## **Informe de Compilación - Fases 0 y 1 de SOUP Emprendimientos**

Este informe documenta las decisiones clave, los bugs encontrados y sus soluciones durante la implementación de la Fase 0 (Configuración Inicial y Estructura del Proyecto) y la Fase 1 (Autenticación y Gestión de Perfil de Usuario) de SOUP Emprendimientos.

### **Fase 0: Configuración Inicial y Estructura del Proyecto**

**Objetivo:** Establecer la arquitectura básica del proyecto con un backend FastAPI (Python) conectado a PostgreSQL y un frontend React con Tailwind CSS.

#### **1. Decisiones Tomadas**

* **Arquitectura:**
  + **Backend:** FastAPI (Python) por su alto rendimiento, facilidad de uso, validación automática de datos con Pydantic y generación de documentación API (Swagger UI).
  + **Base de Datos:** PostgreSQL por su robustez, escalabilidad y amplio soporte para características avanzadas (como PostGIS, considerado para fases futuras). SQLAlchemy como ORM para la interacción con la base de datos.
  + **Frontend:** React por su popularidad, gran ecosistema, flexibilidad y rendimiento en la construcción de interfaces de usuario interactivas.
  + **Estilos Frontend:** Tailwind CSS por su enfoque utility-first que permite un desarrollo rápido y un alto grado de personalización sin escribir CSS tradicional.
* **Gestión de Entornos:** Uso de entornos virtuales (venv) tanto para el backend (Python) como para el frontend (Node.js/npm) para aislar las dependencias y evitar conflictos.
* **Configuración de Base de Datos:**
  + Usuario de PostgreSQL: soupuser
  + Contraseña de PostgreSQL: souppass
  + Base de Datos: soup\_app\_db
  + Puerto: 5432 (por defecto de PostgreSQL)
* **Configuración de Backend:**
  + Servidor ASGI: Uvicorn para servir FastAPI.
  + Puerto del Backend: 8000.
  + CORS: Configuración explícita del middleware CORS para permitir la comunicación entre el frontend (localhost:3000) y el backend (localhost:8000).
* **Configuración de Frontend:**
  + Iniciador de proyecto: Create React App (CRA) para una configuración rápida.
  + Puerto del Frontend: 3000.
  + Componentes UI: Shadcn UI para componentes pre-estilizados y accesibles con Tailwind CSS.

#### **2. Bugs Encontrados y Soluciones (Fase 0)**

La mayoría de los problemas en esta fase fueron de configuración inicial:

* **Bug 0.1: uvicorn no reconocido.**
  + **Causa:** El entorno virtual (venv) del backend no estaba activado en la terminal al intentar ejecutar uvicorn.
  + **Solución:** Activar el entorno virtual usando .\venv\Scripts\activate antes de ejecutar uvicorn.
* **Bug 0.2: IndentationError: unexpected indent en main.py.**
  + **Causa:** Un espacio o tabulación extra al inicio de una línea en Python, donde la indentación es crucial para la sintaxis.
  + **Solución:** Eliminar el indentado extra en la línea afectada en backend/app/main.py.
* **Bug 0.3: Module not found para componentes de Shadcn UI (Button, Card, etc.) en el frontend.**
  + **Causa:**
    - Los comandos npx shadcn-ui@latest add ... (versión antigua) o npx shadcn@latest add ... (versión actual) no se ejecutaron correctamente o no crearon los componentes en la ubicación esperada.
    - Falta de archivos de configuración (components.json, jsconfig.json) para que Shadcn UI y React entiendan la estructura del proyecto y los alias.
    - Rutas de importación incorrectas en los archivos React (apuntando a Common en lugar de ui, o problemas de mayúsculas/minúsculas).
    - Los componentes se generaron inicialmente en una ruta inesperada (src/@/components/ui/) debido a cómo shadcn interpretó el proyecto sin una configuración completa.
    - Falta del archivo de utilidad src/lib/utils.js que los componentes de Shadcn UI requieren.
  + **Solución:**
    1. Instalar las dependencias básicas de Shadcn (npm install class-variance-authority clsx lucide-react tailwind-merge tailwindcss-animate).
    2. Crear frontend/components.json para configurar Shadcn UI.
    3. Crear frontend/jsconfig.json para definir alias (@/components, @/lib) y configuración de JavaScript para el proyecto.
    4. Ejecutar el comando correcto de Shadcn UI: npx shadcn@latest add button card input label select textarea.
    5. **Mover manualmente** los componentes generados de src/@/components/ui/ a la ruta estándar src/components/ui/ (si se generaron incorrectamente).
    6. Crear el archivo frontend/src/lib/utils.js con la función cn.
    7. **Corregir las rutas de importación** en todos los archivos de pantalla (LoginScreen, RegisterScreen, DashboardScreen, ProfileScreen) para que apunten correctamente a los componentes en ../../components/ui/ o ../components/ui/ (dependiendo de la profundidad del archivo) y asegúrarse de que el **casing** (mayúsculas/minúsculas) de los nombres de archivo (Button, Card, etc.) coincida exactamente.
    8. Reemplazar los alias @/lib/utils dentro de los archivos de componentes de Shadcn (button.jsx, card.jsx, etc.) por rutas relativas ../../lib/utils para asegurar que React los resuelva correctamente.

### **Fase 1: Autenticación y Gestión de Perfil de Usuario**

**Objetivo:** Implementar el registro, login y gestión básica del perfil de usuario en el backend y frontend.

#### **1. Decisiones Tomadas**

* **Modelado de Usuario:** Se definió el modelo Usuario en backend/app/models.py con campos para nombre, email, password\_hash, localización, info de contacto (JSONB), tipo de tier (CLIENTE, FREELANCER, MICROEMPRENDIMIENTO) y timestamps.
* **Esquemas de Validación:** Se crearon esquemas Pydantic en backend/app/schemas.py (UserCreate, UserLogin, UserResponse, Token, TokenData) para validar los datos de entrada y salida de la API.
* **Autenticación JWT:** Se implementó JWT (JSON Web Tokens) para la autenticación, con funciones para hashear contraseñas (bcrypt), crear y decodificar tokens de acceso (auth.py). OAuth2PasswordBearer se usó para la seguridad en FastAPI.
* **CRUD de Usuario:** Se desarrollaron funciones CRUD (create\_user, get\_user\_by\_email, update\_user\_profile, update\_user\_cv) en backend/app/crud/user.py para interactuar con la base de datos.
* **Endpoints API:** Se definieron los routers FastAPI en backend/app/routers/auth\_router.py (/register, /login) y backend/app/routers/user\_router.py (/me, /me/cv para perfil y CV del freelancer).
* **Estado de Autenticación Frontend:** Se implementó un AuthContext en React (frontend/src/context/AuthContext.js) para manejar el estado de autenticación de forma global, incluyendo el almacenamiento del token en localStorage y la redirección.
* **Lógica de API Frontend:** Se centralizaron las llamadas a la API de autenticación en frontend/src/api/authApi.js.
* **Pantallas Frontend:** Se crearon las pantallas de LoginScreen, RegisterScreen, ProfileScreen y DashboardScreen en frontend/src/screens/.

#### **2. Bugs Encontrados y Soluciones (Fase 1)**

Los bugs en esta fase se centraron en la comunicación backend-frontend y la persistencia de la base de datos.

* **Bug 1.1: Failed to fetch / net::ERR\_CONNECTION\_REFUSED.**
  + **Causa:** El frontend no podía conectar con el backend. Generalmente, el backend no estaba ejecutándose o el puerto/URL configurado en authApi.js era incorrecto.
  + **Solución:** Asegurarse de que el servidor FastAPI esté siempre corriendo en http://localhost:8000 y que la API\_BASE\_URL en authApi.js coincida.
* **Bug 1.2: 422 Unprocessable Entity en el registro.**
  + **Causa:** Los datos enviados desde el formulario de registro del frontend no cumplían con las reglas de validación del esquema Pydantic en el backend. Específicamente, la contraseña no cumplía con min\_length=8.
  + **Solución:** Revisar los detalles del error en la pestaña "Red" (Network) del navegador para identificar el campo problemático (en este caso, la contraseña) y asegurarse de que los datos de entrada del formulario de registro cumplan con los requisitos del backend. Ingresar una contraseña de al menos 8 caracteres.
* **Bug 1.3: 500 Internal Server Error con UndefinedTable: no existe la relación «usuarios».**
  + **Causa:** La tabla usuarios (y otras) no existía en la base de datos PostgreSQL a la que FastAPI estaba intentando conectarse. Aunque el servidor de FastAPI decía "Database tables created.", las tablas no se estaban materializando. Esto se debía a un problema con la instancia de Base de SQLAlchemy. Había múltiples instancias o los modelos no estaban correctamente registrados con la Base.metadata global.
  + **Solución:**
    1. **Asegurar una única instancia de Base**: Modificar backend/app/models.py para **eliminar** la línea Base = declarative\_base() y, en su lugar, **importar Base desde app.database** (from app.database import Base). Esto asegura que todos los modelos hereden de la misma instancia de Base definida en database.py.
    2. **Forzar la carga de modelos**: Asegurarse de que import app.models esté en backend/app/main.py antes de Base.metadata.create\_all() para que todos los modelos se registren.
    3. **Depuración profunda con psql y REPL de Python**: Para confirmar que las tablas no existían (\dt mostraba "No se encontró ninguna relación.") y que Base.metadata.tables estaba vacío (FacadeDict({}) en el REPL), lo que confirmó el problema de registro de modelos.
    4. **Recreación limpia de la base de datos**: Eliminar (DROP DATABASE) y volver a crear (CREATE DATABASE) la base de datos soup\_app\_db desde psql (conectado como postgres), y luego otorgar privilegios a soupuser. Esto garantiza un estado limpio.
    5. **Reiniciar el backend**: Permitir que FastAPI, con los modelos ahora correctamente registrados, cree las tablas en la base de datos limpia al inicio. La verificación en los logs de FastAPI (INFO sqlalchemy.engine.Engine CREATE TABLE...) confirmó la creación exitosa.

Este informe cubre las decisiones fundamentales y los desafíos superados en las dos primeras fases del proyecto. La base está sólidamente establecida para el desarrollo futuro.