# **Flujo de Trabajo para Codificar SOUP Emprendimientos con Asistencia de IA**

Con el documento actual, podemos estructurar el flujo de trabajo de codificación de manera **modular y eficiente**, siguiendo el Principio de Responsabilidad Única (SRP) y aprovechando la asistencia de IA en cada paso.

## **1. Estructura de Alto Nivel y División en Archivos (SRP)**

Tu propuesta de determinar la arquitectura de alto nivel y la división en archivos según el Principio de Responsabilidad Única es **excelente y la mejor práctica.**

Aquí tienes una sugerencia de cómo estructurar el proyecto para el backend (Python/FastAPI) y el frontend (React/JavaScript/TypeScript):

### **a. Estructura de Proyecto Sugerida**

/SOUP\_Emprendimientos  
├── /backend  
│ ├── /app  
│ │ ├── main.py # Configuración de FastAPI, routers principales  
│ │ ├── database.py # Conexión a DB, sesión de SQLAlchemy  
│ │ ├── models.py # Definiciones de Modelos de Datos (SQLAlchemy)  
│ │ ├── schemas.py # Pydantic Schemas para Request/Response (validación de datos)  
│ │ ├── dependencies.py # Dependencias inyectables (ej. get\_db, get\_current\_user)  
│ │ ├── auth.py # Lógica de autenticación (JWT, hashing de contraseñas)  
│ │ ├── /crud # Operaciones CRUD básicas de base de datos  
│ │ │ ├── user.py  
│ │ │ ├── negocio.py  
│ │ │ ├── producto.py  
│ │ │ ├── insumo.py  
│ │ │ ├── contacto.py  
│ │ │ ├── mensaje\_registro.py  
│ │ │ ├── encargo.py  
│ │ │ ├── review.py  
│ │ │ ├── publicidad.py  
│ │ ├── /services # Lógica de negocio más compleja, cálculos  
│ │ │ ├── pricing\_service.py # Lógica de la calculadora de precios (COGS, margen)  
│ │ │ ├── report\_service.py # Lógica de generación de informes  
│ │ │ ├── ia\_service.py # Interfaz con el modelo de IA (gemini-2.0-flash)  
│ │ │ ├── communication\_service.py # Lógica para generar links externos de mensajería  
│ │ ├── /routers # Endpoints de API (organizados por módulo)  
│ │ │ ├── auth\_router.py  
│ │ │ ├── user\_router.py  
│ │ │ ├── negocio\_router.py  
│ │ │ ├── producto\_router.py  
│ │ │ ├── contacto\_router.py  
│ │ │ ├── mensaje\_router.py  
│ │ │ ├── encargo\_router.py  
│ │ │ ├── informe\_router.py  
│ │ │ ├── marketplace\_router.py  
│ │ │ ├── review\_router.py  
│ │ │ ├── advertising\_router.py  
│ │ │ ├── ia\_router.py # Endpoints para interacciones directas con IA  
│ │ ├── /core # Configuraciones globales, settings  
│ │ │ ├── config.py  
│ │ │ ├── security.py  
│ │ ├── /utils # Funciones de utilidad (ej. generadores UUID)  
│ │ │ ├── geospatial\_utils.py # Para PostGIS  
│ ├── .env.example  
│ ├── requirements.txt  
│ ├── Dockerfile  
│ └── README.md  
├── /frontend  
│ ├── /public  
│ │ └── index.html  
│ ├── /src  
│ │ ├── App.js # Router principal, gestión de estado global  
│ │ ├── index.js # Punto de entrada de React  
│ │ ├── /api # Funciones para interactuar con el Backend API  
│ │ │ ├── authApi.js  
│ │ │ ├── usersApi.js  
│ │ │ ├── productsApi.js  
│ │ │ ├── marketplaceApi.js  
│ │ │ # ... y más archivos por cada dominio de la API  
│ │ ├── /components # Componentes UI reutilizables pequeños  
│ │ │ ├── Auth # Login, Register  
│ │ │ ├── Common # Botones, Inputs, Modales  
│ │ │ ├── Forms # Formularios específicos  
│ │ │ ├── Layout # Navbar, Sidebar, Footer  
│ │ │ ├── ProductCards.js # Tarjeta de producto para Marketplace  
│ │ │ ├── BusinessProfile.js # Perfil del negocio para Marketplace  
│ │ │ # ... etc.  
│ │ ├── /screens # Componentes de "pantalla" o "página" (vistas principales)  
│ │ │ ├── AuthScreens # LoginScreen, RegisterScreen  
│ │ │ ├── DashboardScreen.js  
│ │ │ ├── ProfileScreen.js  
│ │ │ ├── BusinessManagementScreen.js  
│ │ │ ├── ProductServiceManagementScreen.js  
│ │ │ ├── EncargosManagementScreen.js  
│ │ │ ├── ClientsSuppliersScreen.js  
│ │ │ ├── ReportsScreen.js  
│ │ │ ├── MarketplacePublicScreen.js  
│ │ │ ├── PublicProfileScreen.js  
│ │ │ ├── PublicProductDetailScreen.js  
│ │ ├── /hooks # Custom React Hooks (ej. useAuth, useForm)  
│ │ ├── /context # React Context para estado global (ej. AuthContext)  
│ │ ├── /utils # Funciones de utilidad (formato de fechas, validaciones)  
│ │ ├── /assets # Imágenes, iconos, etc.  
│ │ ├── index.css # Tailwind CSS base y custom styles  
│ ├── package.json  
│ ├── postcss.config.js  
│ ├── tailwind.config.js  
│ └── README.md  
├── .gitignore  
├── README.md

### **b. Justificación del SRP en esta Estructura:**

* **models.py vs. schemas.py:** models.py define cómo se guardan los datos en la DB (SQLAlchemy ORM). schemas.py define cómo se ven los datos cuando se envían a través de la API (Pydantic, para validación y serialización). Son responsabilidades distintas pero relacionadas.
* **crud/ vs. services/:** crud/ maneja las operaciones directas de DB (Crear, Leer, Actualizar, Borrar). services/ contiene la lógica de negocio *compleja* que usa las funciones CRUD (ej., pricing\_service calcula el precio usando datos de producto y insumo, report\_service extrae y agrega datos).
* **routers/:** Solo se encargan de definir los endpoints HTTP y orquestar las llamadas a los servicios y operaciones CRUD.
* **Frontend api/ vs. components/ vs. screens/:**
  + api/: Responsabilidad de la comunicación con el backend.
  + components/: Responsabilidad de partes pequeñas y reutilizables de la UI.
  + screens/: Responsabilidad de la lógica de una página completa, orquestando componentes y llamadas a la API.

## **2. Flujo de Trabajo Propuesto para la Codificación con IA**

Dado que tienes un excelente documento de especificaciones, podemos seguir un enfoque iterativo y modular, pidiéndole a la IA que construya partes específicas en cada paso.

**Paso 1: Configuración del Proyecto y Modelos de Datos (Backend Core)**

1. **Generar la estructura básica de carpetas y archivos** (main.py, database.py, config.py, requirements.txt, package.json, etc.).
2. **Definir las dependencias iniciales** en requirements.txt (FastAPI, SQLAlchemy, psycopg2-binary, python-dotenv, passlib, python-jose, uvicorn) y package.json (React, react-dom, tailwindcss, react-router-dom, lucide-react, shadcn/ui).
3. **Implementar la conexión a la base de datos** (database.py) y la configuración básica de main.py.
4. **Generar los modelos de datos de SQLAlchemy** en models.py basándose en las entidades que has detallado. Incluye las relaciones.
5. **Generar los esquemas de Pydantic** en schemas.py correspondientes a cada modelo, para Request (entrada de datos) y Response (salida de datos).
6. **Crear las operaciones CRUD básicas** para cada modelo en la carpeta crud/.

**Instrucción a la IA:** "Genera la configuración inicial de un proyecto FastAPI con SQLAlchemy y PostgreSQL. Incluye la definición de los modelos de datos para Usuario, Negocio, Producto, Insumo, Contacto, MensajeRegistro, Encargo, Review, Publicidad, según las especificaciones proporcionadas, usando SQLAlchemy ORM. También genera los esquemas Pydantic correspondientes para cada modelo (Create, Update, Read)."

**Paso 2: Autenticación y Perfil de Usuario (Backend y Frontend)**

1. **Backend:**
   * Implementar la lógica de autenticación en auth.py (hashing de contraseña, JWT token creation/validation).
   * Crear los routers de autenticación en routers/auth\_router.py (/register, /login).
   * Implementar el router de usuario (/me) en routers/user\_router.py.
2. **Frontend:**
   * Crear los componentes LoginScreen y RegisterScreen (src/components/Auth/).
   * Implementar la integración con la API de autenticación (src/api/authApi.js).
   * Configurar el router en App.js para las rutas de autenticación.

**Instrucción a la IA:** "Basándose en el stack de FastAPI/SQLAlchemy, implementa la autenticación de usuarios (registro, login, perfil GET /me) usando JWT. Crea los modelos, esquemas, lógica de hashing y los routers necesarios. Luego, genera los componentes de React para las pantallas de Login y Registro, que se conecten a estos endpoints."

**Paso 3: Gestión de Negocios y Productos/Servicios (Backend y Frontend)**

1. **Backend:**
   * Implementar los routers para Negocio y Producto (routers/negocio\_router.py, routers/producto\_router.py).
   * Crear la lógica de la calculadora de precios (services/pricing\_service.py) y el endpoint POST /productos/calculate\_price.
2. **Frontend:**
   * Crear las pantallas BusinessManagementScreen (para crear/listar negocios/perfiles de freelancer).
   * Crear ProductServiceManagementScreen y ProductServiceForm (el formulario dinámico de insumos, COGS, precio sugerido, y la llamada a la IA).

**Instrucción a la IA:** "Genera los endpoints de FastAPI para la gestión CRUD de Negocios y Productos/Servicios. Incluye la lógica de la calculadora de precios para productos, que tome insumos y margen para calcular COGS y precio sugerido. Luego, crea los componentes de React para el formulario de creación de Productos/Servicios con la sección dinámica de insumos y la visualización de la calculadora, conectándose a los endpoints relevantes."

**Paso 4: Marketplace Público (Backend y Frontend)**

1. **Backend:**
   * Implementar el router del marketplace (routers/marketplace\_router.py) con GET /marketplace/search, GET /marketplace/negocios/{id}, GET /marketplace/productos/{id}. Asegúrate de incluir el soporte para filtros de ubicación (PostGIS).
2. **Frontend:**
   * Crear MarketplacePublicScreen con búsqueda, filtros y visualización de tarjetas.
   * Crear PublicProfileScreen y PublicProductDetailScreen para los detalles públicos.

**Instrucción a la IA:** "Desarrolla la funcionalidad de búsqueda pública para el Marketplace de SOUP Emprendimientos. Crea los endpoints de FastAPI para buscar negocios y productos con filtros de texto, rubro y ubicación (usando PostGIS). Luego, genera los componentes de React para la interfaz de búsqueda, la visualización de resultados como tarjetas y las páginas de detalle para perfiles de negocios y productos públicos."

**Pasos Posteriores (Iterativos):**

* **Gestión de Encargos:** (Backend y Frontend)
* **Gestión de Clientes/Proveedores y Mensajería:** (Backend y Frontend)
* **Informes y Publicidad:** (Backend y Frontend)
* **Integración Real de la IA (Gemini API):** Una vez que las funcionalidades core estén estables, integrar la llamada real a la API de Gemini para las sugerencias de precios, insights y plantillas de mensajes.

## **Conclusión sobre el Flujo de Trabajo**

Este enfoque modular te permite:

* **Validar partes del sistema rápidamente:** Puedes tener un login y creación de productos funcionales en las primeras iteraciones.
* **Gestionar la complejidad:** No intentas construir todo a la vez.
* **Aprovechar la IA:** La IA puede generar los esqueletos de código, las operaciones CRUD, los formularios y la integración básica, lo que acelera enormemente tu proceso como desarrollador. Luego tú (el arquitecto y visionario) revisas, refinas y te aseguras de que el código generado se alinee con tu visión única.

¡Tienes todo lo necesario para comenzar este emocionante viaje de codificación!