Daily Board #02 - Smart Wallet Complementar

NotusLabs DX Research - Testes Adicionais

Data: 01/10/2025

Sessão de Teste

1. Qual é o objetivo desta sessão?

Completar os testes da funcionalidade **Smart Wallet** da API NotusLabs, testando os endpoints complementares que não foram cobertos na sessão anterior:

- GET /wallets/{walletAddress}/portfolio
- GET /wallets/{walletAddress}/history
- POST /wallets/{walletAddress}/deposit
- PATCH /wallets/{walletId}/metadata
- PATCH /wallets/transactions/{transactionId}/metadata

---

2. Qual abordagem você vai usar?

Ferramenta: Postman para testes de API REST

Metodologia:

1. Testar cenários de sucesso (happy path)
2. Testar cenários de erro quando aplicável
3. Validar consistência com endpoints testados anteriormente

Cenários planejados:

- Visualizar portfolio de wallet vazia
- Visualizar histórico de wallet sem transações
- Tentar criar depósito (validar campos e formato)
- Atualizar metadata de wallet
- Verificar persistência de metadata
- Atualizar metadata de transação (validar formato)

---

3. Há algo que precisa ser configurado antes de começar?

Configurações já realizadas (da sessão anterior):

- Conta NotusLabs criada
- API Key obtida
- Postman configurado com collection "NotusLabs DX Research"

- Variáveis de ambiente:
  - `base_url` : `https://api.notus.team/api/v1`
  - `api_key` : [configurada]
  - `eoa_address` : `0x96832a27567538a804dfb8493513d7199ac63944`
  - `smart_wallet_address` : `0x34378c28a87ac84266211aa9b1c77caca241a659`
- 3 wallets já criadas (salts 0, 1, 2) disponíveis para testes

Contexto da sessão anterior:

- Daily Board #01 completado
- 8 testes realizados (register, get, list + 5 casos de erro)
- 4 problemas de DX identificados

4. Você conseguiu atingir o objetivo da sessão?

- Sim

Detalhamento: consegui testar todos os 5 endpoints complementares planejados. Cada endpoint foi testado com pelo menos um cenário de sucesso e/ou erro.

Resumo dos testes:

- Teste #9: GET /wallets/{address}/portfolio - Sucesso (após erro 500 inicial)
- Teste #10: GET /wallets/{address}/history - Sucesso
- Teste #11: POST /wallets/{address}/deposit - Validado (requer fundos reais)
- Teste #12: PATCH /wallets/{id}/metadata - Sucesso
- Teste #13: PATCH /wallets/transactions/{id}/metadata - Validado parcialmente

Estatísticas:

- Testes executados: 5
- Novos problemas identificados: 3
- Problemas confirmados/expandidos: 1 (problema #7 do Daily Board #01)
- Total de problemas (ambas sessões): 8

5. Problemas encontrados

Problema #5: Erro 500 intermitente no endpoint GET /portfolio

Severidade: Médio

Descrição: primeira tentativa de acessar portfolio resultou em erro 500 Internal Server Error. Segunda tentativa, segundos depois com mesma URL, funcionou corretamente retornando status 200.

Como reproduzir:

```
GET /wallets/0x34378c28a87ac84266211aa9b1c77caca241a659/portfolio
x-api-key: [API_KEY]
```

Primeira chamada (erro):

```
Status: 500 Internal Server Error
{
    "message": "An unexpected error occurred. Our team has been notified.",
    "id": "INTERNAL_SERVER_ERROR",
    "traceId": "1e99a5156d504cf7985d8066e3334a9b"
}
```

Segunda chamada (sucesso):

```
Status: 200 OK
{
    "tokens": [],
    "nfts": [],
    "portfolio": []
}
```

Impacto:

- Sugere instabilidade ou problema de infraestrutura
- Pode causar falhas intermitentes em produção
- Desenvolvedores precisarão implementar retry logic
- Afeta percepção de confiabilidade da API
- Experiência inconsistente para usuários finais

Possíveis causas:

- Cold start de funções serverless
- Problema de cache warming
- Timeout em serviço dependente
- Race condition no backend

Sugestão:

- Monitorar taxa de erro 500 especificamente neste endpoint
- Implementar retry automático no backend para requisições idempotentes
- Adicionar health check específico
- Investigar se ocorre em horários específicos

TraceId: `1e99a5156d504cf7985d8066e3334a9b`

Problema #6: Status code inadequado para erro de saldo insuficiente

Severidade: Médio

Descrição: quando usuário tenta criar depósito sem saldo suficiente de tokens, a API retorna status 500 Internal Server Error ao invés de 400 Bad Request.

Como reproduzir:

```
POST /wallets/0x34378c28a87ac84266211aa9b1c77caca241a659/deposit
Content-Type: application/json
```

```
x-api-key: [API_KEY]

Body:
{
  "amount": "1000000",
  "token": "0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48",
  "chainId": 1,
  "fromAddress": "0x96832a27567538a804dfb8493513d7199ac63944"
}
```

Response:

```
Status: 500 Internal Server Error
{
    "message": "Execution reverted with reason: ERC20: transfer amount exceeds balance.",
    "id": "FAILED_TO_ESTIMATE_TRANSFER",
    "traceId": "9a28608b82af49b6b5240bca5725cdbb"
}
```

Problema:

- Status 500 indica erro do servidor
- Mas o problema é erro do cliente (saldo insuficiente)
- Não segue convenções HTTP

Impacto na DX:

- Desenvolvedores podem implementar retry logic desnecessário para erro 500
- Sistemas de monitoring/logging categorizam incorretamente como server error
- Métricas de disponibilidade da API ficam incorretas
- Não segue padrões REST esperados

Pontos positivos:

- Mensagem de erro é clara sobre o problema real
- Detecta saldo antes de tentar executar (gas estimation)
- Inclui traceId para debug

Sugestão de melhoria: Retornar status 400 Bad Request com estrutura mais útil:

```
Status: 400 Bad Request
{
    "message": "Insufficient token balance for transfer",
    "id": "INSUFFICIENT_BALANCE",
    "details": {
        "token": "0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48",
        "tokenSymbol": "USDC",
        "required": "1000000",
        "available": "0"
    },
    "traceId": "..."
}
```

TraceId: `9a28608b82af49b6b5240bca5725cdbb`

Problema #7 (Confirmado e Expandido): GET /wallets/address ignora parâmetro address

Severidade: Crítico

Contexto: este problema foi inicialmente identificado no Daily Board #01 (Teste #2), mas nesta sessão foi **confirmado com evidência mais robusta** através do teste de metadata.

Descrição: o endpoint GET /wallets/address **ignora completamente o parâmetro** `address` fornecido na query string e sempre retorna a wallet com salt "0" baseado apenas em EOA + factory.

Evidência desta sessão:

Passo 1 - Atualizar metadata da wallet salt "2":

```
PATCH /wallets/0x34378c28a87ac84266211aa9b1c77caca241a659/metadata
Body: {
  "metadata": {
    "name": "Minha Wallet de Teste",
    "description": "Wallet criada durante DX Research",
    "tag": "test"
  }
}

Response: 200 OK
{
    "wallet": {
        "accountAbstraction": "0x34378c28a87ac84266211aa9b1c77caca241a659",
        "salt": "2",
        "metadata": {
            "name": "Minha Wallet de Teste",
            "description": "Wallet criada durante DX Research",
            "tag": "test"
        },
        ...
    }
}
```

✅ Metadata atualizado com sucesso na wallet salt "2"

Passo 2 - Tentar recuperar metadata via GET:

```
GET /wallets/address?
address=0x34378c28a87ac84266211aa9b1c77caca241a659&externallyOwnedAccount=0x96832a27567538a804dfb8493
513d7199ac63944&factory=0x0000000000400CdFef5E2714E63d8040b700BC24

Response: 200 OK
{
    "wallet": {
        "accountAbstraction": "0xe5c43b6b156e4d81b7c8939d1d2482715c8a9f32",
        "salt": "0",
        "metadata": null,
        ...
    }
}
```

❌ Retornou wallet ERRADA (salt "0" ao invés de salt "2")

Tentativas adicionais:

- Múltiplas requisições GET sempre retornam wallet salt "0"
- Independente do address fornecido no parâmetro

- Metadata da wallet salt "2" não pode ser recuperado

Impacto Crítico:

- **Bloqueia funcionalidade básica:** impossível consultar wallet específica por endereço
- **Usuários com múltiplas wallets:** não conseguem acessar wallets com salt ≠ 0
- **Metadata inutilizado:** dados atualizados não podem ser recuperados
- **Parâmetro inútil:** o parâmetro `address` não tem efeito algum
- **Contradição:** endpoint PATCH funciona com address, mas GET não
- **Problema de arquitetura:** sugere uso incorreto de chaves primárias

Comportamento observado:

- API busca usando apenas `externallyOwnedAccount + factory`
- Retorna sempre a primeira wallet encontrada (salt "0")
- Parâmetro `address` é completamente ignorado
- Não há forma de acessar wallets com salt diferente de "0"

Comportamento esperado:

- API deveria usar `address` como identificador primário (padrão Web3)
- Retornar a wallet específica solicitada
- EOA + factory seriam validação adicional ou opcionais

Relação com Daily Board #01:

- Teste #2 já havia identificado comportamento estranho
- Esta sessão confirma com caso de uso real (metadata)
- Problema mais grave do que inicialmente identificado

Problema #8: Inconsistência no formato de metadata entre endpoints

Severidade: Médio

Descrição: os dois endpoints de atualização de metadata têm formatos incompatíveis para o campo metadata.

PATCH /wallets/{id}/metadata - Aceita OBJECT:

```
PATCH /wallets/0x34378c28a87ac84266211aa9b1c77caca241a659/metadata

Body:
{
  "metadata": {
    "name": "Minha Wallet",
    "description": "Descrição detalhada",
    "tag": "test"
  }
}

Response: 200 OK ✅
```

PATCH /wallets/transactions/{id}/metadata - Aceita apenas STRING:

```
PATCH /wallets/transactions/invalid-transaction-id/metadata

Body (INCORRETO):
{
  "metadata": {
    "note": "Test transaction",
    "category": "deposit"
  }
}

Response: 400 Bad Request
{
    "message": "Bad Request",
    "id": "BAD_REQUEST",
    "errors": [{
        "code": "invalid_type",
        "expected": "string",
        "received": "object",
        "path": ["metadata"],
        "message": "Expected string, received object"
    }]
}
```

```
Body (CORRETO):
{
  "metadata": "Test transaction for DX research"
}

Response: 404 Not Found ✅ (transação não existe, mas formato aceito)
```

Impacto na DX:

> - Desenvolvedores precisam lembrar formatos diferentes
> - Aumenta complexidade do código cliente (dois serializers)
> - Não há razão técnica óbvia para formatos diferentes
> - Documentação não explica ou menciona essa diferença
> - Descoberta apenas por tentativa e erro
> - Inconsistência gera frustração

Sugestões de melhoria:

Opção 1 (Recomendada): ambos endpoints aceitarem object JSON

```
{
  "metadata": {
    "key": "value",
    ...
  }
}
```

> - Mantém flexibilidade
> - Permite estruturação de dados
> - Padrão mais comum em APIs modernas

Opção 2: Ambos aceitarem string

```
{
  "metadata": "string value"
}
```

- Mais simples
- Menos flexível
- Cliente precisa serializar manualmente se quiser estrutura

Opção 3 (Mínimo):

- Documentar claramente a diferença
- Explicar por que há diferença (se intencional)
- Adicionar exemplos explícitos na documentação

---

6. Observações adicionais

Testes Executados - Detalhamento

Teste #9: Get Smart Wallet Portfolio

- **Endpoint:** GET /wallets/{walletAddress}/portfolio
- **Status:** Sucesso (após retry)
- **Problema identificado:** Erro 500 intermitente (#5)

Primeira tentativa: 500 Internal Server Error
Segunda tentativa: 200 OK com response válido

Response (sucesso):

```
{
    "tokens": [],
    "nfts": [],
    "portfolio": []
}
```

Observações:

- Estrutura clara com separação por tipo de ativo
- Arrays vazios para wallet sem ativos (comportamento correto)
- Documentação poderia explicar diferença entre os 3 arrays

Teste #10: Get Smart Wallet History

- **Endpoint:** GET /wallets/{walletAddress}/history
- **Status:** Sucesso
- **Sem problemas identificados**

Response:

```
{
    "nextLastId": null,
    "transactions": []
}
```


Observações:

- Usa cursor-based pagination (nextLastId) - boa prática
- Mais eficiente que offset pagination
- Arrays vazios para wallet sem transações (correto)
- Documentação poderia explicar:
  - Como usar nextLastId para paginar
  - Qual query parameter passar
  - Limite padrão de registros
  - Filtros disponíveis

Teste #11: Create Deposit Transaction

- **Endpoint:** POST /wallets/{walletAddress}/deposit
- **Status:** funcional mas não testável completamente
- **Problemas identificados:** Status code incorreto (#6), documentação incompleta

Campos obrigatórios descobertos:

- `amount` (string) - valor em unidades mínimas do token
- `token` (string) - endereço do contrato ERC-20
- `chainId` (number) - ID da blockchain
- `fromAddress` (string) - endereço de origem dos fundos

Validações testadas:

- ❌ Token `0x0000...0000` (ETH nativo) - não suportado
- ✅ Token `0xA0b8...eB48` (USDC Ethereum) - suportado
- ✅ Detecta saldo insuficiente antes de executar

Problemas de documentação:

- Não lista tokens suportados
- Não lista chains suportadas
- Não explica formato do amount
- Limitação sobre ETH nativo não documentada

Teste #12: Update Wallet Metadata

- **Endpoint:** PATCH /wallets/{walletAddress}/metadata

- **Status:** sucesso completo
- **Sem problemas identificados neste endpoint**

Request:

```
{
  "metadata": {
    "name": "Minha Wallet de Teste",
    "description": "Wallet criada durante DX Research",
    "tag": "test"
  }
}
```

Response: 200 OK - Wallet completa com metadata atualizado

Observações positivas:

- Funciona perfeitamente
- Aceita object JSON com estrutura livre
- Flexibilidade total
- Response confirma atualização imediatamente

Limitação: Metadata não pode ser verificado via GET devido ao problema #7

Teste #13: Update Transaction Metadata

- **Endpoint:** PATCH /wallets/transactions/{transactionId}/metadata
- **Status:** validado parcialmente
- **Problema identificado:** formato inconsistente (#8)

Formato correto (string):

```
{
  "metadata": "Test transaction for DX research"
}
```

Response: 404 Not Found (transação não existe, mas formato aceito)

Formato incorreto (object):

```
{
  "metadata": {
    "note": "...",
    "category": "..."
  }
}
```

Response: 400 Bad Request - "Expected string, received object"

Observações:

- Validação de formato funciona corretamente
- Erro 404 apropriado para recurso inexistente

- Inconsistente com endpoint de wallet metadata

Pontos Positivos da API

1. **Portfolio Endpoint:**
   - Organização clara por tipo de ativo
   - Response limpo e estruturado
   - Diferenciação entre tokens e NFTs
2. **History Endpoint:**
   - Cursor-based pagination (moderna e eficiente)
   - Adequado para infinite scroll
   - Performance melhor com grandes volumes
3. **Deposit Endpoint:**
   - Validações robustas de campos
   - Detecta tokens não suportados rapidamente
   - Gas estimation antes de executar (evita transações falhadas)
   - Mensagens de erro claras (exceto status code)
4. **Wallet Metadata:**
   - Flexibilidade total na estrutura
   - Facilita categorização e organização
   - Útil para melhorar UX
5. **Transaction Metadata:**
   - Validação de formato funciona
   - Tratamento de erro 404 apropriado

Sugestões de Melhoria Prioritárias

Alta Prioridade:

1. **Resolver problema #7 (Crítico):**
   - GET /wallets/address deve usar o parâmetro address
   - Permitir acesso a wallets com qualquer salt
   - Considerar criar endpoint alternativo GET /wallets/{address}
2. **Padronizar formato de metadata (#8):**
   - Ambos endpoints aceitarem object JSON
   - Ou documentar claramente a diferença
3. **Corrigir status code do deposit (#6):**
   - Usar 400 para saldo insuficiente
   - Incluir detalhes no response (saldo disponível vs requerido)

Média Prioridade:

4. **Investigar instabilidade do portfolio (#5):**
   - Monitorar taxa de erro 500
   - Implementar retry automático
   - Melhorar infraestrutura/cache
5. **Melhorar documentação do deposit:**
   - Listar tokens suportados
   - Listar chains suportadas
   - Explicar formato do amount
   - Criar endpoint GET /tokens/supported

Baixa Prioridade:

6. **Documentação geral:**
   - Explicar nextLastId e paginação
   - Exemplos de portfolio/history com dados
   - Boas práticas para metadata