# **Informe Técnico de Referencia: Aceleración del Desarrollo de Backend para un Bot de Arbitraje de CEDEARs**

## **Sección 1: Arquitectura Fundacional y Bibliotecas Centrales**

Esta sección establece la base tecnológica para el Producto Mínimo Viable (MVP), mapeando los componentes conceptuales del documento del proyecto a bibliotecas de código abierto específicas y validadas. Proporciona una pila tecnológica clara y defendible que equilibra el desarrollo rápido para el MVP con una visión hacia la escalabilidad futura.

### **1.1 El Motor: Un Marco de Doble Horizonte para Backtesting y Ejecución**

La elección del motor de trading es la decisión arquitectónica más crítica, ya que dicta la velocidad de desarrollo, la fiabilidad de las pruebas y la facilidad de transición a la operativa en vivo. Se propone un enfoque de dos horizontes: una solución pragmática para el MVP y una ruta de escalabilidad estratégica para el futuro.

#### **1.1.1 Recomendación para el MVP: Backtrader**

Para el desarrollo inicial y la validación del MVP, se recomienda enfáticamente el uso de **Backtrader**. Esta recomendación se fundamenta en su mención explícita en el plan del proyecto , su completo conjunto de características para escribir estrategias, indicadores y analizadores reutilizables 1, y su amplia adopción en la comunidad de código abierto para implementar estrategias de arbitraje estadístico y

*pairs trading*.3

La arquitectura central de Backtrader, denominada "Cerebro", orquesta de manera eficiente todos los componentes del flujo de trabajo de backtesting. Un objeto Cerebro gestiona las fuentes de datos (Data Feeds), las lógicas de trading (Strategies), los cálculos de métricas (Analyzers) y la visualización de resultados (Plotting).2 Esta estructura modular permite a los desarrolladores centrarse en la implementación de la lógica de la estrategia, tal como se define en los requisitos, sin necesidad de construir la infraestructura de simulación desde cero.1 La capacidad de

Backtrader para ingerir datos directamente desde DataFrames de pandas es particularmente ventajosa, ya que se alinea perfectamente con el conjunto de herramientas de análisis de datos que se utilizarán para el motor de señales.6

#### **1.1.2 Planificación a Futuro: NautilusTrader**

Si bien Backtrader es ideal para la creación rápida de prototipos del MVP, es prudente considerar una ruta de escalamiento para futuras iteraciones que puedan requerir un rendimiento ultra alto o una operativa de alta frecuencia. Para este fin, se presenta **NautilusTrader** como una consideración estratégica.7

NautilusTrader es una plataforma de grado de producción cuyo núcleo está escrito en Rust, un lenguaje conocido por su seguridad y rendimiento. Su diseño se centra en una arquitectura puramente orientada a eventos y, fundamentalmente, aborda el "desafío de paridad": mantener una consistencia absoluta entre el entorno de investigación/backtesting en Python y el entorno de ejecución en vivo.7 Esto elimina la necesidad de reescribir la estrategia en un lenguaje de menor nivel (como C++ o Java) para la producción, un problema común en la industria del trading algorítmico.7 Su compatibilidad con una amplia gama de clases de activos y su capacidad para operar en múltiples centros de negociación simultáneamente lo convierten en una opción robusta para las futuras ambiciones del bot, como la conexión a Primary/Matba Rofex o a brokers internacionales.

#### **1.1.3 El Abismo del Backtesting a la Producción: Un Patrón Arquitectónico Clave**

Una de las dificultades más subestimadas en el trading algorítmico es la transición de una estrategia desde un entorno de backtesting simulado a un entorno de trading en vivo.8 Los backtesters, por necesidad, hacen suposiciones simplificadoras. Por ejemplo, una orden límite en una simulación puede asumirse que se ejecuta tan pronto como el precio de mercado toca el precio límite. En la realidad, la ejecución depende de la liquidez disponible en el libro de órdenes, la latencia de la red y la prioridad de la orden. El documento del proyecto especifica una lógica de ejecución compleja, como

cancel/replace para órdenes que no se ejecutan en un plazo determinado, una funcionalidad que no es estándar en la mayoría de los frameworks de backtesting y que requiere una implementación personalizada.

Esta discrepancia entre la simulación y la realidad crea un riesgo operativo significativo. Para mitigar este riesgo desde el inicio, la arquitectura del MVP debe tratar el módulo de ejecución de órdenes (el "broker") como un componente distinto e intercambiable. Este patrón, a menudo denominado "pluggable broker" (broker conectable), es fundamental.

Durante el desarrollo y las pruebas, la estrategia interactuará con un SimulatedBroker que replica el comportamiento del mercado basándose en datos históricos. Para la operativa real, este componente se sustituirá por un LiveIOLBroker, que encapsulará todas las interacciones con la API de InvertirOnline. La lógica central de la estrategia (el cálculo de señales, la gestión de posiciones) permanece idéntica en ambos modos. Lo único que cambia es la implementación subyacente del broker.

Esta separación de responsabilidades no solo mitiga el riesgo de la transición a producción, sino que también alinea la arquitectura del MVP con los principios de diseño robustos de plataformas avanzadas como NautilusTrader 7, facilitando futuras migraciones y la adición de nuevos brokers (adaptadores) sin necesidad de modificar el núcleo estratégico.

### **1.2 El Conjunto de Herramientas Analíticas: pandas, numpy y statsmodels**

El motor de señales, responsable de identificar las oportunidades de arbitraje, dependerá de un conjunto de bibliotecas de Python bien establecidas y probadas en la industria financiera.

* **pandas**: Será la biblioteca fundamental para toda la manipulación de datos de series temporales. Se utilizará para cargar, limpiar, alinear y realizar cálculos sobre los datos de precios y spreads.9 Su estructura  
  DataFrame es el formato de entrada ideal para Backtrader.6
* **numpy**: Proporcionará la base para todos los cálculos numéricos eficientes, especialmente operaciones vectorizadas sobre los arrays de precios y retornos.9
* **statsmodels**: Es la biblioteca clave para los análisis estadísticos requeridos por la estrategia. Específicamente, se utilizará la función statsmodels.tsa.stattools.coint para realizar la prueba de cointegración de Engle-Granger, tal como se especifica en el plan del proyecto.1 Un análisis detallado de esta función, incluyendo la correcta interpretación de sus parámetros (  
  y0, y1, trend) y sus valores de retorno (p-valor, estadístico t), es crucial para una implementación correcta.11

### **1.3 La Línea Vital: Estrategia para el Cliente de la API de IOL**

La conexión con el broker InvertirOnline (IOL) es el único punto de contacto con el mercado real para la ejecución de órdenes y la obtención de datos en tiempo real. La fiabilidad de este componente es, por lo tanto, no negociable.

#### **1.3.1 Análisis de los Wrappers Existentes**

Una investigación de las bibliotecas Python existentes que actúan como *wrappers* para la API de IOL revela un panorama de riesgo. Se han identificado varios proyectos en GitHub, como aairabella/iol-python-api 13,

fernandezpablo85/mcpiol 14 y

iolConn en PyPI.15 Sin embargo, un análisis más profundo muestra un patrón preocupante:

* **Falta de Mantenimiento:** Muchos de estos proyectos parecen ser esfuerzos de un solo desarrollador y no han sido actualizados en años. Por ejemplo, aairabella/iol-python-api no ha tenido una actualización significativa en más de seis años.13
* **Funcionalidad Incompleta:** Algunos wrappers, como iolConn, declaran explícitamente que funcionalidades críticas como el envío de órdenes no están implementadas.15
* **Ausencia de una Biblioteca Oficial:** La propia organización de InvertirOnline en GitHub (invertironlineargentina) proporciona ejemplos de código en C#, pero no una biblioteca oficial de Python mantenida.16

Depender de una biblioteca de terceros no mantenida o incompleta para una función tan crítica como la ejecución de operaciones financieras introduce un riesgo inaceptable en el proyecto. Un cambio en la API de IOL, un bug no resuelto o una vulnerabilidad de seguridad podrían paralizar las operaciones del bot sin previo aviso y sin un soporte claro al que recurrir.

#### **1.3.2 Recomendación Definitiva: Construir un Cliente de API Interno**

Dada la evaluación de riesgos anterior, se recomienda de manera contundente **construir un cliente de API de IOL dedicado y de propiedad interna** para el MVP. Aunque esto representa una inversión inicial de tiempo de desarrollo (estimada en unos pocos días-hombre para un desarrollador experimentado), los beneficios a largo plazo superan con creces el costo:

* **Control Total:** El equipo de desarrollo tendrá control absoluto sobre la implementación, incluyendo la gestión de errores, el manejo de límites de tasa (*rate limiting*), el registro de eventos y la lógica de reintentos.
* **Fiabilidad:** El cliente se construirá directamente sobre la documentación oficial de la API de IOL 17, asegurando que se ajuste con precisión a las necesidades del proyecto para la autenticación (incluida la renovación automática de tokens de acceso y refresco), la obtención de datos de mercado y la gestión completa del ciclo de vida de las órdenes (envío, consulta, cancelación).
* **Mantenibilidad:** El cliente se convierte en un activo central y mantenible del proyecto. Cualquier cambio futuro en la API de IOL puede ser abordado internamente de manera proactiva.
* **Seguridad:** La gestión de credenciales y tokens se manejará según las mejores prácticas de seguridad del proyecto, sin depender de las decisiones de un tercero.

Este cliente de API interno será un módulo Python bien estructurado, probablemente utilizando bibliotecas como requests para las llamadas HTTP síncronas o aiohttp si se contempla una arquitectura asíncrona en el futuro. Se convertirá en la base fiable sobre la que se construirá el LiveIOLBroker mencionado en la sección 1.1.3.

## **Sección 2: Implementación de la Estrategia de Arbitraje Estadístico**

Esta sección sirve como una guía práctica y orientada al código para construir la estrategia de paridad de CEDEARs. Traduce los conceptos teóricos del documento del proyecto en pasos de implementación concretos y reutilizables.

### **2.1 El Pipeline de Datos: Agregación de Entradas de Misión Crítica**

La estrategia de paridad se basa en el cálculo preciso de la siguiente fórmula, como se indica en el documento del proyecto :

Paridad Esperada (ARS)=Psubyacente USD​×ratio×tipo de cambio implıˊcito (CCL)

Para implementar esto, el bot debe obtener y sincronizar cuatro puntos de datos clave de manera fiable y automatizada. La siguiente tabla identifica las fuentes autorizadas para cada entrada de datos y el método de acceso recomendado, proporcionando una hoja de ruta clara para el equipo de desarrollo.

| Punto de Datos | Fuente Autorizada | Método de Acceso / Biblioteca | Referencia | Notas e Implementación Detallada |
| --- | --- | --- | --- | --- |
| **Precio del CEDEAR (ARS)** | API de InvertirOnline | Cliente de API Interno (ver Sec. 1.3) | 1 | Utilizar el endpoint GET /api/cotizaciones/{instrumento}. Este dato es el precio de mercado local del certificado. |
| **Precio del Subyacente (USD)** | API de InvertirOnline | Cliente de API Interno (ver Sec. 1.3) | 1 | La API de IOL proporciona el precio del activo subyacente. Para futuras expansiones, un adaptador a un broker internacional como IBKR ofrecería datos de mayor calidad y menor latencia. |
| **Tipo de Cambio CCL** | DolarAPI | requests o aiohttp | 19 | Utilizar el endpoint GET https://dolarapi.com/v1/dolares/ccl. Se recomienda encapsular esta llamada en un módulo propio con una capa de caché para evitar llamadas excesivas y manejar posibles fallos de la API externa. |
| **Ratio de Conversión** | Banco Comafi / BYMA | pandas (para parsing) + requests | 21 | Las fuentes oficiales publican esta información en archivos .xlsx o PDF. Es crucial automatizar la descarga y el parseo de estos documentos de forma periódica (ej. diariamente). |

#### **2.1.1 La Naturaleza Dinámica de los Datos "Estáticos"**

Un punto crítico que a menudo se pasa por alto es que ciertos datos, como los ratios de conversión de los CEDEARs, no son verdaderamente estáticos. Aunque cambian con poca frecuencia, lo hacen en respuesta a eventos corporativos como *splits* de acciones o cambios en el programa de CEDEARs.23 Un sistema que codifica estos valores de forma fija (

*hardcoding*) es inherentemente frágil y está destinado a fallar, generando señales de trading incorrectas y potenciales pérdidas financieras.

Para abordar esta vulnerabilidad, la arquitectura del bot debe incluir un componente dedicado, que se puede denominar "Data Master" o "trabajador de datos programado". La única responsabilidad de este servicio será:

1. Ejecutarse de manera periódica (por ejemplo, una vez al día antes del inicio del mercado).
2. Descargar la última versión de los documentos oficiales que contienen los ratios de conversión (por ejemplo, el archivo .xlsx de Banco Comafi 25 o los listados de BYMA 22).
3. Utilizar pandas para parsear estos archivos y extraer la tabla de ratios.
4. Actualizar un almacén de datos interno y fiable (que puede ser desde un simple archivo JSON versionado hasta una tabla en una base de datos).

La lógica de trading principal nunca debe acceder directamente a las fuentes externas. En su lugar, leerá los ratios de conversión desde este almacén de datos interno, que actúa como una fuente de verdad única y controlada. Este diseño desacopla la lógica de trading, que es de misión crítica, del proceso de *scraping*, que es inherentemente más frágil y propenso a fallos.

### **2.2 Análisis de Cointegración en Python**

El primer paso para validar un par de activos para la estrategia es confirmar que sus series de precios están cointegradas, lo que indica una relación de equilibrio a largo plazo.1 La prueba de Engle-Granger es el método especificado para esto.

El siguiente fragmento de código muestra cómo implementar esta prueba utilizando statsmodels. Se asume que cedear\_prices y underlying\_prices\_in\_ars son objetos pandas.Series que contienen las series temporales de precios históricos.

Python

import pandas as pd  
from statsmodels.tsa.stattools import coint  
  
# Asumiendo que 'cedear\_prices' y 'underlying\_prices\_usd' son pandas Series.  
# 'ccl\_rate' y 'conversion\_ratio' son los valores actuales.  
# El precio del subyacente debe ser convertido a ARS para la comparación.  
underlying\_prices\_in\_ars = underlying\_prices\_usd \* ccl\_rate / conversion\_ratio  
  
# Realizar la prueba de cointegración de Engle-Granger  
score, pvalue, crit\_values = coint(cedear\_prices, underlying\_prices\_in\_ars, trend='c')  
  
print(f"Resultado de la prueba de cointegración:")  
print(f" Estadístico t: {score:.4f}")  
print(f" P-valor: {pvalue:.4f}")  
print(f" Valores críticos: 1%: {crit\_values:.4f}, 5%: {crit\_values:.4f}, 10%: {crit\_values:.4f}")  
  
# Interpretación del resultado  
# La hipótesis nula es que NO hay cointegración.  
if pvalue < 0.05: # Nivel de significancia del 5%  
 print("\nConclusión: Se rechaza la hipótesis nula. Las series están probablemente cointegradas.")  
 print("El par es un candidato viable para la estrategia de reversión a la media.")  
else:  
 print("\nConclusión: No se puede rechazar la hipótesis nula. Las series no están cointegradas.")  
 print("La estrategia de arbitraje estadístico no es aplicable para este par.")

Es fundamental interpretar correctamente el resultado: la hipótesis nula (H0​) de la prueba coint es que "no hay cointegración".11 Por lo tanto, se busca un p-valor bajo (típicamente menor a 0.05) para poder

*rechazar* la hipótesis nula con un nivel de confianza estadística. Este rechazo proporciona la evidencia de que existe una relación de equilibrio a largo plazo entre las dos series, justificando el uso de una estrategia de reversión a la media. Confiar únicamente en una alta correlación es un error común que puede llevar a relaciones espurias.4

### **2.3 Cálculo del Z-Score y Generación de Señales**

Una vez que un par ha sido validado como cointegrado, el siguiente paso es calcular el *spread* (diferencial) entre sus precios y normalizarlo utilizando un Z-score. El Z-score mide cuántas desviaciones estándar se ha alejado el spread actual de su media histórica móvil, proporcionando una señal normalizada para la entrada y salida de operaciones.

El siguiente código demuestra cómo calcular el Z-score de un spread utilizando pandas.

Python

import pandas as pd  
  
# Calcular el spread. Este es el residuo de la regresión de cointegración,  
# o más simplemente, la diferencia de precios si el ratio de cobertura es 1.  
# spread = cedear\_prices - underlying\_prices\_in\_ars  
# Para un enfoque más riguroso, el spread se calcula a partir de los residuos  
# de una regresión OLS entre los dos activos.  
  
# Supongamos que 'spread' es una pandas.Series con el diferencial histórico.  
window = 60 # Ejemplo de ventana móvil (configurable, p. ej., 60 períodos)  
  
# Calcular la media y la desviación estándar móviles del spread  
spread\_mean = spread.rolling(window=window).mean()  
spread\_std = spread.rolling(window=window).std()  
  
# Calcular el Z-score  
z\_score = (spread - spread\_mean) / spread\_std  
  
# Visualizar los últimos valores del Z-score  
print("Últimos valores del Z-score:")  
print(z\_score.tail())

Este código, inspirado en patrones encontrados en varios repositorios de arbitraje estadístico 9, debe ser encapsulado en una función reutilizable o, idealmente, implementado como un Indicador personalizado dentro de

Backtrader. Esto permite una integración limpia y eficiente en la clase de la estrategia principal, manteniendo el código organizado y legible. Los umbrales para generar señales (por ejemplo, Z>1.5 para vender el spread, Z<−1.5 para comprarlo) se aplicarán directamente sobre la serie z\_score resultante.

### **2.4 Código Base de la Estrategia en Backtrader**

El núcleo del bot de trading es la clase bt.Strategy. El siguiente es un esqueleto de código completo y comentado que integra los componentes anteriores. Sirve como un punto de partida de alta calidad para el equipo de desarrollo, demostrando la estructura y el flujo de una estrategia de *pairs trading* en Backtrader.

Python

import backtrader as bt  
  
class CedearParityStrategy(bt.Strategy):  
 params = (  
 ('z\_entry\_threshold', 1.5), # Umbral de Z-score para entrar en una operación  
 ('z\_exit\_threshold', 0.5), # Umbral de Z-score para salir de una operación  
 ('lookback\_period', 60), # Período para calcular la media/std móvil  
 )  
  
 def \_\_init\_\_(self):  
 # Referencias a las líneas de datos para facilitar el acceso  
 self.cedear = self.datas  
 self.underlying = self.datas  
  
 # Para seguir el estado de las órdenes  
 self.order = None  
  
 # Calcular el spread entre los dos activos  
 # Nota: En una implementación real, los precios deben estar en la misma moneda.  
 # Aquí se asume que los datos ya han sido pre-procesados y alineados.  
 self.spread = self.cedear.close - self.underlying.close  
  
 # Calcular el Z-score del spread  
 spread\_mean = bt.indicators.SimpleMovingAverage(self.spread, period=self.p.lookback\_period)  
 spread\_std = bt.indicators.StandardDeviation(self.spread, period=self.p.lookback\_period)  
 self.zscore = (self.spread - spread\_mean) / spread\_std  
  
 def notify\_order(self, order):  
 if order.status in:  
 # Orden enviada/aceptada por el broker  
 return  
  
 if order.status in [order.Completed]:  
 if order.isbuy():  
 self.log(f'COMPRA EJECUTADA, Precio: {order.executed.price:.2f}, Costo: {order.executed.value:.2f}, Comisión: {order.executed.comm:.2f}')  
 elif order.issell():  
 self.log(f'VENTA EJECUTADA, Precio: {order.executed.price:.2f}, Costo: {order.executed.value:.2f}, Comisión: {order.executed.comm:.2f}')  
   
 elif order.status in:  
 self.log('Orden Cancelada/Margen/Rechazada')  
  
 # Permitir que se envíen nuevas órdenes  
 self.order = None  
  
 def log(self, txt, dt=None):  
 ''' Función de logging para la estrategia '''  
 dt = dt or self.datas.datetime.date(0)  
 print(f'{dt.isoformat()} {txt}')  
  
 def next(self):  
 # Comprobar si hay una orden pendiente  
 if self.order:  
 return  
  
 # Comprobar si no tenemos una posición abierta  
 if not self.position:  
 if self.zscore > self.p.z\_entry\_threshold:  
 # El spread es demasiado alto, vender el spread  
 # Vender el CEDEAR (sobrevalorado) y comprar el subyacente (infravalorado)  
 self.log(f'SEÑAL VENTA SPREAD: Z-score {self.zscore:.2f} > {self.p.z\_entry\_threshold}')  
 self.order = self.sell(data=self.cedear)  
 self.order = self.buy(data=self.underlying)  
   
 elif self.zscore < -self.p.z\_entry\_threshold:  
 # El spread es demasiado bajo, comprar el spread  
 # Comprar el CEDEAR (infravalorado) y vender el subyacente (sobrevalorado)  
 self.log(f'SEÑAL COMPRA SPREAD: Z-score {self.zscore:.2f} < {-self.p.z\_entry\_threshold}')  
 self.order = self.buy(data=self.cedear)  
 self.order = self.sell(data=self.underlying)  
 else:  
 # Ya estamos en el mercado, buscar señal de salida (reversión a la media)  
 if self.position.size > 0: # Si estamos largos en el spread (compramos CEDEAR)  
 if self.zscore >= -self.p.z\_exit\_threshold:  
 self.log(f'SEÑAL CIERRE LARGO: Z-score {self.zscore:.2f} >= {-self.p.z\_exit\_threshold}')  
 self.close(data=self.cedear)  
 self.close(data=self.underlying)  
 else: # Si estamos cortos en el spread (vendimos CEDEAR)  
 if self.zscore <= self.p.z\_exit\_threshold:  
 self.log(f'SEÑAL CIERRE CORTO: Z-score {self.zscore:.2f} <= {self.p.z\_exit\_threshold}')  
 self.close(data=self.cedear)  
 self.close(data=self.underlying)

Este código base encapsula la lógica principal. El método \_\_init\_\_ prepara los indicadores necesarios (spread y z-score). El método next contiene la lógica de decisión para cada barra de datos: comprueba si hay una orden activa, si hay una posición abierta y si el Z-score ha cruzado los umbrales de entrada o salida definidos en los parámetros. Finalmente, notify\_order gestiona el ciclo de vida de las órdenes, registrando su estado y liberando el bloqueo para permitir nuevas operaciones.

## **Sección 3: Ejecución, Riesgo y Simulación en la Práctica**

Esta sección aborda los aspectos mecánicos de la implementación, pasando de la generación de señales teóricas a la colocación práctica de órdenes, la gestión de riesgos y la garantía de que las simulaciones sean realistas y fiables.

### **3.1 Gestión Avanzada de Órdenes en Backtrader**

La forma en que se ejecutan las órdenes es tan importante como la señal que las genera. Una ejecución deficiente puede convertir una estrategia teóricamente rentable en una perdedora. El documento del proyecto especifica requisitos claros para minimizar el deslizamiento (*slippage*) y gestionar órdenes no ejecutadas.

#### **3.1.1 Implementación de Órdenes Límite**

Para mitigar el riesgo de que una orden se ejecute a un precio peor que el esperado (deslizamiento), especialmente en mercados volátiles, es imperativo utilizar órdenes límite en lugar de órdenes a mercado. Backtrader facilita esto a través del parámetro exectype en los métodos buy y sell.

El siguiente ejemplo muestra cómo emitir una orden de compra límite a un precio ligeramente por debajo del precio de cierre actual:

Python

# Dentro del método next() de la estrategia  
if self.zscore < -self.p.z\_entry\_threshold:  
 # Calcular el precio límite deseado  
 # Por ejemplo, el precio de cierre actual como límite superior  
 limit\_price = self.cedear.close  
   
 self.log(f'Creando orden de COMPRA LÍMITE para CEDEAR a {limit\_price:.2f}')  
   
 # Enviar la orden de compra límite  
 self.order = self.buy(  
 data=self.cedear,  
 exectype=bt.Order.Limit,  
 price=limit\_price  
 )

La documentación de Backtrader detalla cómo el motor de simulación maneja estas órdenes, intentando replicar un comportamiento realista. Por ejemplo, una orden de compra límite se ejecutará si el precio de apertura de la siguiente barra ya es inferior al precio límite, o si el precio mínimo (low) de la barra alcanza el precio límite.27

#### **3.1.2 Gestión del Ciclo de Vida de las Órdenes con notify\_order**

El método notify\_order es el centro de control para el seguimiento de las órdenes. Es fundamental implementar una lógica robusta aquí para reaccionar a los diferentes estados que una orden puede tener a lo largo de su vida.

Python

def notify\_order(self, order):  
 if order.status == order.Submitted or order.status == order.Accepted:  
 # La orden ha sido enviada o aceptada, no se requiere acción.  
 self.log(f'Orden {order.ref} Aceptada/Enviada')  
 return  
  
 if order.status == order.Completed:  
 # La orden se ha completado con éxito.  
 if order.isbuy():  
 self.log(f'COMPRA EJECUTADA: {order.executed.size} a {order.executed.price:.2f}')  
 else: # Es una venta  
 self.log(f'VENTA EJECUTADA: {order.executed.size} a {order.executed.price:.2f}')  
 self.order = None # Liberar para permitir nuevas órdenes  
  
 elif order.status in:  
 # La orden fue cancelada, rechazada o no se pudo ejecutar por falta de margen.  
 self.log(f'Orden {order.ref} Cancelada/Rechazada/Margen')  
 self.order = None # Liberar para permitir nuevas órdenes  
  
 elif order.status == order.Expired:  
 # La orden ha expirado (si se usó un parámetro de validez).  
 self.log(f'Orden {order.ref} Expirada')  
 self.order = None # Liberar para permitir nuevas órdenes

Este manejo detallado del estado es esencial. Por ejemplo, al establecer self.order = None solo después de que una orden se complete, cancele o rechace, se evita el envío de múltiples órdenes mientras una ya está activa en el mercado.5

#### **3.1.3 Implementación de la Lógica de Cancelar/Reemplazar**

El requisito de "cancelar y reemplazar" una orden si no se ejecuta en un plazo determinado es una táctica de ejecución avanzada. Backtrader no ofrece una orden Cancel/Replace nativa, pero esta lógica se puede construir implementando una máquina de estados simple dentro de la estrategia.

Este patrón requiere que la estrategia realice un seguimiento no solo de la orden activa, sino también del momento en que se creó.

1. **En el método \_\_init\_\_**, se añade una variable para almacenar la barra de creación de la orden:  
   Python  
   self.order\_creation\_bar = 0
2. **En el método next()**, al crear una orden, se almacena la barra actual:  
   Python  
   #... al crear una orden...  
   self.order = self.buy(...)  
   self.order\_creation\_bar = len(self)
3. **La lógica de next() se expande** para comprobar el tiempo de vida de la orden pendiente:  
   Python  
   # Dentro de next()  
   if self.order: # Si hay una orden pendiente  
    # Definir un timeout, por ejemplo, 3 barras  
    timeout\_bars = 3  
    if len(self) > (self.order\_creation\_bar + timeout\_bars):  
    self.log(f'La orden {self.order.ref} ha superado el timeout. Cancelando.')  
    self.cancel(self.order)  
    return # No hacer nada más mientras se espera
4. **En notify\_order()**, cuando se confirma la cancelación (order.Canceled), self.order se establece en None. Esto permitirá que en la siguiente llamada a next(), si la señal de trading persiste, se pueda emitir una nueva orden, potencialmente con un precio ajustado (un *tick* menos favorable para aumentar la probabilidad de ejecución).

Este patrón implementa eficazmente la lógica cancel/replace requerida, haciendo que el bot sea más adaptable a las condiciones de liquidez del mercado.

### **3.2 Dimensionamiento de Posiciones y Controles de Riesgo**

Una gestión de riesgos sólida es lo que diferencia a un sistema de trading sostenible de un experimento peligroso. El documento del proyecto especifica varios controles de riesgo clave.

#### **3.2.1 Implementación de un Sizer Personalizado**

Backtrader permite desacoplar la lógica de dimensionamiento de la posición de la lógica de la señal a través de los Sizers. Para implementar un requisito como "limitar el tamaño de la posición al 5% del capital", se debe crear un Sizer personalizado.

Python

import backtrader as bt  
  
class PercentageSizer(bt.Sizer):  
 params = (  
 ('percents', 5), # Porcentaje del capital a arriesgar por operación  
 )  
  
 def \_getsizing(self, comminfo, cash, data, isbuy):  
 # Calcular el tamaño de la posición basado en un porcentaje del valor del portafolio  
 port\_value = self.broker.getvalue()  
 size\_in\_value = port\_value \* (self.p.percents / 100.0)  
   
 # Convertir el valor monetario al número de acciones/contratos  
 price = data.close  
 if price == 0:  
 return 0 # Evitar división por cero  
   
 return size\_in\_value / price

Este Sizer se puede añadir al motor Cerebro con cerebro.addsizer(PercentageSizer, percents=5), aplicando esta regla de dimensionamiento a todas las operaciones.28

#### **3.2.2 Implementación de Controles de Riesgo en la Estrategia**

Otros controles de riesgo, como los límites de pérdida diaria y los *circuit breakers*, deben implementarse directamente en la lógica de la estrategia. La siguiente tabla traduce los requisitos del proyecto en código concreto.

| Control de Riesgo (de ) | Método de Implementación | Componente Backtrader | Lógica de Código / Pseudocódigo |
| --- | --- | --- | --- |
| **Dimensionamiento de Posición (% del capital)** | Sizer Personalizado | bt.Sizer | class PercentageSizer(bt.Sizer):... (como se muestra arriba) |
| **Límite de Pérdida Diaria** | Lógica de Estrategia | bt.Strategy | En \_\_init\_\_: self.daily\_start\_value = self.broker.getvalue(). En next(): if self.broker.getvalue() - self.daily\_start\_value < -max\_daily\_loss: self.stop\_trading = True. Se necesita lógica adicional para resetear daily\_start\_value cada día. |
| **Circuit Breaker (Drawdown Máximo)** | Lógica de Estrategia / Analizador | bt.Strategy / bt.Analyzer | En \_\_init\_\_: self.portfolio\_peak = self.broker.getvalue(). En next(): self.portfolio\_peak = max(self.portfolio\_peak, self.broker.getvalue()). if self.broker.getvalue() < self.portfolio\_peak \* (1 - max\_drawdown\_pct): self.stop\_trading = True. |
| **Interruptor DRY-RUN / REAL** | Configuración Global | Cerebro / params de Estrategia | En el script principal: if config == 'LIVE': cerebro.setbroker(IOLBroker(...)) else: cerebro.broker.set\_cash(...). |

### **3.3 Metodologías de Backtesting Robustas**

Un backtest que produce resultados espectaculares a menudo es una señal de sobreajuste (*overfitting*) o de suposiciones poco realistas, no de una "gallina de los huevos de oro".30 Para generar confianza en la estrategia, el proceso de backtesting debe ser riguroso.

1. **Incorporar Fricciones del Mercado:** El backtest debe ser lo más realista posible. Esto implica configurar el broker de Backtrader para que tenga en cuenta los costos que erosionan la rentabilidad:
   * **Comisiones:** Utilizar cerebro.broker.setcommission(commission=0.005) (por ejemplo, 0.5%) para simular las comisiones de compra/venta.
   * **Deslizamiento (Slippage):** Simular el deslizamiento en órdenes a mercado con cerebro.broker.set\_slippage\_perc(perc=0.001) (por ejemplo, 0.1%).
   * **Spread Bid-Ask:** Aunque más complejo de simular con datos OHLC, se puede añadir un costo fijo adicional a cada operación para representar el spread.
2. **Combatir el Sobreajuste:** El sobreajuste ocurre cuando una estrategia se optimiza tanto para un conjunto de datos históricos que pierde su capacidad de generalizar a datos nuevos. El documento del proyecto menciona explícitamente la necesidad de evitarlo. Las técnicas clave son:
   * **Pruebas Fuera de Muestra (Out-of-Sample):** Dividir los datos históricos en un período de entrenamiento (para desarrollar y optimizar la estrategia) y un período de prueba fuera de muestra (para validar su rendimiento en datos no vistos).
   * **Análisis Walk-Forward (Ventanas Móviles):** Este es un método más avanzado y robusto. La estrategia se optimiza en una ventana de datos (ej. 2 años) y luego se prueba en la siguiente ventana (ej. 6 meses). Este proceso se repite, deslizando las ventanas a través de todo el conjunto de datos. Esto simula de manera más realista cómo se adaptaría la estrategia a las condiciones cambiantes del mercado.

La implementación de estas metodologías robustas es esencial para evaluar si el rendimiento observado de la estrategia es genuino o simplemente un artefacto del proceso de prueba.

## **Sección 4: Recomendaciones Accionables y Hoja de Ruta de Desarrollo**

Esta sección final sintetiza todos los hallazgos en un plan de implementación estratégico y por fases, diseñado para guiar al equipo de desarrollo desde el concepto hasta la producción.

### **4.1 Plan de Implementación del Backend del MVP (Enfoque por Fases)**

Se propone un enfoque de desarrollo iterativo en cuatro fases para construir el núcleo de trading del bot. Este enfoque prioriza la construcción de una base sólida y la mitigación de riesgos en cada etapa.

* **Fase 1: Cimientos de Datos y Conectividad (Duración estimada: 1-2 semanas)**
  1. **Desarrollar el Cliente de API Interno para IOL:** Siguiendo la recomendación de la Sección 1.3, la primera tarea es construir un módulo Python robusto y bien probado que encapsule todas las interacciones con la API de IOL. Esto debe incluir la gestión de autenticación (tokens de acceso y refresco), endpoints para obtener cotizaciones (/api/cotizaciones), consultar el portafolio (/api/micuenta/miportafolio) y enviar/cancelar órdenes (/api/operar/acciones).1
  2. **Implementar el "Data Master":** Crear el servicio o script programado (descrito en la Sección 2.1.1) para descargar y parsear automáticamente los ratios de conversión de CEDEARs desde las fuentes oficiales (Comafi, BYMA).22
  3. **Implementar el Cliente de CCL:** Crear un pequeño módulo para consultar la API dolarapi.com y obtener el tipo de cambio CCL, con lógica de caché y manejo de errores.19
* **Fase 2: Estrategia y Backtesting Riguroso (Duración estimada: 2-3 semanas)**
  1. **Implementar la Estrategia en Backtrader:** Utilizando el código base de la Sección 2.4, implementar la lógica completa de la estrategia CedearParityStrategy. Esto incluye la ingesta de datos históricos (que se pueden obtener a través de la API de IOL o fuentes externas), el cálculo del spread y el Z-score como indicadores personalizados.
  2. **Realizar Backtesting Exhaustivo:** Ejecutar la estrategia sobre un amplio conjunto de datos históricos para múltiples pares de CEDEARs. Configurar el broker de Backtrader con comisiones y deslizamiento realistas.
  3. **Análisis de Sensibilidad y Optimización:** Analizar cómo varía el rendimiento de la estrategia al cambiar parámetros clave (umbrales de Z-score, período de *lookback*). Realizar pruebas de robustez utilizando metodologías de walk-forward para evitar el sobreajuste.
* **Fase 3: Gestión de Riesgos y Ejecución en Sandbox (Duración estimada: 1-2 semanas)**
  1. **Desarrollar el Sizer Personalizado:** Implementar la clase PercentageSizer (Sección 3.2.1) para gestionar el tamaño de las posiciones según el riesgo definido.
  2. **Integrar Controles de Riesgo en la Estrategia:** Añadir la lógica para los límites de pérdida diaria y los *circuit breakers* por *drawdown* directamente en la clase de la estrategia.
  3. **Construir y Probar el Módulo de Ejecución en Vivo:** Desarrollar el LiveIOLBroker que utiliza el cliente de API interno. Implementar la lógica de cancel/replace. Probar exhaustivamente este módulo conectándolo al **entorno de sandbox de IOL**.1 El interruptor DRY-RUN/REAL debe estar firmemente en modo DRY-RUN (sandbox).
* **Fase 4: Puesta en Producción y Monitorización (Duración estimada: 1 semana)**
  1. **Orquestación y Despliegue:** Empaquetar la aplicación y desplegarla utilizando un orquestador ligero como Airflow o n8n, como se sugiere en el plan del proyecto.
  2. **Configurar Observabilidad:** Implementar logging estructurado para registrar cada evento, señal y orden. Exportar métricas clave (PnL, número de operaciones, deslizamiento) a un sistema de monitorización como Prometheus y visualizarlas en un dashboard de Grafana.
  3. **Activación Controlada:** Tras una validación final y exhaustiva en el entorno de sandbox, activar el modo de trading real. Se recomienda comenzar con un capital limitado para observar el comportamiento del bot en condiciones de mercado reales antes de escalar la inversión.

### **4.2 Estructura de Proyecto Recomendada**

Una estructura de directorios clara y lógica es fundamental para la mantenibilidad y escalabilidad del proyecto. Se recomienda la siguiente organización:

bot\_arbitraje\_mvp/  
├── main.py # Punto de entrada principal de la aplicación  
├── config.yaml # Archivo de configuración (claves de API, umbrales, parámetros)  
├── core/  
│ ├── engine.py # Configuración y ejecución del motor Cerebro de Backtrader  
│ ├── strategies/  
│ │ └── cedear\_parity.py # Clase de la estrategia principal  
│ └── sizers/  
│ └── percentage\_sizer.py # Sizer de posición personalizado  
├── data/  
│ ├── iol\_client.py # Cliente de API interno para InvertirOnline  
│ ├── ccl\_client.py # Cliente para la API de cotización del dólar CCL  
│ └── data\_master.py # Script para la actualización de ratios de conversión  
├── execution/  
│ └── iol\_broker.py # Implementación del broker en vivo para IOL  
├── logs/ # Directorio para los archivos de log estructurados  
├── tests/ # Pruebas unitarias y de integración para todos los módulos  
└── requirements.txt # Dependencias del proyecto Python

Esta estructura separa claramente las responsabilidades: core contiene la lógica de trading pura, data se encarga de la conectividad con fuentes externas, execution maneja la interacción con el broker en vivo, y config centraliza todos los parámetros configurables.

### **4.3 Diseño para la Escalabilidad Futura**

La arquitectura propuesta para el MVP no solo cumple con los requisitos inmediatos, sino que también sienta las bases para el crecimiento futuro delineado en el roadmap del proyecto.

* **Adición de Nuevos Brokers:** El patrón de "pluggable broker" (Sección 1.1.3) es clave para la escalabilidad. Para añadir soporte para Primary/Matba Rofex o Interactive Brokers (IBKR), solo se necesita crear un nuevo módulo de broker (ej. primary\_broker.py) que implemente la misma interfaz que iol\_broker.py. La lógica de la estrategia en core/strategies no necesitará ninguna modificación, lo que acelera enormemente el desarrollo de nuevas integraciones.
* **Modelos de Señal Avanzados:** La modularidad del diseño permite que el motor de señales sea fácilmente reemplazable. Si en el futuro se decide explorar modelos de *machine learning* (como clustering de spreads o redes neuronales recurrentes), se puede desarrollar un nuevo generador de señales que produzca las mismas salidas (señales de compra/venta/cierre) que el actual motor basado en Z-score. Este nuevo motor se puede integrar sin afectar la infraestructura de datos, ejecución y gestión de riesgos ya construida.
* **Integración con la Capa Agéntica:** La clara separación del núcleo de trading del resto del sistema es fundamental para la arquitectura MCP/RAG propuesta. El núcleo de trading, con sus componentes de ejecución y consulta de estado, puede ser expuesto como una "herramienta" a través de una API interna (por ejemplo, REST o gRPC). La capa agéntica podrá entonces invocar esta herramienta para realizar acciones como "obtener las señales actuales" o "ejecutar una operación simulada", alineándose perfectamente con la visión de un sistema orquestado por un modelo de lenguaje.1

En conclusión, este informe proporciona una hoja de ruta técnica detallada y accionable. Al seguir las recomendaciones arquitectónicas, las implementaciones de código de referencia y el plan por fases, el equipo de desarrollo puede acelerar significativamente la construcción de un backend robusto, fiable y escalable para el MVP del bot de trading.

#### Obras citadas

1. Bot arbitraje BMB Argentina.docx
2. The ML4T Workflow: From Model to Strategy Backtesting - Machine Learning for Trading, fecha de acceso: agosto 13, 2025, <https://www.ml4trading.io/chapter/7>
3. Statistical Arbitrage in the Cryptocurrency Market: Strategies, Best Practices, and Key Statistical Methods using Python - GOKE ADEKUNLE; #Wolfwords, fecha de acceso: agosto 13, 2025, <https://thisgoke.medium.com/statistical-arbitrage-in-the-cryptocurrency-market-strategies-best-practices-and-key-statistical-42e7c719ad8f>
4. arikaufman/algorithmicTrading: Experimenting with Algo Trading using Backtrader Python Module. Specifically, statistical arbitrage using cointegration. - GitHub, fecha de acceso: agosto 13, 2025, <https://github.com/arikaufman/algorithmicTrading>
5. Backtrader: What it is, How to Install, Strategies, Trading and More - QuantInsti Blog, fecha de acceso: agosto 13, 2025, <https://blog.quantinsti.com/backtrader/>
6. Backtrader: Welcome, fecha de acceso: agosto 13, 2025, <https://www.backtrader.com/>
7. nautechsystems/nautilus\_trader: A high-performance algorithmic trading platform and event-driven backtester - GitHub, fecha de acceso: agosto 13, 2025, <https://github.com/nautechsystems/nautilus_trader>
8. livealgos/README.md at master - GitHub, fecha de acceso: agosto 13, 2025, <https://github.com/impulsecorp/livealgos/blob/master/README.md>
9. nirajdsouza/statistical-arbitage-strategy: Simple Python program to understand the basics of Statistical Arbitrage - GitHub, fecha de acceso: agosto 13, 2025, <https://github.com/nirajdsouza/statistical-arbitage-strategy>
10. Preprocesar y analizar rendimientos bursátiles con Python | by Jesús López | es.datons, fecha de acceso: agosto 13, 2025, <https://medium.com/datons-stories-es/preprocesar-y-analizar-rendimientos-burs%C3%A1tiles-con-python-4af0ecf14742>
11. statsmodels.tsa.stattools.coint - statsmodels 0.14.4, fecha de acceso: agosto 13, 2025, <https://www.statsmodels.org/stable/generated/statsmodels.tsa.stattools.coint.html>
12. statsmodels.tsa.stattools.coint, fecha de acceso: agosto 13, 2025, <https://www.statsmodels.org/0.8.0/generated/statsmodels.tsa.stattools.coint.html>
13. aairabella/iol-python-api: API Python3 para InvertirOnline.com - GitHub, fecha de acceso: agosto 13, 2025, <https://github.com/aairabella/iol-python-api>
14. fernandezpablo85/mcpiol: MCP for the IOL public API - GitHub, fecha de acceso: agosto 13, 2025, <https://github.com/fernandezpablo85/mcpiol>
15. iolConn · PyPI, fecha de acceso: agosto 13, 2025, <https://pypi.org/project/iolConn/>
16. IOL invertironline - GitHub, fecha de acceso: agosto 13, 2025, <https://github.com/invertironlineargentina>
17. API | IOL invertironline, fecha de acceso: agosto 13, 2025, <https://www.invertironline.com/api>
18. Documentación API | IOL invertironline, fecha de acceso: agosto 13, 2025, <https://www.invertironline.com/documentacion-api>
19. Dólares Ámbito - DolarApi.com, fecha de acceso: agosto 13, 2025, <https://dolarapi.com/docs/argentina/ambito/>
20. Dólar Blue | DolarApi.com, fecha de acceso: agosto 13, 2025, <https://dolarapi.com/docs/argentina/operations/get-dolar-blue.html>
21. Banco Comafi | CEDEARS, fecha de acceso: agosto 13, 2025, <https://www.comafi.com.ar/custodiaglobal/programas.aspx>
22. CEDEARs Negociables en BYMA, fecha de acceso: agosto 13, 2025, <https://data-widgets.byma.com.ar/wp-content/uploads/dlm_uploads/2019/09/BYMA-Tabla-CEDEARs-2025-01-15.pdf>
23. Más oportunidades para invertir en CEDEAR - Guía de Ayuda - Inicio, fecha de acceso: agosto 13, 2025, <https://help.bullmarketbrokers.com/faq/nuevos-ratios-de-conversion/>
24. cambios de ratio cedears, fecha de acceso: agosto 13, 2025, <https://www.comafi.com.ar/Multimedios/otros/7118.xls>
25. lista total de cedears - Banco Comafi, fecha de acceso: agosto 13, 2025, <https://www.comafi.com.ar/Multimedios/otros/7279.xlsx?v=1032025>
26. Crypto Pairs Trading: Part 3 — Constructing Your Strategy with Logs, Hedge Ratios, and Z-Scores - Amberdata Blog, fecha de acceso: agosto 13, 2025, <https://blog.amberdata.io/constructing-your-strategy-with-logs-hedge-ratios-and-z-scores>
27. Orders - Creation/Execution - Backtrader, fecha de acceso: agosto 13, 2025, <https://www.backtrader.com/docu/order-creation-execution/order-creation-execution/>
28. Sizers - Backtrader, fecha de acceso: agosto 13, 2025, <https://www.backtrader.com/docu/sizers/sizers/>
29. How to write your first, simple backtrader strategy - Quant Nomad, fecha de acceso: agosto 13, 2025, <https://quantnomad.com/faq-how-to-write-your-first-simple-backtrader-strategy/>
30. I found a statistical arbitrage with ~1% return / day : r/algotrading - Reddit, fecha de acceso: agosto 13, 2025, <https://www.reddit.com/r/algotrading/comments/1m5keqj/i_found_a_statistical_arbitrage_with_1_return_day/>