Prompt Maestro para el Desarrollo de "AlgoGuardian"

Asunto: Modelo Mental Completo de la Aplicación SaaS "AlgoGuardian".

Objetivo: Este documento sirve como un "modelo mental" detallado de la aplicación "AlgoGuardian". Su propósito es proporcionar una comprensión profunda no solo del **qué** (la arquitectura del código, los componentes y la tecnología) sino también del **porqué** (la lógica de negocio, el valor para el usuario y los objetivos del producto). Utiliza este documento como base para todas las futuras solicitudes de desarrollo y análisis.

---

1. Visión General y Propósito Principal

"AlgoGuardian" es una aplicación SaaS (Software as a Service) diseñada para *traders algorítmicos*.

- **Problema Principal que Resuelve:** Las estrategias de trading algorítmico, que funcionan bien en pruebas históricas (backtesting), a menudo fallan en el mercado real debido a cambios en las condiciones del mismo. Este fenómeno se conoce como **"deriva estadística"** o **"degradación del rendimiento"**. Detectar esta deriva a tiempo es crucial para proteger el capital.
- **Solución Principal:** AlgoGuardian automatiza la monitorización de esta deriva. Lo hace comparando estadísticamente el rendimiento del backtest con los resultados en tiempo real, detectando las desviaciones significativas para que el trader pueda tomar acciones informadas.
- **Propuesta de Valor:** Proporcionar a los traders una herramienta de gestión de riesgos estadísticamente robusta que actúa como un "guardián" automatizado para sus carteras de estrategias, permitiéndoles operar con mayor confianza.

2. El Usuario y su Flujo de Trabajo (User Journey)

1. **Registro/Inicio de Sesión:** Un nuevo usuario llega a la *Landing Page*, entiende la propuesta de valor, elige un plan y se registra. Un usuario existente inicia sesión.

2. **Creación de un Portafolio:** El usuario crea un "Portafolio" (por ejemplo, "Estrategias de Forex Agresivas"). Este es un contenedor para agrupar estrategias relacionadas.
3. **Carga de Datos:** Dentro de un portafolio, el usuario accede a la sección "Upload". Aquí, sube dos archivos para una nueva estrategia:
    - Un reporte de **Backtest**.
    - Un reporte de **Real-Time**.
    - El sistema es capaz de analizar reportes de **MetaTrader (4/5)** y archivos **CSV/TXT genéricos** que contengan al menos columnas de beneficio ("profit") y fecha de cierre ("close time").
4. **Análisis Automático:** Al subir los archivos, el mockApi (que simula el backend) realiza lo siguiente:
    - Analiza ambos reportes para extraer cada operación individual.
    - Calcula un conjunto completo de **métricas de rendimiento y riesgo** (Net Profit, Max Drawdown, Profit Factor, Winrate, etc.) para ambos periodos.
    - Compara estadísticamente las métricas de Backtest vs. Real-Time, calculando un **p-valor** para cada una, que indica la significancia de la desviación.
    - Crea una curva de PNL combinada.
5. **Visualización y Decisión:**
    - **Vista de Portafolio (PortfolioView):** El usuario ve un resumen de todas sus estrategias, su estado actual (OK, Alerta, Desactivada), y métricas agregadas del portafolio. Destaca aquí la **Matriz de Correlación**, que ayuda a entender la diversificación.
    - **Vista de Detalle (StrategyDetail):** Al hacer clic en una estrategia, el usuario ve:
        - Una comparación visual de las curvas de PNL (Backtest vs. Real-Time).
        - Un **"Drift Score"** (calculado en el frontend) que cuantifica la deriva general.

    - Tarjetas detalladas para cada métrica clave, mostrando la desviación y su significancia estadística.
6. **Configuración y Ajuste:** El usuario puede ir a "Portfolio Settings" para definir **reglas de alerta** personalizadas (ej: "Activar una alerta si el Max Drawdown se desvía más de un 15%"). Estas reglas determinan automáticamente el estado de las estrategias.

3. Modelo de Datos Central (types.ts)

- Portfolio: El contenedor principal. Tiene un name y una lista de metricRules.
- Strategy: Representa una única estrategia de trading. Contiene metadatos (magicNumber, symbol, etc.), una lista de metrics, y los datos de la pnlCurve. Su status (OK, Alert, Deactivated) se determina dinámicamente.
- Metric: Una única métrica de rendimiento (ej: "Profit Factor"). Contiene su name, category, valores de backtestValue y realtimeValue, y el pValue de la comparación.
- MetricRule: Una regla definida por el usuario a nivel de portafolio. Asocia un metricId con umbrales (alertThreshold, deactivationThreshold) que activan los cambios de estado.

4. Lógica de Negocio Clave (services/api.ts)

- **A. Análisis de Reportes (calculateMetrics en api.ts):**
    - Esta es la función más crítica. Detecta el tipo de archivo (reporte de MT4/5 o CSV genérico) y el delimitador.
    - Limpia y normaliza los datos de entrada.
    - Extrae una lista de operaciones individuales (TradeData[]), cada una con su beneficio y fecha.
    - Calcula una serie de KPIs a partir de esta lista de operaciones. Es fundamental entender que todos los análisis se basan en estos cálculos.
- **B. Detección de Drift y Sistema de Estatus:**
    - El "drift" se cuantifica con un **p-valor**. En la mockApi, este se simula, pero en un sistema real se usaría una prueba estadística (como una

prueba T o Z). Un pValue < 0.05 indica una desviación estadísticamente significativa.

- El status de una estrategia en PortfolioView se recalcula en el frontend. Se itera sobre las metricRules del portafolio y se compara la desviación porcentual de cada métrica con los umbrales definidos en la regla.

- **C. Análisis a Nivel de Portafolio (PortfolioView.tsx):**
  - **Curva de Equity Agregada:** Combina las curvas de PNL en tiempo real de todas las estrategias para mostrar el rendimiento total del portafolio.
  - **Matriz de Correlación (calculateCorrelationMatrix):** Una función muy importante. Calcula los retornos diarios de cada estrategia y luego computa la correlación de Pearson entre cada par de estrategias. Un valor alto (cercano a 1) significa que las estrategias se comportan de manera similar (alto riesgo), mientras que un valor bajo o negativo indica una buena diversificación.

5. Arquitectura del Frontend (React)

- **Gestión de Estado:** El estado principal se maneja en el componente App.tsx usando useState y useEffect. Esto incluye la autenticación, la lista de portafolios, el portafolio seleccionado, las estrategias actuales y el estado de las ventanas modales.
- **Estructura de Componentes:**
  - App.tsx: Orquestador principal.
  - LandingPage: Contiene todos los componentes de marketing. Es una mini-app en sí misma.
  - Sidebar.tsx: Navegación principal entre portafolios y vistas.
  - PortfolioView.tsx: La vista principal post-autenticación. Contiene la lógica de las pestañas ("Overview", "Strategies", "Upload", "Settings").
    - El componente PortfolioOverview es complejo y contiene múltiples gráficos de la librería recharts.
  - StrategyDetail.tsx: Vista de análisis profundo para una sola estrategia.
  - Modal.tsx: Componente genérico para todas las ventanas emergentes (autenticación, confirmación de borrado, ajustes).

6. Guía de Estilo y UX/UI

- **Tema:** Oscuro (bg-gray-900), con texto claro (text-gray-100).
- **Color Primario:** Una gama de azules (primary-400 a primary-600) utilizada para botones, enlaces y elementos destacados.
- **Fuente:** Inter.
- **Iconos:** Se utiliza un conjunto de iconos SVG personalizados en Icons.tsx.
- **Experiencia de Usuario (UX):** La interfaz es limpia, orientada a datos y busca presentar información compleja de manera visual e intuitiva (gráficos, tarjetas, indicadores de estado). Las animaciones son sutiles (fade-in, slide-in) para mejorar la fluidez sin distraer.

7. Resumen de Tecnologías

- **Framework:** React
- **Lenguaje:** TypeScript
- **Estilos:** Tailwind CSS
- **Gráficos:** Recharts