Auditoría Técnica y Hoja de Ruta Estratégica: Proyecto Caria

Introducción al Análisis

Este documento presenta una auditoría técnica y un plan de remediación para el proyecto "Caria". El análisis previo identificó una desconexión crítica entre los componentes de *backend* (escritos en Python y lanzados desde services) y la interfaz de *frontend*. Las funciones principales, incluyendo login, valuacion, model, community forum, ideal portfolio y chat, existen pero no logran integrarse o responder adecuadamente.

El objetivo de este informe es triple:

1. **Remediación Técnica Inmediata:** Diagnosticar las causas raíz de los fallos de conexión y proporcionar ajustes de código prácticos para establecer una comunicación funcional entre servicios (API RESTful) y el módulo de chat (WebSocket).
2. **Validación e Implementación Cuantitativa:** Evaluar la robustez del modelo económico, reformular los criterios del portafolio ideal, implementar un análisis de portafolio de nivel profesional e integrar proyecciones gráficas Monte Carlo para validar las valuaciones.
3. **Hoja de Ruta Estratégica:** Definir una arquitectura de sistema escalable y establecer *benchmarks* de experiencia de usuario (UX) para asegurar que el proyecto pueda evolucionar hacia una plataforma FinTech profesional y robusta.

Este informe está estructurado como una revisión de arquitectura senior, proporcionando diagnóstico, código de remediación y dirección estratégica.

Parte 1: Auditoría Técnica y Remediación del Código de Servicios

La prioridad más alta es resolver los fallos de conectividad que actualmente paralizan la aplicación. El problema se divide en dos categorías distintas: las APIs de servicios (REST) y el

servicio de chat (WebSocket).

1.1. Diagnóstico de Conectividad de Servicios Centrales (API)

El síntoma ("no conectan") para los servicios de login, valuacion, modelo y portafolio apunta a una arquitectura de *frontend* de JavaScript que intenta comunicarse con un *backend* de Python (probablemente Flask o Django).[1] En este escenario, los fallos de conexión son casi universalmente atribuibles a dos problemas concurrentes que deben resolverse.

Identificación del Problema #1: Fallo de CORS (Cross-Origin Resource Sharing)

Cuando el *frontend* (ej. http://localhost:3000) y el *backend* (ej. http://localhost:5000) operan en puertos diferentes, el navegador bloquea las solicitudes por razones de seguridad bajo la política de "mismo origen". Esto no es un error en el código de Python; es una característica de seguridad del navegador que el servidor debe anular explícitamente.

Solución (Código Backend - Python/Flask):
La implementación de flask-cors es la solución estándar. Esta biblioteca gestionará la respuesta de "preflight" (OPTIONS) del navegador y añadirá las cabeceras Access-Control-Allow-Origin necesarias a todas las respuestas.
Código de Ajuste (backend.py / services/run.py) [3]:

Python

```
from flask import Flask, jsonify
from flask_cors import CORS  # 1. Importar la biblioteca

#... otras importaciones...

app = Flask(__name__)

# 2. Habilitar CORS para toda la aplicación.
# Para producción, se puede restringir a dominios específicos:
# CORS(app, resources={r"/api/*": {"origins": "https://www.caria.com"}})
```

```
CORS(app)

@app.route("/api/login", methods=)
def handle_login():
    #... lógica de autenticación...
    # (El frontend ahora podrá recibir esta respuesta)
    return jsonify({"status": "success", "token": "jwt_token_aqui"})

@app.route("/api/valuation", methods=)
def handle_valuation():
    #... lógica de valuación...
    return jsonify({"company": "AAPL", "value": 180.50})

#... resto de los endpoints...

if __name__ == "__main__":
    # Asegurarse de que el host esté en 0.0.0.0 para aceptar conexiones externas
    # (no solo localhost) si se usa en contenedores.
    app.run(host='0.0.0.0', port=5000)
```

Identificación del Problema #2: Llamadas de Frontend Malformadas (Errores 404)

Incluso con CORS habilitado, las llamadas desde el *frontend* fallarán si la URL del *endpoint* es incorrecta. Un error común es usar rutas relativas (ej. /api/login) o URLs sin protocolo (ej. localhost:5000/api/login), lo que resulta en errores 404 Not Found en la consola del navegador.

Solución (Código Frontend - JavaScript):
Las llamadas fetch o axios deben usar la URL absoluta y completa del servicio de backend.
Código de Ajuste (frontend/api.js) [3]:

JavaScript

```
// Definir la base de la API en un solo lugar (ej. en configs)
const API_BASE_URL = "http://localhost:5000";

async function fetchValuation(companyTicker) {
```

```
 try {
   // INCORRECTO: const url = "/api/valuation";
   // INCORRECTO: const url = "localhost:5000/api/valuation";
   // CORRECTO:
   const url = `${API_BASE_URL}/api/valuation?ticker=${companyTicker}`;

   const response = await fetch(url, {
     method: 'GET',
     headers: {
       'Content-Type': 'application/json',
       // 'Authorization': `Bearer ${your_jwt_token}` // (Añadir después de que el login funcione)
     }
   });

   if (!response.ok) {
     // Capturar errores del servidor (4xx, 5xx) que no son errores de red
     // Esto es crucial para la depuración [4, 5]
     const errorData = await response.json();
     console.error(`Error del API: ${response.status}`, errorData);
     throw new Error(`HTTP error! status: ${response.status}`);
   }

   const data = await response.json();
   console.log("Conexión con 'valuacion' exitosa:", data);
   return data;

 } catch (error) {
   // Esto captura fallos de red (ej. CORS, servidor caído, DNS)
   console.error("Fallo al conectar con el servicio 'valuacion':", error);
 }
}
```

La siguiente tabla debe usarse como un registro de depuración metódico para verificar cada servicio.

Tabla 1: Registro de Auditoría de Endpoints de API

Servicio	Endpoint (Ruta)	Estado Consola Navegador (Error)	Estado Log Servidor (Error)	Acción de Remediación

Login	/api/login	CORS policy...	N/A	Habilitar flask-cors en el *backend*.
Valuación	/api/valuation	404 Not Found	N/A (no llega la petición)	Corregir URL en fetch (Frontend) a http://localhost:5000/api/valuation.
Modelo	/api/model/vision	500 Internal Server Error	AttributeError: 'NoneType' object...	El código de Python en el *backend* está fallando. Depurar la lógica del modelo.
Foro	/api/forum/posts	200 OK (pero response.json() falla)	N/A	El *backend* no está devolviendo un JSON válido. Asegurar return jsonify(...).
Portafolio	/api/portfolio/ideal	403 Forbidden	Auth Error: No token	La ruta está protegida. La llamada fetch (Frontend) debe incluir el JWT en la cabecera Authorization.

1.2. Depuración y Resolución del Servicio de Chat (WebSocket)

El hecho de que el chat "no funcione" es un problema significativamente más complejo que el de las APIs REST. A diferencia de las solicitudes REST sin estado, los WebSockets son conexiones persistentes y con estado.[6] Un fallo aquí suele ser una cascada de tres problemas distintos.

Problema #1: Fallo de Autenticación en el Handshake (Error 401/403)

El *frontend* no puede simplemente "abrir" un WebSocket. La conexión WebSocket comienza como una solicitud HTTP Upgrade.[6] Si el servicio de login está funcionando, el chat debe estar autenticado. La causa más probable del fallo inicial es que esta solicitud de Upgrade no incluye el token de autenticación (ej. JWT) que el *backend* espera, resultando en un rechazo 401 Unauthorized.[7]

Solución (Código Frontend y Backend):
El cliente debe transmitir el token JWT durante el handshake inicial. El servidor debe tener una función authenticate para validarlo.7
Código de Ajuste (Frontend - ej. con socket.io-client):

JavaScript

```
import { io } from "socket.io-client";

// Obtener el token guardado después del login
const jwtToken = localStorage.getItem("user_token");

// Conectar al servidor de chat, pasando el token en el handshake
const socket = io("http://localhost:5000", { // Asumiendo que el chat está en el mismo servidor
  auth: {
    token: jwtToken // Enviar el token para la autenticación
  }
});

socket.on("connect_error", (err) => {
  // Esto capturará el error 401 si el token es inválido
  console.error("Fallo de autenticación de WebSocket:", err.message);
```

```
});

socket.on("connect", () => {
  console.log("Chat conectado y autenticado con éxito.");
});
```

En el Backend (ej. Flask-SocketIO), se debe verificar este token en el evento on_connect.

Problema #2: Caídas de Conexión por Inactividad (Timeouts de Proxy)

Un problema común que coincide con la descripción (funciona en localhost pero falla en el servidor Dev/QA) [8] es que los *proxies* intermedios (Nginx, ELB) tienen temporizadores de inactividad. Si no se envía ningún dato durante 60 segundos, el *proxy* cerrará la conexión silenciosamente.

Solución (Código Backend y Frontend):
La biblioteca WebSocket debe configurarse para enviar un mecanismo de "heartbeat" (Ping/Pong) para mantener la conexión activa.
Código de Ajuste (Backend - ej. Flask-SocketIO):

Python

```
# Habilitar el heartbeat de PING/PONG
# El cliente (socket.io-client) responderá automáticamente al ping del servidor.
socketio = SocketIO(app, cors_allowed_origins="*",
                    ping_timeout=60, # El servidor espera 60s por un PONG
                    ping_interval=25) # El servidor envía un PING cada 25s
```

Problema #3: Pérdida de Mensajes en la Reconexión (Fallo de UX Crítico)

Este es el fallo más sutil y perjudicial para la experiencia del usuario.[7]

- *Escenario:* El usuario está en el chat y entra en un túnel (pierde la red por 30 segundos).
- *Mientras tanto:* El agente (u otro usuario) envía 3 mensajes.

- *Resultado:* El cliente se reconecta automáticamente al WebSocket. Recibirá el mensaje #4 en adelante, pero los 3 mensajes enviados mientras estaba desconectado se pierden para siempre.[7] Para el usuario, el chat está "roto" y no es confiable.

Solución (Código Frontend):
La reconexión de WebSocket no recupera el historial perdido. En el evento de reconexión (connect), el cliente debe realizar una llamada API REST separada para solicitar al servidor todos los mensajes desde la última marca de tiempo que el cliente recibió.7
Código de Ajuste (Frontend - ej. con socket.io-client):

JavaScript

```
//... (código de conexión del Problema #1)...

// Variable global para rastrear el último mensaje
let lastMessageTimestamp = 0;

// Escuchar el evento 'connect' (que también se dispara en la reconexión)
socket.on("connect", () => {
  console.log("WebSocket conectado. Verificando mensajes perdidos...");
  // LLAMAR A LA API REST PARA OBTENER EL HISTORIAL PERDIDO
  fetchMissingMessages();
});

socket.on("chat_message", (message) => {
  // Renderizar el mensaje...
  renderMessage(message);
  // Actualizar la última marca de tiempo
  lastMessageTimestamp = message.timestamp;
});

async function fetchMissingMessages() {
  // Esta lógica de 'getTranscript' es crucial
  try {
    const response = await
fetch(`${API_BASE_URL}/api/chat/history?since=${lastMessageTimestamp}`, {
      headers: { 'Authorization': `Bearer ${jwtToken}` }
    });

    if (!response.ok) throw new Error("No se pudo obtener el historial del chat");
```

```
    const missingMessages = await response.json(); // Debería ser una lista de mensajes

    if (missingMessages.length > 0) {
      console.log(`Recuperados ${missingMessages.length} mensajes perdidos.`);
      // Renderizar los mensajes perdidos en la UI, ordenados
      renderMissingMessages(missingMessages);
      // Actualizar la marca de tiempo al último mensaje recuperado
      lastMessageTimestamp = missingMessages[missingMessages.length - 1].timestamp;
    }
  } catch (error) {
    console.error("Error al recuperar el historial del chat:", error);
  }
}
```

Tabla 2: Checklist de Diagnóstico de WebSocket

Ítem de Prueba	Diagnóstico / Pregunta	Estado (Fallo/Éxito)	Solución Propuesta
Handshake	¿La solicitud HTTP Upgrade devuelve el código 101 Switching Protocols?	**Fallo (401)**	El *frontend* debe pasar el JWT en el *handshake* (Problema #1).[7]
Persistencia	¿La conexión se cae después de 60 segundos de inactividad?	**Fallo**	Habilitar el ping_interval (Ping/Pong) en el *backend* (Problema #2).[8]
Recuperación	¿Se pierden mensajes si el cliente se desconecta y reconecta?	**Fallo**	Implementar getTranscript / fetchMissingMessages en el evento on connect del *frontend* (Problema #3).[7]

Parte 2: Revisión y Validación del Modelo Cuantitativo

Con los servicios de comunicación restaurados, el enfoque se traslada a la calidad y eficacia de los módulos cuantitativos.

2.1. Evaluación del Modelo de "Visión Económica"

Se solicita una evaluación de la "eficiencia o significancia estadística" del modelo económico. Esto requiere un proceso de validación riguroso (backtesting) y una interpretación honesta de las métricas estadísticas.[9]

Metodología de Validación: Backtesting y Benchmarking

1. **Backtesting:** Este proceso mide la precisión del modelo comparando sus predicciones pasadas con los resultados reales observados.[9] Si el modelo predice "expansión" o "recesión", se debe recopilar un historial de estas predicciones y compararlas con un indicador de la verdad (ej. datos del NBER). Se requiere un mínimo de 30, e idealmente más de 100, puntos de datos (predicciones) para comenzar a inferir la significancia estadística.[11]
2. **Benchmarking:** El rendimiento del modelo debe medirse contra un modelo "challenger" o de referencia.[9] ¿El modelo supera a un modelo simple de media móvil, o incluso a una estrategia de "comprar y mantener" (buy-and-hold)?

Interpretación de Métricas: El Dilema del P-value y R-cuadrado

En los modelos financieros, es extremadamente común y esperado encontrar modelos que tengan **P-values bajos (significativos)** pero **R-cuadrados (R-squared) muy bajos**.[12]

- **P-value (Significancia):** Responde a la pregunta: ¿Es probable que la relación observada entre el predictor (ej. "tasas de interés") y el resultado (ej. "crecimiento del PIB") sea producto del azar?.[13] Un P-value bajo (ej. < 0.05) sugiere que la relación es estadísticamente significativa; es *real*.[12]

- **R-cuadrado (Bondad de Ajuste):** Responde: ¿Qué porcentaje de la variación en el resultado es explicado por mi modelo?.[14] Un R-cuadrado del 10% significa que el 90% de la variación se debe a otros factores (ruido, variables omitidas).

Es fundamental comunicar esto correctamente al usuario final. Un R-cuadrado bajo *no* invalida el modelo si los P-values son significativos.[12] Significa que el modelo ha identificado un factor de tendencia *real*, pero que las predicciones individuales serán "ruidosas" y no precisas.

Cómo Reportar la "Visión" [12]:

- **NO DECIR:** "Nuestro modelo predice que el S&P 500 estará en 5,100 el próximo mes." (Esto requiere un R-cuadrado alto, que es imposible).
- **SÍ DECIR:** "Nuestro modelo ha identificado una relación estadísticamente significativa (P-value < 0.05) entre [Indicador X] y el rendimiento del mercado. Actualmente, este indicador sugiere una *tendencia promedio* alcista (o bajista). Sin embargo, esta tendencia explica solo una pequeña parte de la variabilidad diaria (R-cuadrado bajo), por lo que la precisión de cualquier predicción puntual es limitada."

Tabla 3: Métricas de Validación del Modelo Económico

Métrica	Pregunta que Responde	Resultado Aceptable (Finanzas)	Cómo Reportarlo al Usuario
P-value	¿Es esta relación real o solo suerte? [13]	< 0.05	"Hemos encontrado un factor que está *genuinamente* relacionado con el resultado." [12]
R-cuadrado	¿Cuánto del resultado explica mi modelo? [14]	A menudo bajo (ej. 5%-20%)	"El factor que encontramos es solo una pieza del rompecabezas. La predicción es *imprecisa*." [12]
Backtest (Brier / ROC)	¿Qué tan precisas fueron las predicciones	Mejor que el azar (ROC > 0.5)	"Históricamente, el modelo ha acertado en la

	pasadas? [9]		dirección el X% de las veces."

2.2. Reformulación de los Criterios del "Portafolio Ideal"

La solicitud de que el "portafolio ideal" se base en la "visión económica" es la definición de **Asignación Táctica de Activos (TAA)**.[15] En lugar de un portafolio estático, se crea un portafolio macro-condicional que se ajusta a los "regímenes" económicos identificados por el modelo.[16]

La "visión" del modelo (de 2.1) debe ser destilada en señales de régimen claras. Por ejemplo:

- **Régimen de Alto Riesgo:** Señalado por un VIX > 20 [16] o una alta dispersión en los pronósticos macroeconómicos.
- **Régimen de Bajo Riesgo:** Señalado por un VIX < 20.

Una estrategia de inversión dinámica utiliza estos indicadores para ajustar la asignación entre activos de riesgo (ej. un índice de mercado como el IPC) y activos de renta fija (ej. CETES).[17] La estrategia se basa en ajustar la exposición para mitigar el riesgo ante caídas anticipadas por el ciclo económico.[17]

Tabla 4: Reglas de Portafolio Macro-Condicional (Ejemplo de Implementación)

Señal del Modelo (T5)	Condición del Indicador	Asignación de Portafolio (Acciones)	Asignación de Portafolio (Bonos)	Justificación
Alto Riesgo	VIX > 20 O Modelo_Caria == "Recesión"	30%	70%	Mitigación de riesgo; los bonos (renta fija) proveen estabilidad.[17]
Neutral	VIX entre 15-20 Y Modelo_Caria == "Neutral"	60%	40%	Portafolio balanceado estándar.

Bajo Riesgo	VIX < 15 Y Modelo_Caria == "Expansión"	90%	10%	Maximizar ganancias durante condiciones de mercado favorables.[16]

2.3. Confiabilidad y Mejora del Módulo de Valuación

La solicitud de "confiabilidad" en la valuación y la de "proyecciones Monte Carlo" están intrínsecamente ligadas. La confiabilidad de un modelo de valuación (ej. Flujo de Caja Descontado - DCF) no es una propiedad estática; es el resultado de un proceso de prueba riguroso.[10]

Un modelo DCF simple no es confiable porque se basa en *suposiciones* puntuales (ej. "crecimiento de ingresos del 10%"). La confiabilidad se establece probando el impacto de esas suposiciones a través de:

1. **Análisis de Sensibilidad:** Evalúa cómo cambia la valuación final (output) cuando un solo *input* (ej. tasa de descuento) se ajusta.[18]
2. **Análisis de Escenarios:** Evalúa la valuación bajo 3-5 futuros distintos (ej. Pesimista, Base, Optimista).[18]
3. **Simulación Monte Carlo:** Es el análisis de escenarios más avanzado. En lugar de 3 escenarios, simula 10,000+ escenarios [19] donde los *inputs* (crecimiento, márgenes) se seleccionan aleatoriamente de una distribución de probabilidad.

Por lo tanto, la implementación de la simulación Monte Carlo (solicitada en T8) es la *respuesta* directa a la solicitud de confiabilidad (T7). Transforma una valuación de una suposición singular a una distribución de probabilidad, permitiendo al usuario decir: "Hay un 75% de probabilidad de que el valor de esta empresa esté por encima de X".

Parte 3: Implementación de Nuevas Funcionalidades Cuantitativas

Esta sección proporciona el "cómo" técnico para las dos nuevas características solicitadas:

Monte Carlo y el script de análisis de portafolio.

3.1. Incorporación de Proyecciones Gráficas Monte Carlo

La solicitud es implementar una simulación Monte Carlo para proyecciones de portafolio. La arquitectura correcta para esto es:

1. **Backend (Python):** Ejecutar las N simulaciones (ej. 10,000) usando numpy y pandas.[20] La función perform_monte_carlo [21] es un excelente punto de partida, ya que modela el crecimiento del portafolio, calcula los percentiles y devuelve los resultados.
2. **Frontend (JavaScript):** Recibir los resultados del *backend* y visualizarlos usando una biblioteca interactiva. Plotly.js es ideal porque puede integrarse directamente con React [22] y manejar los datos JSON generados por Plotly-Python.[23]

Peligro de Rendimiento en la Visualización:
El intento de trazar 10,000 simulaciones como 10,000 trazas de Plotly (líneas separadas) colapsará el navegador del cliente.25
Solución de Rendimiento (Técnica de Optimización):
La única forma eficiente de visualizar miles de líneas es concatenarlas todas en una sola traza, usando np.nan como separador entre cada simulación. Además, se debe usar Scattergl (que utiliza WebGL, acelerado por GPU) en lugar del Scatter estándar (que usa SVG).25
Código de Implementación (Backend - Python/Plotly para generar el gráfico) [25]:

Python

```
import numpy as np
import plotly.graph_objs as go

def generate_monte_carlo_plot(simulations_data):
    """
    Genera una figura Plotly optimizada para N simulaciones.
    'simulations_data' debe ser una lista de listas o un array 2D (N_simulaciones x N_pasos).
    """

    N = len(simulations_data) # ej. 10000 simulaciones
    M = len(simulations_data) # ej. 100 pasos (años/días)
```

```
    # 1. Crear los arrays X e Y para las N líneas
    # Replicar el eje X (0 a M-1) para cada simulación
    all_xs = np.tile(np.arange(M, dtype='float64'), N)

    # Aplanar todos los datos Y en un solo array
    all_ys = np.array(simulations_data, dtype='float64').flatten()

    # 2. Crear el array de 'nan' para la separación
    # Necesitamos un 'nan' después de cada línea, así que N 'nan's en total
    nan_separator = np.array([np.nan] * N)

    # 3. Concatenar los arrays
    # Intercalar los datos con 'nan's
    # (Esto requiere una reorganización más compleja que la simple concatenación)

    # Enfoque más simple de :
    all_xs_list = [np.arange(M, dtype='float64') for _ in range(N)]
    all_ys_list = simulations_data # Asumir que ya es una lista de listas/arrays

    # Añadir 'nan' al final de CADA segmento
    all_xs_with_nan = [np.concatenate((xs, [np.nan])) for xs in all_xs_list]
    all_ys_with_nan = [np.concatenate((ys, [np.nan])) for ys in all_ys_list]

    # Concatenar todos los segmentos en una sola línea gigante
    xs_final = np.concatenate(all_xs_with_nan)
    ys_final = np.concatenate(all_ys_with_nan)

    # 4. Crear la figura usando go.Scattergl (optimizado por GPU)
    fig = go.Figure(data=)

    fig.update_layout(
        title="Proyección de Portafolio (Simulación Monte Carlo)",
        xaxis_title="Período (Años)",
        yaxis_title="Valor del Portafolio ($)",
        showlegend=False
    )

    # 5. Convertir la figura a JSON para enviarla al frontend
    # El frontend (Plotly.js) puede renderizar este JSON directamente.
    return fig.to_json()
```

3.2. Script de Análisis de Portafolio del Usuario (Métricas Profesionales)

Se solicita un script para el análisis de portafolio con métricas profesionales (Sharpe, Sortino, Alpha, Beta, Max Drawdown).

Solución de Implementación:
No es necesario reinventar este script. La biblioteca quantstats 26 está diseñada para este propósito exacto. Es superior a otras (como empyrical 27) para esta tarea porque puede descargar datos de benchmark (como 'SPY' para el S&P 500) y generar un informe HTML completo de "hoja de desgarro" (tearsheet) con una sola línea de código.26
Código de Implementación (Backend - Python/QuantStats) [26]:

Python

```python
import quantstats as qs
import pandas as pd
# Asumiendo que yfinance está instalado (quantstats lo usa)
import yfinance as yf

def generate_professional_portfolio_report(user_returns_series, benchmark_ticker='SPY'):
    """
    Genera un informe profesional de análisis de portafolio y devuelve la ruta del archivo.

    :param user_returns_series: Una serie de Pandas de retornos diarios (ej. 0.01, -0.005,...)
    :param benchmark_ticker: El ticker del benchmark (ej. 'SPY' para S&P 500)
    :return: Ruta al archivo HTML generado.
    """

    try:
        # Asegurar que los retornos del usuario sean una serie de pandas
        if not isinstance(user_returns_series, pd.Series):
            raise ValueError("user_returns_series debe ser una pd.Series")

        # Ponerle nombre a la serie (quantstats lo usa en el informe)
        user_returns_series.name = "Portafolio Caria"
```

```
        # 1. Extender pandas con las funciones de quantstats
        qs.extend_pandas()

        # 2. Descargar los datos del benchmark
        # (qs.utils.download_returns es una envoltura para yfinance)
        benchmark_returns = qs.utils.download_returns(benchmark_ticker)

        # 3. Generar el informe HTML completo
        # Esto calcula automáticamente Sharpe, Sortino, Max Drawdown, Calmar,
        # Alpha, Beta, y genera gráficos de rentabilidad.
        output_filename = 'report/user_portfolio_analysis.html'

        qs.reports.html(
            user_returns_series,
            benchmark=benchmark_returns,
            output=output_filename,
            title='Análisis de Portafolio del Usuario (vs. SPY)'
        )

        print(f"Informe profesional generado en {output_filename}")

        # 4. (Opcional) Calcular métricas individuales [29]
        metrics = {
            'sharpe': user_returns_series.sharpe(),
            'sortino': user_returns_series.sortino(),
            'max_drawdown': user_returns_series.max_drawdown(),
            'alpha': user_returns_series.alpha(benchmark=benchmark_returns),
            'beta': user_returns_series.beta(benchmark=benchmark_returns),
            'cagr': user_returns_series.cagr() # Compound Annual Growth Rate
        }

        print("Métricas individuales calculadas:", metrics)

        return output_filename, metrics

    except Exception as e:
        print(f"Error al generar el informe de portafolio: {e}")
        return None, None

# --- Ejemplo de uso ---
# (Esto sería llamado por el endpoint de la API)
# sim_returns = pd.Series(np.random.normal(0.0005, 0.01, 1000))
# generate_professional_portfolio_report(sim_returns, 'SPY')
```

Para que estas métricas sean útiles, el usuario final debe entender qué significan.

Tabla 5: Definición de Métricas de Portafolio para el Usuario Final

Métrica	Pregunta que Responde	Interpretación Sencilla
Sharpe Ratio	¿Cuánta rentabilidad obtuve por cada unidad de riesgo (volatilidad) que tomé?	Más alto es mejor. Mide la calidad de la rentabilidad ajustada al riesgo total.
Sortino Ratio	¿Cuánta rentabilidad obtuve por cada unidad de riesgo *malo* (volatilidad a la baja)? [29]	Más alto es mejor. Es más inteligente que Sharpe, ya que no penaliza la volatilidad "buena" (hacia arriba).
Max Drawdown	¿Cuál fue la peor caída de pico a valle que experimentó mi portafolio? [30]	Más bajo (cercano a 0) es mejor. Mide el dolor máximo que el inversor habría sufrido.
Alpha	¿Mi portafolio superó (o no) al mercado (ej. SPY), después de ajustar por el riesgo? [30]	Positivo es bueno (agregó valor). Negativo es malo (destruyó valor).
Beta	¿Qué tan volátil es mi portafolio *en relación* con el mercado (SPY)? [29]	Beta = 1.0 (se mueve con el mercado). Beta > 1.0 (más volátil). Beta < 1.0 (menos volátil).

Parte 4: Hoja de Ruta Estratégica para Escalabilidad y Experiencia de Usuario (UX)

Con los módulos cuantitativos y de comunicación en funcionamiento, la visión debe ampliarse

hacia la sostenibilidad a largo plazo (escalabilidad) y la adopción del usuario (UX).

4.1. Recomendaciones de Arquitectura para la Escalabilidad

La solicitud de un sitio "profesional" con "escalabilidad" es una preocupación arquitectónica fundamental. En el mundo FinTech, la escalabilidad no se refiere solo a más usuarios, sino a [33]:

- **Rendimiento de Transacciones:** Capacidad de manejar miles de operaciones concurrentes sin latencia.
- **Velocidad de Onboarding/KYC:** Procesamiento rápido de nuevos usuarios.
- **Fiabilidad y Tolerancia a Fallos:** Garantizar que las operaciones (ej. transferencias) no se dupliquen ni se pierdan.[33]
- **Cumplimiento Global:** Adherencia a regulaciones como PCI DSS y GDPR.

La industria FinTech favorece las arquitecturas de **Microservicios** [35] porque permiten escalar componentes individuales (ej. el servicio de 'Pagos' puede escalar independientemente del servicio de 'Foro').[33]

Sin embargo, dado el estado actual del proyecto (un desarrollador principal luchando con la integración de servicios básicos), adoptar una arquitectura de microservicios *ahora* sería un error estratégico. Aumentaría exponencialmente la complejidad operativa (despliegue, monitoreo, comunicación entre servicios).

La recomendación estratégica es adoptar un **Monolito Modular** [33]:

1. **Mantener una Base de Código Única:** Continuar con la aplicación Flask/Python como un solo servicio desplegable (un Monolito).
2. **Diseño Basado en Dominios (Modular):** *Internamente*, estructurar el código con límites estrictos entre dominios (ej. carpetas separadas para identidad, portafolios, pagos, social), como si fueran servicios separados.[33]
3. **Persistencia Políglota:** No usar una sola base de datos para todo.[37] Usar SQL (ej. PostgreSQL) para datos transaccionales (crítico) y una base de datos NoSQL (ej. Redis) para datos volátiles (ej. sesiones de chat, caché de precios).
4. **APIs Idempotentes:** Un concepto crítico en FinTech. Asegurar que las APIs que mueven dinero (ej. POST /api/transfer) puedan ser llamadas múltiples veces (debido a reintentos de red) pero ejecuten la transacción solo *una vez*.[33]

Esta arquitectura de Monolito Modular permite velocidad de desarrollo *ahora*, mientras se mantiene una ruta clara para extraer estos módulos en microservicios *en el futuro* cuando la escala lo exija.

Tabla 6: Comparativa de Arquitecturas para "Caria" (Etapa Actual)

Criterio	Monolito Modular (Recomendado)	Microservicios (prematuro)
Velocidad de Desarrollo	**Alta.** Base de código única, depuración simplificada.	**Baja.** Requiere gestión de N servicios, RPC, descubrimiento de servicios.
Complejidad Operacional	**Baja.** Un solo despliegue, un solo servidor.	**Muy Alta.** Requiere orquestación (Kubernetes), monitoreo complejo.
Escalabilidad	**Media.** Escala verticalmente (mejor máquina) o todo el monolito horizontalmente.	**Alta.** Escala componentes individuales (ej. solo el chat).
Resiliencia	**Baja.** Un fallo en el 'Foro' puede tumbar los 'Pagos'.	**Alta.** Un fallo en el 'Foro' no afecta a los 'Pagos'.
Conclusión	**Óptimo para la etapa actual.** Prioriza la entrega de producto sobre la escala granular.	**Incorrecto para la etapa actual.** Introduce complejidad que matará el proyecto.

4.2. Estrategias para una Experiencia de Usuario (UX) Profesional

Una "buena experiencia para los usuarios" en FinTech se reduce a dos elementos: **Confianza** y **Velocidad**.[38] La expectativa del consumidor en 2024/2025 es de "cero fricción".[40] El 86% de los usuarios de banca digital considerará cambiar de proveedor si la experiencia digital no cumple con sus expectativas.[38]

Un hallazgo clave [41] es que FinTech tiene la tasa de finalización de *onboarding* (proceso de registro) más alta de todas las industrias (24.5%). Esto no es porque a los usuarios les gusten los formularios, sino porque las "altas apuestas" (manejar su dinero) los motivan a completar

el proceso. La estrategia no es eliminar pasos, sino hacer que cada paso sea fácil y genere confianza.

Recomendaciones Accionables (Benchmarks):

1. **Onboarding (Registro):**
   - **Benchmark Objetivo:** Apuntar al estándar de Chime (Neobanco de EE.UU.): **4.5 minutos para completar el onboarding**.[39]
   - **Divulgación Progresiva:** No pedir todo (KYC, documentos, etc.) de inmediato.[39] El registro inicial debe ser mínimo (ej. Teléfono + OTP).[40] Los documentos de KYC solo deben solicitarse cuando el usuario intente realizar una acción que lo requiera (ej. "Depositar Fondos").
   - **Autenticación Moderna:** Usar Teléfono + OTP (One-Time Password) como estándar sobre email/contraseña. Habilitar el inicio de sesión biométrico (Face ID, huella digital) desde el primer día.[40]
   - **Mostrar Progreso:** Usar siempre una barra de progreso visible durante el *onboarding* para que el usuario sepa cuántos pasos faltan.[40]
2. **Viajes del Usuario (User Journeys):**
   - Medir el rendimiento de cada tarea (ej. "abrir una cuenta", "vender una acción") en **clics y segundos**.[38] El objetivo no es tener cero clics, sino que cada clic sea simple, comprensible y sin esfuerzo.[38]

Tabla 7: Benchmarks de UX FinTech (2024/2025)

Característica	Métrica	Benchmark / Mejor Práctica de la Industria
Onboarding	Tiempo de Finalización	**4.5 minutos** (Benchmark de Chime) [39]
Login	Método	Teléfono + OTP; Biometría (Face ID) [40]
Formularios KYC	Diseño	**Divulgación Progresiva** (Pedir datos solo cuando se necesiten) [39]
Medición de Tareas	KPI	Número de clics y segundos por "viaje del usuario" [38]

Conclusiones y Próximos Pasos

El proyecto "Caria" se encuentra en un punto de inflexión crítico. Los problemas de conectividad, aunque parecen graves, son resolubles y comunes en el desarrollo full-stack. La implementación de las correcciones de código para CORS, las llamadas de API y, fundamentalmente, la lógica de recuperación de historial del WebSocket, restaurará la funcionalidad básica.

Las recomendaciones estratégicas son las siguientes:

1. **Acciones Inmediatas (Arreglar):**
    - Implementar flask-cors en el *backend* (Sección 1.1).
    - Corregir todas las llamadas fetch del *frontend* para usar URLs absolutas (Sección 1.1).
    - Implementar la solución de 3 partes para el chat (Autenticación JWT, Ping/Pong, y Recuperación de Historial REST) (Sección 1.2).
2. **Acciones a Corto Plazo (Validar y Construir):**
    - Integrar la biblioteca quantstats (Sección 3.2) para implementar *inmediatamente* el análisis de portafolio profesional.
    - Implementar la visualización optimizada de Monte Carlo (Scattergl con nan) (Sección 3.1).
    - Utilizar el framework de la Tabla 3 para validar estadísticamente el modelo de "visión económica".
    - Implementar el portafolio ideal usando las reglas de régimen de la Tabla 4.
3. **Filosofía a Largo Plazo (Estrategia):**
    - Comprometerse con la arquitectura de **Monolito Modular** (Sección 4.1). Resistir la tentación de los microservicios hasta que el equipo y los ingresos crezcan.
    - Hacer de la UX una métrica clave. Medir el tiempo de *onboarding* y apuntar al *benchmark* de **4.5 minutos** (Sección 4.2).

Obras citadas

1. Frontend and Backend Integration with RESTful APIs in Python and Django, fecha de acceso: noviembre 13, 2025, https://dev.to/jacobisah/frontend-and-backend-integration-with-restful-apis-in-python-and-django-15d6
2. Python Backend Development: A Complete Guide for Beginners - DataCamp, fecha de acceso: noviembre 13, 2025, https://www.datacamp.com/tutorial/python-backend-development
3. How to connect backend (python, flask) with frontend (html, css ..., fecha de acceso: noviembre 13, 2025,

https://stackoverflow.com/questions/55549164/how-to-connect-backend-python-flask-with-frontend-html-css-javascript
4. WebSocket Debugging: Keeping Real-Time Apps Running - PixelFreeStudio Blog, fecha de acceso: noviembre 13, 2025, https://blog.pixelfreestudio.com/websocket-debugging-keeping-real-time-apps-running/
5. Solucionar problemas con el widget de comunicaciones de Amazon ..., fecha de acceso: noviembre 13, 2025, https://docs.aws.amazon.com/es_es/connect/latest/adminguide/ts-cw.html
6. ¿Cómo resolver el problema de las conexiones websocket que se pierden en una aplicación de chat? (Golang - Reddit, fecha de acceso: noviembre 13, 2025, https://www.reddit.com/r/golang/comments/whwm5w/how_to_solve_websocket_connections_losing_issue/?tl=es-es
7. Performance Testing: Benchmarking Vs. Back-Testing - RiskSpan, fecha de acceso: noviembre 13, 2025, https://riskspan.com/performance-testing-benchmarking-vs-back-testing/
8. Evaluación De La Precisión Y Confiabilidad De Los Modelos Financieros - FasterCapital, fecha de acceso: noviembre 13, 2025, https://fastercapital.com/es/tema/evaluaci%C3%B3n-de-la-precisi%C3%B3n-y-confiabilidad-de-los-modelos-financieros.html/1
9. How Many Trades Are Enough? A Guide to Statistical Significance in Backtesting - Medium, fecha de acceso: noviembre 13, 2025, https://medium.com/@trading.dude/how-many-trades-are-enough-a-guide-to-statistical-significance-in-backtesting-093c2eac6f05
10. How to Interpret Regression Models that have Significant Variables ..., fecha de acceso: noviembre 13, 2025, https://statisticsbyjim.com/regression/low-r-squared-regression/
11. How to interpret the p-value, R^2, and adjusted R^2 in your models | Green Belt Six Sigma, fecha de acceso: noviembre 13, 2025, https://www.youtube.com/watch?v=DXanv1kHiiI
12. Análisis de Regresión: ¿Cómo Puedo Interpretar el R-cuadrado y Evaluar la Bondad de Ajuste? - Minitab Blog, fecha de acceso: noviembre 13, 2025, https://blog.minitab.com/es/blog/analisis-de-regresion-como-puedo-interpretar-el-r-cuadrado-y-evaluar-la-bondad-de-ajuste
13. Portfolio Optimization with Sector Return Prediction Models - MDPI, fecha de acceso: noviembre 13, 2025, https://www.mdpi.com/1911-8074/17/6/254
14. How to Build Portfolios with Macro-Conditional Market ... - LSEG, fecha de acceso: noviembre 13, 2025, https://www.lseg.com/content/dam/data-analytics/en_us/documents/white-papers/re2518092-fathom-building-portfolios-whitepaper.pdf
15. Leading macroeconomic indicators for a dynamic investment strategy, fecha de acceso: noviembre 13, 2025, https://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S2007-07052022000100210&lng=es&nrm=iso
16. Valuation & Financial Modeling: Guide, Methods, Examples, fecha de acceso:

noviembre 13, 2025, https://growthequityinterviewguide.com/private-equity/valuation-and-financial-modeling
17. Python Monte Carlo for Beginners: Investment Risk Analysis Made Easy - YouTube, fecha de acceso: noviembre 13, 2025, https://www.youtube.com/watch?v=BGLt_aSrMKc
18. Introduction to Monte Carlo Simulation in Python - Hayes H - Medium, fecha de acceso: noviembre 13, 2025, https://hhundman.medium.com/introduction-to-monte-carlo-simulation-in-python-d3e21efbd7e6
19. How to build and visualise a Monte Carlo simulation with Python ..., fecha de acceso: noviembre 13, 2025, https://www.shedloadofcode.com/blog/how-to-build-and-visualise-a-monte-carlo-simulation-with-python-and-plotly/
20. React plotly.js in JavaScript, fecha de acceso: noviembre 13, 2025, https://plotly.com/javascript/react/
21. Plotly JavaScript Open Source Graphing Library, fecha de acceso: noviembre 13, 2025, https://plotly.com/javascript/
22. D3.js vs Plotly: Which JavaScript Visualization Library Should You Choose? - Medium, fecha de acceso: noviembre 13, 2025, https://medium.com/@ebojacky/d3-js-vs-plotly-which-javascript-visualization-library-should-you-choose-dbf8ad67321f
23. Monte Carlo Plot - Plotly Python - Plotly Community Forum, fecha de acceso: noviembre 13, 2025, https://community.plotly.com/t/monte-carlo-plot/16225
24. ranaroussi/quantstats: Portfolio analytics for quants, written ... - GitHub, fecha de acceso: noviembre 13, 2025, https://github.com/ranaroussi/quantstats
25. Empyrical Module for Risk and Performance Metrics | by DolphinDB - Medium, fecha de acceso: noviembre 13, 2025, https://medium.com/@DolphinDB_Inc/empyrical-module-for-risk-and-performance-metrics-e1a3847758c8
26. quantopian/empyrical: Common financial risk and performance metrics. Used by zipline and pyfolio. - GitHub, fecha de acceso: noviembre 13, 2025, https://github.com/quantopian/empyrical
27. Build a Free Stock Screener in Python with Beta, Sharpe, and Sortino Ratios, fecha de acceso: noviembre 13, 2025, https://python.plainenglish.io/build-a-free-stock-screener-in-python-with-beta-sharpe-and-sortino-ratios-139eeb1a909b
28. How to Quickly Backtest a Portfolio with Python and Empyrical | by Intrinio | Medium, fecha de acceso: noviembre 13, 2025, https://intrinio.medium.com/how-to-quickly-backtest-a-portfolio-with-python-fac34229f5e9
29. How to Do Portfolio Analytics in Python (Amazing 1400% Return) - Quant Science, fecha de acceso: noviembre 13, 2025, https://quantscience.io/newsletter/b/portfolio-analytics-in-python
30. Sharpe, Sortino and Calmar Ratios with Python | Codearmo, fecha de acceso:

noviembre 13, 2025, https://www.codearmo.com/blog/sharpe-sortino-and-calmar-ratios-python
31. How to Build a Scalable FinTech App in 5 Smart Stages - ProCreator, fecha de acceso: noviembre 13, 2025, https://procreator.design/blog/build-scalable-fintech-app-smart-stages/
32. Building a Fintech App in 2025: Best Tech Stacks and Architecture Choices, fecha de acceso: noviembre 13, 2025, https://dev.to/lucas_wade_0596/building-a-fintech-app-in-2025-best-tech-stacks-and-architecture-choices-4n85
33. How to Build a Fintech App: Approach, Architecture, and Scalability - MobiDev, fecha de acceso: noviembre 13, 2025, https://mobidev.biz/blog/how-to-build-fintech-app-approach-architecture-scalability
34. Building a Scalable Fintech App: A Deep-Dive Case Study - Luminoguru, fecha de acceso: noviembre 13, 2025, https://luminoguru.com/blog/scalable-fintech-app-a-deep-dive-case-study/
35. Scalable Architecture Patterns for High-Growth Startups That Every Business Owner Should Know Today - Full Scale, fecha de acceso: noviembre 13, 2025, https://fullscale.io/blog/scalable-architecture-patterns/
36. Learning from your competitors: How FinTech Insights measures UX, fecha de acceso: noviembre 13, 2025, https://www.fintechinsights.io/blog/methodology-for-ux
37. Fintech Website Onboarding Best Practices With 4 Useful Examples - CleverTap, fecha de acceso: noviembre 13, 2025, https://clevertap.com/blog/onboarding-best-practices-for-fintech/
38. 10 Best Fintech UX Practices for Mobile Apps in 2025 - ProCreator Design, fecha de acceso: noviembre 13, 2025, https://procreator.design/blog/best-fintech-ux-practices-for-mobile-apps/
39. Customer Onboarding Checklist Completion Rate: 2024 Benchmark Report - Userpilot, fecha de acceso: noviembre 13, 2025, https://userpilot.com/blog/onboarding-checklist-completion-rate-benchmarks/
40. App Onboarding Guide - Top 10 Onboarding Flow Examples 2025 - UXCam, fecha de acceso: noviembre 13, 2025, https://uxcam.com/blog/10-apps-with-great-user-onboarding/