## Lista de Tarefas para Correções do Bot-ZEUS

Aqui está uma lista de tarefas organizada para abordar os erros e pontos de melhoria identificados no seu bot. Recomendo seguir a ordem de prioridade para garantir que as questões mais críticas sejam resolvidas primeiro.

### Prioridade 1: Erros Críticos e Fundamentais

* **1.1. Manuseio de Decimais de Tokens (CRÍTICO - EM TODO O CÓDIGO):**
  + Remover ou refatorar a função converter\_para\_uint256 em utils/config.py para que ela aceite os decimais específicos do token como um argumento.
  + Implementar uma forma de obter os decimais de cada token (pode ser consultando o contrato ERC20 do token ou armazenando-os em config.py para os tokens suportados).
  + Revisar **todas** as ocorrências de conversão de quantidade (quantidade \* (10 \*\* 18)) em src/flash\_loan.py, src/arbitrage.py, utils/liquidity\_utils.py e utils/config.py para usar os decimais corretos do token em questão.
* **1.2. Arquitetura do Flash Loan (CRÍTICO - src/arbitrage.py e Contrato FlashLoanReceiver.sol):**
  + **No contrato FlashLoanReceiver.sol:** Implementar a lógica completa de arbitragem (swap de compra e swap de venda) dentro da função executeOperation. Esta função será chamada pelo protocolo Aave quando o flash loan for concedido.
  + **No src/arbitrage.py:** A função executar\_arbitragem\_com\_flashloan deve apenas chamar iniciar\_flash\_loan (que por sua vez interage com o contrato FlashLoanReceiver), e não chamar comprar\_token e vender\_token separadamente *após* a chamada do flash loan.
* **1.3. Incompatibilidade de ABIs e Chamadas de Funções DEX:**
  + **No src/flash\_loan.py (definir\_funcao\_transacao):**
    - Remover a verificação para swapExactTokensForTokens se o contrato dex\_contract for um roteador Uniswap V3 (pois ele usa exactInputSingle).
    - Garantir que a lógica para roteadores V2 (Quickswap/Sushiswap, que usam swapExactTokensForTokens) seja distinta e que o ABI correto seja carregado para eles.
  + **Nos utils/config.py e deploy.py:** Confirmar que os ABIs para Quickswap e Sushiswap estão corretos e contêm a função getAmountsOut e swapExactTokensForTokens se essas forem as funções esperadas.
* **1.4. Parâmetros do Construtor do Contrato de Flash Loan (deploy.py):**
  + **No deploy.py:** Corrigir os argumentos passados para o construtor do FlashLoanReceiver.sol (FlashLoanReceiver.constructor(...)) para que correspondam **exatamente** aos parâmetros esperados pelo contrato: provider, \_usdc, \_weth, \_uniswapV3Router, \_sushiswapRouter. Certifique-se de que wmatic\_address não seja passado para \_weth e quickswap\_router\_address não seja passado para \_sushiswapRouter.
* **1.5. Função initialize\_nonce Inexistente (utils/nonce\_utils.py e Chamadas):**
  + **Na classe NonceManager (utils/nonce\_utils.py):** Se a intenção era ter uma função para re-inicializar o nonce, renomeie reset\_nonce para initialize\_nonce ou crie uma nova função.
  + **Em todo o código:** Remover todas as chamadas nonce\_manager.initialize\_nonce() (pois a inicialização já ocorre no \_\_init\_\_) e usar nonce\_manager.sync\_with\_network() ou nonce\_manager.get\_nonce(refresh=True) para atualizações forçadas quando necessário.

### Prioridade 2: Correções e Melhorias Importantes

* **2.1. Ajustar Chamadas para iniciar\_bot\_arbitragem (src/bot\_main.py):**
  + **No src/bot\_main.py:** Modificar a função start\_bot e o loop principal para passar todos os argumentos necessários (web3, wallet\_address, private\_key, dex\_contracts, usdc\_address, etc.) para a função iniciar\_bot\_arbitragem em src/arbitrage.py.
* **2.2. Caminho ABI Incorreto (deploy.py):**
  + **No deploy.py:** Corrigir a variável abis\_dir para que ela aponte corretamente para a pasta abis na raiz do projeto, por exemplo, abis\_dir = os.path.join(base\_dir, 'abis').
* **2.3. Unificar convert\_usdt\_to\_matic:**
  + Mover a função convert\_usdt\_to\_wmatic de src/flash\_loan.py para src/dex\_operations.py (ou onde a função convert\_usdt\_to\_matic já está). Remover a duplicação.
* **2.4. Refatorar o Gerenciamento de Nonce (src/flash\_loan.py e src/arbitrage.py):**
  + Substituir a lógica manual de if receipt['status'] == 1: nonce\_manager.increment\_nonce() por nonce\_manager.incrementar\_se\_confirmado(receipt) para todas as transações, para garantir consistência e tratamento de falhas.
  + Remover chamadas duplas de nonce\_manager.increment\_nonce() em src/arbitrage.py.
* **2.5. Correção do Cálculo de Lucro (src/arbitrage.py):**
  + **No executar\_arbitragem\_com\_flashloan:** Obter o saldo\_inicial *antes* de executar as operações de compra e venda.
  + **Na chamada verificar\_lucro\_apos\_arbitragem:** Passar o saldo\_inicial (obtido antes) e o saldo\_final (obtido depois) corretamente.

### Prioridade 3: Otimizações e Boas Práticas

* **3.1. Otimização de gas\_limit (Em todo o código):**
  + Em vez de usar um gas\_limit fixo e muito alto, considere usar web3.eth.estimate\_gas(tx) para estimar o gás necessário para cada transação e adicionar uma pequena margem (e.g., 10-20%) para segurança. Isso otimizará os custos.
* **3.2. Unificação da Obtenção de gasPrice:**
  + Padronizar o uso de obter\_taxa\_gas em todas as transações (iniciar\_flash\_loan, realizar\_transacao, deploy.py) para garantir consistência na estratégia de gás.
* **3.3. Uso do Cliente Uniswap-Python (utils/config.py e utils/liquidity\_utils.py):**
  + Decidir se o SDK uniswap-python será utilizado. Se sim, adaptar obter\_liquidez\_uniswap\_v3 para utilizá-lo. Se não, remover a inicialização do uniswap\_client em config.py para evitar código não utilizado.
* **3.4. Refatorar calcular\_quantidade\_ideal (utils/liquidity\_utils.py):**
  + Considerar mover a lógica de reabastecimento de saldo para um módulo ou função separada, fora do loop principal de arbitragem, para manter a clareza das responsabilidades.
  + Garantir que as conversões de decimais dentro desta função também sejam corrigidas.