protegido).
  + **Payload (Carga)**: Ver los datos que el frontend envía al backend.
  + **Response (Respuesta) / Preview (Vista previa)**: Ver los datos y mensajes de error exactos que el backend devuelve. **Crucial para entender los errores 422 y 500**.
* **Terminal del Frontend (donde corres npm start)**:
  + **Errores de Compilación (Webpack)**: Indican problemas en el código JavaScript/React (ej., Module not found, SyntaxError, errores de transpilación).
  + **Advertencias de ESLint**: Aunque no detienen la compilación, señalan posibles problemas de rendimiento o estilo en el código.
* **frontend/src/api/authApi.js**: Verificar que API\_BASE\_URL apunte a la dirección correcta del backend.
* **Problemas de Shadcn UI**:
  + **Coincidencia de mayúsculas/minúsculas (casing)**: Asegurarse de que las importaciones de componentes (import { Button } from '...') coincidan exactamente con el nombre de archivo del componente (Button.jsx). Windows es indulgente, Linux no.
  + **Rutas de importación relativas**: Si los alias (@/) no funcionan, usar rutas relativas explícitas (ej., ../../components/ui/Button).
  + **Archivos de configuración**: frontend/components.json y frontend/jsconfig.json deben estar presentes y bien formados para que Shadcn UI y React entiendan la estructura del proyecto y los alias.
  + **src/lib/utils.js**: Este archivo con la función cn es esencial para el funcionamiento de los componentes de Shadcn UI. Asegurarse de que exista y que los componentes lo importen correctamente.

Este informe servirá como un registro detallado de las soluciones a problemas comunes y específicos encontrados durante las primeras fases de SOUP Emprendimientos, facilitando la depuración y el desarrollo futuro.