Fullstore

Documentação técnica da integração entre o isnapp e a plataforma FullStore. Destinada à equipe de T.I. responsável pela manutenção, suporte e diagnóstico do serviço.

Token do serviço: **/bibliotecas/ba4e7bb5-4770-4780-beaa-b28980153cc7/wosk/webservice/fullStoreService**
Namespace: **WOSK\fullStoreService**

- Visão Geral
- Arquitetura e Tecnologias
- Fluxo Funcional
- Canal PDV — Vendas
- Canal Trocas e Devoluções
- Canal Omni
- Canal E-commerce
- API — Parâmetros e Resposta
- Configuração e Dependências
- Troubleshooting e Testes

Visão Geral

O que consolidamos

Nossa integração coleta dados de vendas de **quatro canais distintos** e os entrega à FullStore em um único JSON estruturado, permitindo que a plataforma calcule comissões e analise o desempenho de cada vendedor sem precisar acessar diretamente os bancos de dados internos.

PDV

Vendas realizadas no balcão da loja física, com itens e NF-e.

Trocas

Devoluções e trocas vinculadas à NF da venda original.

Omni

Pedido criado pelo vendedor na loja com entrega em domicílio ou retirada em outra unidade.

E-commerce

Pedidos do site processados pelo sistema Linx (SQL Server).

Sistemas envolvidos

Sistema	Papel	Banco
ERP OSK (loja)	Fonte de dados — PDV, Trocas, Omni	MySQL

Sistema	Papel	Banco
Linx	Fonte de dados — E-commerce e BI	SQL Server `LX_ZERO_300`
FullStore	Consumidora da API	—

Ação disponível

Disponibilizamos uma única ação: `getVendas`. Ela recebe CNPJ da loja e intervalo de datas, consulta os quatro canais em sequência e retorna tudo consolidado.

Estrutura da resposta

```
{
  "success": true,
  "message": "Vendas retornadas com sucesso!",
  "data": {
    "vendas":           [ /* PDV — vendas com itens agrupados       */ ],
    "trocas":           [ /* Trocas agrupadas por NF original        */ ],
    "omni":             [ /* Pedidos omni com entrega/retirada        */ ],
    "vendas_ecommerce": [ /* Pedidos e-commerce vindos do SQL Server  */ ],
    "fallback": null      /* Preenchido se o período exceder o limite */
  }
}
```

Restrições de período

Limite de 200 dias no passado Consultamos no máximo **200 dias retroativos** (`LIMITE_DIAS_CONSULTA = 200`). Se `dataInicio` ou `dataFim` estiver além desse limite, retornamos `success: true` com todos os arrays vazios e o objeto `fallback` preenchido com os parâmetros recebidos — sem erro de sistema.
CNPJ obrigatório O `cnpjLoja` é obrigatório e filtra *todos* os canais. Sem ele, a requisição falha na validação antes de qualquer consulta ao banco.

Arquitetura e Tecnologias

Stack técnico, hierarquia de classes, dependências e conexões com bancos de dados.

Diagrama de componentes

FullStore HTTP POST getVendas isnapp controle.php PHP 7.3 setSubmit() MySQL OSK PDV, Trocas Omni, filiais SQL Server Linx/BI LX_ZERO_300 W_BI_FAT_VENDEDOR... CNPJ + periodo CTEs + filiais prepared statement JSON success / message / data Figura 1 - Componentes e fluxo de dados da integracao.

Stack

Camada	Tecnologia	Detalhe
Linguagem	PHP **7.3**	Namespaces, PDO, Traits
Framework	CodeIgniter (`MY_Controller`)	Base do backend
Banco operacional	MySQL / PDO	PDV, Omni, Trocas
Banco E-commerce	SQL Server / PDO `sqlsrv:`	Linx `LX_ZERO_300`
Resposta	JSON via `setSubmit()`	Envelope padrão `success / message / data`

Hierarquia de classes

biblioteca base do loader fullStoreService carrega classe por UUID/token utilOsk PDO MySQL OSK WOSK\fullStoreService\WebService valida, consulta canais e monta resposta WOSK\Commons\WebService trait: setSubmit(), logs db_sqlserver PDO sqlsrv para e-commerce usa trait instancia Cadeia principal biblioteca -> fullStoreService / utilOsk -> WOSK\fullStoreService\WebService -> trait + db_sqlserver

Arquivos required

Em `controle.php`, carregamos três dependências via `require_once(APPPATH . '...')`:

UUID	Arquivo	Fornece
`4cb18a6c-...`	`wosk_webservice.php`	Trait `WOSK\Commons\WebService` — formata JSON (`setSubmit`), valida campos e salva logs no MySQL
`592ccfa0-...`	`utilOsk.php`	Classe pai — conecta ao MySQL OSK e expõe `$this->conexao` (PDO)
`0670acec-...`	`db_sqlserver.php`	Gerencia conexão PDO com SQL Server (driver `sqlsrv:`)

Conexão MySQL

Como configuramos

A classe `utilOsk` chama `getConfiguracao('db_osk_prod')` — credenciais armazenadas em Base64 na tabela de configuração do sistema. Em seguida, `_setConexao()` cria a conexão PDO:

```
// connection string:
"mysql:host={host};port={porta};dbname={banco}"
// PDO::ATTR_PERSISTENT  = true   (conexão persistente)
// PDO::ATTR_TIMEOUT     = 120    (timeout em segundos)
// charset               = UTF-8
```

Conexão SQL Server

Instanciamos em buscarDadosEcommercePorLoja()

```
$sqlserver = new \db_sqlserver(false, 'LX_ZERO_300', true);
// false  = produção (não homologação)
// 'LX_ZERO_300' = banco Linx
// true   = conecta imediatamente
```

Ambiente	Host	Porta	Banco
Produção	`<HOST_SQLSERVER_PROD>`	`<PORTA_PROD>`	`LX_ZERO_300`
Homologação	`<HOST_SQLSERVER_HML>`	`<PORTA_HML>`	`LINX_HMLG`

Credenciais hardcoded As credenciais do SQL Server ficam em `db_sqlserver.php`.

Tabelas MySQL utilizadas

Tabela	Alias	O que fornece
`movimentacao`	m	Cabeçalho da transação (tipo, data, situação, módulo)
`movimentacao_detalhe`	md	Itens da venda (produto, qtde, preço)
`movimentacao_nfe`	nf / nfe	Número, série e situação da NF-e
`movimentacao_movimentacao`	mm	Vínculos entre movimentações (troca↔venda, omni↔orçamento)
`pessoa`	v	Nome e código do vendedor
`entidade`	e	Código da filial / loja
`juridica`	j	CNPJ da empresa

Objetos SQL Server utilizados

Objeto	Tipo	O que fornece
`W_BI_FAT_VENDEDOR_VDK_OMNI_V0`	View BI	Pedidos e-commerce por vendedor: NF, valor, CPF, filial, pedido site/WMS
`FILIAIS`	Tabela	Cadastro de filiais — usamos para excluir franquias (`TIPO_FILIAL <> 'FRANQUIA'`)

Fluxo Funcional

Passo a passo de uma requisição, validações e tratamento de erros.

Diagrama de sequência

FullStore isnapp run() + validação MySQL OSK SQL Server Linx/BI POST getVendas Normaliza período verifica limite 200 dias PDV / Trocas Omni + filiais consultas MySQL E-commerce W_BI_FAT_VENDEDOR... placeholders por filial mapeia vendas_ecommerce Monta JSON vendas / trocas / omni Figura 3 - Sequencia completa de uma chamada a acao getVendas.

Diagrama de fluxo (decisões)

POST getVendas ação válida? success:false Acao nao encontrada parâmetros preenchidos? success:false O parametro {campo} é obrigatorio dentro de 200 dias? success:false fallback interno não exposto consulta quatro canais se algum canal tem dados: success:true não sim não sim não sim Figura 4 - Fluxo de decisão completo do endpoint.

Passo a passo detalhado

1. **Recebimento da requisição**
   Recebemos o payload via `run(array $params)`. O parâmetro `acao` deve ser `'getVendas'`; qualquer outro valor resulta em erro imediato.
2. **Validação de parâmetros**
   Validamos via `validarParametrosObrigatorios()`: `dataInicio`, `dataFim` e `cnpjLoja` — presença e valor não vazio. Se algum faltar, lançamos uma exception com o nome do campo faltante.
3. **Normalização do período**
   Via `normalizarPeriodo()`, adicionamos horários às datas: início fica `dataInicio 00:00:00` e fim `dataFim 23:59:59`, garantindo cobertura integral dos dias extremos.
4. **Verificação do limite**
   Via `periodoExcedeLimite()`, checamos se alguma das datas está além de 200 dias no passado. Se sim, `getVendasConsolidadas()` monta arrays vazios e um objeto `fallback`, mas `run()` ainda trata o resultado como nenhum registro encontrado e retorna erro ao

consumidor.

5. **Vendas PDV**
   Executamos query com 2 CTEs nas tabelas `movimentacao + movimentacao_detalhe + movimentacao_nfe`. Retornamos uma linha por item e depois agrupamos por `identificador` via `agruparItensDasVendas()`, gerando o sub-array `itens`.
6. **Trocas**
   Buscamos movimentações de entrada com `tipo_estoque = 'TROCA'`, vinculando à venda original via `movimentacao_movimentacao`. Consolidamos múltiplas trocas do mesmo documento somando valores em `agruparTrocasPorDocumento()`.
7. **Omni**
   Buscamos pedidos omni criados pelo vendedor na loja (orçamentos com `tipo_estoque = 'ORCAMENTO'` e situação Confirmado/Finalizado), vinculados à venda real via `movimentacao_movimentacao`.
8. **E-commerce**
   Em dois sub-passos: (1) resolvemos os códigos de filial ativas via MySQL/CNPJ; (2) consultamos a view SQL Server com placeholders paramétricos dinâmicos para cada filial.
9. **Resposta**
   Montamos o objeto `data` com os quatro arrays e retornamos via `setSubmit(true, 'Vendas retornadas com sucesso!', $data)`.

Tratamento de erros

Situação	Comportamento	success
Parâmetro obrigatório ausente	Mensagem com nome do campo faltante	false
Ação inválida (≠ getVendas)	Mensagem de ação não encontrada	false
Período além de 200 dias	Fallback interno é montado, mas a resposta final atual é erro de nenhum registro	false
CNPJ sem filiais ativas no MySQL	`vendas_ecommerce` vazio; se os demais canais também estiverem vazios, retorna erro de nenhum registro	Depende
Falha no SQL Server	Exception capturada → retorno `false`	false
Parâmetro `falha` enviado	Força erro (uso em testes)	false

Logs automáticos de requisição Toda chamada — mesmo as que resultam em erro — é salva automaticamente nas tabelas `wosk_webservice*` do banco integrador. Em caso de disputa ou diagnóstico, basta consultar `wosk_webservice_retorno` pelo timestamp.

Canal PDV — Vendas

Capturamos as vendas registradas no módulo **PDV** — transações de saída com NF-e autorizada/cancelada quando existir vínculo em `movimentacao_nfe`. Cada venda pode ter um ou mais itens; nossa query retorna uma linha por item e o método `agruparItensDasVendas()` os agrupa em um array aninhado `itens`.

Métodos: `consultarVendasPdv()` → `agruparItensDasVendas()`

Relacionamento das tabelas

movimentacao id, data_emissao, situacao modulo = PDV, tipo = SAIDA id_pessoa, id_vendedor, id_entidade movimentacao_nfe numero, serie, situacao movimentacao_detalhe codigo, EAN, qtde, valor_rateio entidade loja da venda juridica cnpjLoja pessoa codigo vendedor nf.id_movimentacao md.id_movimentacao m.id_entidade e.id_pessoa m.id_vendedor Figura 5 - Tabelas MySQL envolvidas no canal PDV e seus relacionamentos.

Filtros aplicados

Campo	Valor / Condição	Motivo
`m.modulo`	`= 'PDV'`	Apenas transações do módulo de caixa
`m.tipo`	`= 'SAIDA'`	Apenas saídas (vendas)
`m.situacao`	`<> 'Cancelado'`	Excluímos movimentações canceladas no cabeçalho
`m.data_emissao`	`BETWEEN :dataInicio AND :dataFim`	Filtro de período normalizado
`j.cnpj`	`= :cnpjLoja`	Restringe à loja da requisição
`nf.situacao`	`IN ('AUTORIZADO','CANCELADO')`	NF-e com situação definida no SEFAZ
`md.situacao`	`= 'ATIVO'`	Apenas linhas de detalhe ativas

Estrutura da query (2 CTEs)

CTEs utilizados

1. **movimentacoes** — agrupa dados do cabeçalho: vendedor, NF-e, entidade, CNPJ
2. **itens** — junta os detalhes de produto aos cabeçalhos

O SELECT final une os dois CTEs retornando uma linha por item, depois agrupamos em PHP.

Retorno da API

```
// Um elemento do array "vendas"
{
  "codigo_cliente":  123,
  "cod_vendedor":    "V001",
  "cnpj_emp":        "12345678000190",
  "data_documento":  "2026-05-15T10:30:00-03:00",   // ISO 8601
  "identificador":   9001,
  "transacao":       9001,
  "usuario":         42,
  "documento":       "000001234",   // Número NF-e
  "serie":           "001",
  "operacao":        "S",
  "tipo_transacao":  "VENDA",       // Fixo neste canal
  "cancelado":       "N",           // Derivado de movimentacao.situacao; na query atual tende
a "N"
  "delivery":        false,
  "itens": [
    {
      "cod_produto":        "PROD-001",
      "descricao_produto":  "Camiseta Branca M",
      "cod_barra":          "7891234567890",
      "quantidade":         2,
      "preco_unitario":     89.90,
      "valor_total":        179.80
    }
  ]
}
```

Dicionário de campos

Campo	Origem no banco	Descrição
`identificador`	`movimentacao.id`	Chave da venda — usada no agrupamento
`data_documento`	`movimentacao.data_emissao`	Data/hora da venda em ISO 8601
`cod_vendedor`	`pessoa.codigo`	Código do vendedor responsável
`codigo_cliente`	`movimentacao.id_pessoa`	ID do cliente
`cnpj_emp`	`juridica.cnpj`	CNPJ da loja
`documento`	`movimentacao_nfe.numero`	Número da NF-e
`serie`	`movimentacao_nfe.serie`	Série da NF-e
`cancelado`	Derivado de `movimentacao.situacao`	Como a query filtra `m.situacao <> 'Cancelado'`, na prática tende a `"N"`
`tipo_transacao`	Hardcoded	Sempre `"VENDA"` neste canal
`itens[].cod_barra`	`movimentacao_detalhe.codigo_ean`	EAN do produto
`itens[].valor_total`	`movimentacao_detalhe.valor_rateio`	Valor rateado retornado como total do item

Cancelamentos no cabeçalho são filtrados O campo `cancelado` é calculado a partir de `movimentacao.situacao`, não de `movimentacao_nfe.situacao`; portanto, NF-e cancelada pode aparecer no join, mas esse cancelamento não altera automaticamente o flag retornado.

Canal Trocas e Devoluções

Capturamos movimentações de *entrada* do tipo `TROCA`. Quando o cliente devolve ou troca uma peça, o sistema OSK registra uma nova movimentação de entrada e a vincula à venda original via tabela `movimentacao_movimentacao`.

Métodos: `getTrocas()` → `agruparTrocasPorDocumento()`

Vínculo troca ↔ venda original

Troca ENTRADA / TROCA situacao = FINALIZADO movimentacao_movimentacao id_movimentacao_pai = troca id_movimentacao = venda original Venda original SAIDA / PDV movimentacao ms NF-e numero serie mm.id_mov_pai mm.id_movimentacao nfe Figura 6 - movimentacao_movimentacao conecta a troca a venda original.

Filtros aplicados

Campo	Valor	Motivo
`m.modulo`	`'PDV'`	Apenas trocas do PDV
`m.tipo`	`'ENTRADA'`	Mercadoria retornando ao estoque
`m.tipo_estoque`	`'TROCA'`	Distingue de compras e outras entradas
`m.situacao`	`'FINALIZADO'`	Apenas trocas concluídas
`m.data_emissao`	`BETWEEN :dataInicio AND :dataFim`	Período da requisição
`j.cnpj`	`= :cnpjLoja`	Restringe à loja

Estrutura da query (3 CTEs)

CTEs em ordem de execução

1. **e** — filtra a entidade (loja) pelo CNPJ informado

2. **m** — agrega movimentações de troca: soma valores e quantidades por movimentação
3. **tr** — junta ao documento original via `movimentacao_movimentacao` e busca a NF de venda

Agrupamento por documento

A mesma NF pode ter múltiplas trocas (cliente troca um item hoje e outro amanhã). Em `agruparTrocasPorDocumento()` consolidamos assim:

- Somamos `valor_vale` e `valor_original`
- A implementação atual não soma `quantidade` ao agrupar documentos repetidos; mantém a quantidade do primeiro registro do grupo
- Mantemos o **timestamp mais recente**
- Trocas sem vínculo com NF original são incluídas individualmente

Retorno da API

```
// Um elemento do array "trocas"
{
  "motivo":         "Defeito no produto",
  "doc_venda":      "000001234",             // NF da venda original
  "serie_venda":    "001",
  "cod_cliente":    123,
  "cnpj_emp":       "12345678000190",
  "valor_vale":     89.90,                   // Crédito gerado para o cliente
  "valor_original": 89.90,                   // Mesmo valor agregado da troca na implementação
atual
  "timestamp":      "2026-05-18T14:22:00-03:00",
  "cod_vendedor":   "V001",
  "quantidade":     1
}
```

Dicionário de campos

Campo	Origem	Descrição
`doc_venda`	`movimentacao_nfe.numero` da venda	NF da venda que originou a troca

Campo	Origem	Descrição
`serie_venda`	`movimentacao_nfe.serie` da venda	Série da NF original
`valor_vale`	Soma dos itens da troca	Crédito gerado para o cliente
`valor_original`	`SUM(md.valor_rateio)` da movimentação de troca	Na implementação atual, recebe o mesmo agregado usado em `valor_vale`
`timestamp`	`movimentacao.data_emissao`	Data/hora da troca — ISO 8601
`cod_vendedor`	`pessoa.codigo`	Vendedor que registrou a troca
`quantidade`	`SUM(md.qtde)` por movimentação de troca	Não é acumulado novamente quando múltiplas trocas são agrupadas pelo mesmo documento

Trocas sem documento original Se a troca não encontrar uma NF de venda (`doc_venda` nulo), ela é incluída individualmente sem agrupamento. Isso ocorre quando a venda foi feita antes da implantação do controle de vínculos no sistema.

Canal Omni

Pedido criado pelo vendedor na loja física com entrega em domicílio ou retirada em outra unidade.

Implementação recente Canal adicionado em maio de 2026 — commit `2b74ab7` ("IMPLEMENTACAO VENDA OMNI"). É o canal com a lógica mais complexa do endpoint, usando 4 CTEs e um SELECT final.

Fluxo do pedido omni

Cliente na loja atendimento presencial Pedido pelo site vendedor cria pedido no sistema integrado Orçamento omni tipo_estoque = ORCAMENTO id_grupo_operacao = 192 id_tipo_movimentacao = 6 fulfillment do pedido Venda vinculada movimentacao_movimentacao Retirada outra unidade pedido transita entre lojas JSON omni[] retirada entrega Figura 7 - Fluxo completo de um pedido omni-channel.

O cliente vai fisicamente até a loja. O vendedor, usando o sistema, **cria o pedido pelo site** (plataforma de e-commerce VTEX). A partir daí, existem duas possibilidades de fulfillment:

- **Cliente vai à loja**
  Vendedor atende o cliente presencialmente e registra o pedido no sistema via plataforma web.
- **Pedido criado pelo site**
  O sistema gera um *orçamento* (`tipo_estoque = 'ORCAMENTO'`) com `id_tipo_movimentacao = 6` e `id_grupo_operacao = 192`.
- **Fulfillment: Entrega em domicílio**
  O produto é despachado para o endereço do cliente — a venda real é registrada como saída no sistema e vinculada ao orçamento via `movimentacao_movimentacao`.
- **Fulfillment: Retirada em outra unidade**
  O cliente retira o produto em uma loja diferente da que realizou a venda — o pedido transita entre unidades antes de ser entregue.

Identificadores do canal omni

Identificamos os pedidos omni pela combinação de três campos em `movimentacao`:

Campo	Valor	Significado
`id_grupo_operacao`	`192`	Grupo de operação exclusivo do canal omni
`id_tipo_movimentacao`	`6`	Tipo de movimentação omni
`tipo_estoque`	`'ORCAMENTO'`	Registrado como orçamento — não movimenta estoque diretamente
`situacao`	`'Confirmado'` ou `'Finalizado'`	Buscamos apenas pedidos que resultaram em venda real

Estrutura da query (4 CTEs)

CTEs em ordem de execução

1. **e** — filtra a entidade (loja) pelo CNPJ
2. **orcamento** — busca orçamentos omni confirmados/finalizados no período
3. **mm** — busca vínculos em `movimentacao_movimentacao` para localizar a venda real (SAIDA)
4. **i** — busca os itens do *orçamento* via `movimentacao_detalhe` na implementação atual
5. **SELECT final** — une orçamento + vendedor + itens

Retorno da API

```
// Um elemento do array "omni"
{
  "id_cliente":      456,
  "cod_vendedor":    "V007",
  "cnpj_emp":        "12345678000190",
  "data_documento":  "2026-05-20 16:45:00",        // Data da venda vinculada, sem conversão
ISO no método atual
  "id_pessoa":       88,                           // ID do vendedor em pessoa
  "identificador":   3001,                         // ID da movimentação de venda vinculada
  "transacao":       3001,
  "usuario":         42,
  "documento":       "ORÇ-00321",
  "vendedor":        "João Silva",                 // Nome completo do vendedor
  "cancelado":       "N",
  "itens": [
    {
```

```
      "cod_produto":        "PROD-099",
      "descricao_produto":  "Tênis Running 42",
      "cod_barra":          "7896543219870",
      "quantidade":         1,
      "preco_unitario":     349.90,
      "valor_total":        349.90
    }
  ]
}
```

Itens e cabeçalho vêm de fontes diferentes Na implementação atual, o cabeçalho usa a movimentação vinculada pela tabela `movimentacao_movimentacao`, mas a CTE `i` busca os itens do orçamento (`i.id_movimentacao = orcamento.id_movimentacao`). Como o join com os itens é interno, se o orçamento não tiver itens ativos o registro omni não é retornado.

Canal E-commerce

Consulta ao SQL Server (Linx) e mapeamento de colunas para o padrão FullStore.

Coletamos pedidos do e-commerce processados pelo sistema **Linx**. Esses dados ficam em um SQL Server separado (`LX_ZERO_300`), em uma view que já consolida faturamento por vendedor. Nosso processo ocorre em **dois passos**.

Métodos: `buscarDadosEcommercePorLoja()` → `getVendasEcommerce()`

Diagrama de sequência — dois passos

isnapp buscarDadosEcommerce por Loja() Passo 1 - MySQL juridica + entidade j.cnpj = :cnpj e.situacao = ATIVO retorna codigos filial Placeholders :filial0, :filial1... bindValue() Passo 2 - SQL Server W_BI_FAT_VENDEDOR... JOIN FILIAIS TIPO_FILIAL != FRANQUIA vendas_ecommerce[] resolve CNPJ monta IN consulta BI Figura 8 - Sequencia dos dois passos de consulta do canal e-commerce.

Mapeamento de colunas SQL Server → JSON

Coluna SQL Server	Campo no JSON	Descrição
`CGC_CPF`	`documento_cliente`	CPF/CNPJ do cliente
`VENDEDOR`	`cod_vendedor`	Código do vendedor no Linx
`CUPOM_VENDEDOR`	`cupom_vendedor`	Cupom aplicado pelo vendedor
`CODIGO_FILIAL`	`codigo_filial`	Código da filial (padded com zeros)
`EMISSAO`	`data_documento`	Data de emissão no formato retornado pelo SQL Server; o método atual não converte para ISO 8601

Coluna SQL Server	Campo no JSON	Descrição
`PEDIDO_SITE`	`pedido_site`	Número do pedido no e-commerce
`PEDIDO_WMS`	`pedido_wms`	Número do pedido no WMS
`TICKET`	`ticket`	Ticket de atendimento
`NF`	`documento`	Número da NF
`SERIE`	`serie`	Série da NF
`VALOR_LIQUIDO`	`valor`	Valor líquido do pedido
`QTDE_LIQUIDA`	`qtde`	Quantidade líquida de itens

Tratamento de espaços Aplicamos `rtrim()` e `ltrim()` em todos os campos mapeados — o SQL Server da Linx pode devolver strings com espaços à esquerda ou direita.

Retorno da API

```
// Um elemento do array "vendas_ecommerce"
{
  "documento_cliente": "123.456.789-00",
  "cod_vendedor":      "EC001",
  "cupom_vendedor":    "VEND10OFF",         // Pode ser string vazia
  "codigo_filial":     "000001",
  "data_documento":    "2026-05-22 09:15:00",
  "pedido_site":       "PS-98765",
  "pedido_wms":        "WMS-54321",
  "ticket":            "TKT-11111",
  "documento":         "000005678",
  "serie":             "001",
  "valor":             "259.90",
  "qtde":              "2"
}
```

Segurança — placeholders dinâmicos

O número de filiais varia por CNPJ. Em vez de concatenar valores na query, geramos placeholders e usamos `bindValue()` para cada um:

```
// Geração dos placeholders (PHP 7.3)
$placeholders = [];
foreach ($filiais as $i => $codigo) {
    $key = ':filial' . $i;
    $placeholders[] = $key;
    $stmt->bindValue($key, $codigo);
}
// Resultado na query: IN (:filial0, :filial1, :filial2)
// Nenhum valor externo é interpolado na string SQL.
```

Proteção contra SQL injection Todos os parâmetros são vinculados via prepared statements — datas com `bindParam()` e códigos de filial com `bindValue()`. Nenhuma concatenação de input externo ocorre.
CNPJ sem filial ativa `buscarDadosEcommercePorLoja()` retorna um envelope de erro quando não encontra filial ativa no MySQL, mas `getVendasConsolidadas()` consome apenas a chave `data`. Na resposta final, isso aparece como `vendas_ecommerce: []`; se os demais canais também estiverem vazios, o endpoint retorna erro de nenhum registro.

API — Parâmetros e Resposta

Especificação completa de getVendas: entrada, saída, erros e exemplos JSON.

Identificação do serviço

Referência

Token: `<TOKEN_FULLSTORE>`
Método HTTP: `POST` | **Formato:** JSON

BaseUrl: HTTPS://SEUSISTEMA.PDV.MODA/bibliotecas/ba4e7bb5-4770-4780-beaa-b28980153cc7/wosk/webservice/fullStoreService

Payload de entrada

```
{
    "acao": "getVendas",
    "key":" CHAVE API FORNECIDA",
    "parametros": {
        "dataInicio": "2026-03-10",
        "dataFim": "2026-03-10",
        "cnpjLoja": ""
    }
}
```

Campo	Tipo	Formato	Obrig.	Descrição
`acao`	string	literal	Sim	Deve ser exatamente `"getVendas"`

Campo	Tipo	Formato	Obrig.	Descrição
`parametros.dataInicio`	string	`Y-m-d`	Sim	Início do período. Ex: `"2026-05-01"`
`parametros.dataFim`	string	`Y-m-d`	Sim	Fim do período. Ex: `"2026-05-31"`
`parametros.cnpjLoja`	string	14 dígitos	Sim	CNPJ sem máscara — filtra todos os canais
`falha`	any	—	Teste	Se presente e não nulo, forçamos retorno de erro

Limite de 200 dias Se `dataInicio` ou `dataFim` estiver além de 200 dias no passado, a implementação atual monta um `fallback` interno, mas a resposta final retorna `success: false` com mensagem de nenhum registro encontrado.

Resposta de sucesso (com dados)


```
{
  "success": true,
  "message": "Vendas retornadas com sucesso!",
  "data": {
    "vendas": [
      {
        "codigo_cliente": 123, "cod_vendedor": "V001",
        "cnpj_emp": "12345678000190",
        "data_documento": "2026-05-15T10:30:00-03:00",
        "identificador": 9001, "transacao": 9001, "usuario": 42,
        "documento": "000001234", "serie": "001",
        "operacao": "S", "tipo_transacao": "VENDA",
        "cancelado": "N", "delivery": false,
        "itens": [
          { "cod_produto": "PROD-001", "descricao_produto": "Camiseta Branca M",
            "cod_barra": "7891234567890", "quantidade": 2,
            "preco_unitario": 89.90, "valor_total": 179.80 }
        ]
      }
    ],
    "trocas": [
      {
```

```
        "motivo": "Defeito", "doc_venda": "000001234", "serie_venda": "001",
        "cod_cliente": 123, "cnpj_emp": "12345678000190",
        "valor_vale": 89.90, "valor_original": 179.80,
        "timestamp": "2026-05-18T14:22:00-03:00",
        "cod_vendedor": "V001", "quantidade": 1
      }
    ],
    "omni": [
      {
        "id_cliente": 456, "cod_vendedor": "V007",
        "cnpj_emp": "12345678000190",
        "data_documento": "2026-05-20 16:45:00",
        "id_pessoa": 88, "identificador": 3001, "transacao": 3001, "usuario": 42,
        "documento": "ORÇ-00321", "vendedor": "João Silva", "cancelado": "N",
        "itens": [
          { "cod_produto": "PROD-099", "descricao_produto": "Tênis Running 42",
            "cod_barra": "7896543219870", "quantidade": 1,
            "preco_unitario": 349.90, "valor_total": 349.90 }
        ]
      }
    ],
    "vendas_ecommerce": [
      {
        "documento_cliente": "123.456.789-00", "cod_vendedor": "EC001",
        "cupom_vendedor": "VEND10OFF", "codigo_filial": "000001",
        "data_documento": "2026-05-22 09:15:00",
        "pedido_site": "PS-98765", "pedido_wms": "WMS-54321",
        "ticket": "TKT-11111", "documento": "000005678",
        "serie": "001", "valor": "259.90", "qtde": "2"
      }
    ],
    "fallback": null
  }
}
```


Resposta — período excede o limite

```
{
  "success": false,
  "message": "Nenhum registro encontrado no periodo 2025-01-01 - 2025-06-30 para o cnpj 12345678000190",
  "data": null
}
```


Respostas de erro

Parâmetro obrigatório ausente


```
{ "success": false, "message": "O parametro dataInicio é obrigatorio", "data": null }
```


Ação inválida


```
{ "success": false, "message": "Acao nao encontrada", "data": null }
```


Nenhum dado no período


```
{
  "success": false,
  "message": "Nenhum registro encontrado no periodo 2026-05-01 - 2026-05-31 para o cnpj 12345678000190",
  "data": null
}
```


Formato de datas

O formato de data varia por canal na implementação atual. PDV e Trocas convertem datas com `date('c', strtotime($valor))`. Omni e E-commerce retornam a data no formato vindo da consulta, sem conversão explícita para ISO 8601:


```
// PDV / Trocas
"data_documento": "2026-05-15T10:30:00-03:00"
```

```
// Omni / E-commerce
"data_documento": "2026-05-20 16:45:00"
```

Configuração e Dependências

Requires em controle.php

Carregamos três dependências via `require_once(APPPATH . '...')`:

```
require_once(APPPATH . 'controllers/bibliotecas/4cb18a6c-9c9a-4201-82e6-4f6cc3bea445/wosk_webservice.php');
require_once(APPPATH . 'controllers/bibliotecas/592ccfa0-2692-44e3-aaea-cc7c6c6cfcfa/utilOsk.php');
require_once(APPPATH . 'controllers/bibliotecas/0670acec-4aca-43a0-8943-659600407283/db_sqlserver.php');
```

UUID	Arquivo	O que fornece
`4cb18a6c-...`	`wosk_webservice.php`	Trait `WOSK\Commons\WebService` — formata JSON, valida campos e salva logs
`592ccfa0-...`	`utilOsk.php`	Classe pai — conecta ao MySQL e expõe `$this->conexao` (PDO)
`0670acec-...`	`db_sqlserver.php`	Gerencia a conexão PDO com o SQL Server Linx (driver `sqlsrv:`)

Conexão MySQL

Como configuramos

Em `utilOsk`, chamamos `getConfiguracao('db_osk_prod')` — as credenciais ficam em Base64 numa tabela de configuração do sistema. O método `_setConexao()` cria a conexão PDO:

```
// PHP 7.3 — connection string
"mysql:host={$host};port={$porta};dbname={$banco};charset=utf8"

PDO::ATTR_PERSISTENT     = true   // conexão persistente
PDO::ATTR_TIMEOUT        = 120    // timeout em segundos
PDO::ATTR_EMULATE_PREPARES = true
```

Conexão SQL Server

Instanciamos em buscarDadosEcommercePorLoja()

```
// PHP 7.3
$sqlserver = new \db_sqlserver(false, 'LX_ZERO_300', true);
// Parâmetros: ($homologacao = false, $banco, $conectar_agora = true)
// connection string: "sqlsrv:Server=<HOST_SQLSERVER>,<PORTA>;Database=LX_ZERO_300"
```

Ambiente	Host	Porta	Banco	Usuário
Produção	`<HOST_SQLSERVER_PROD>`	`<PORTA_PROD>`	`LX_ZERO_300`	`<USUARIO_SQLSERVER>`
Homologação	`<HOST_SQLSERVER_HML>`	`<PORTA_HML>`	`LINX_HMLG`	`<USUARIO_SQLSERVER>`

Credenciais hardcoded As senhas do SQL Server ficam em `db_sqlserver.php`

Variáveis de ambiente

Variável	Usada por	Descrição
`$_ENV["base_db"]`	`util`	Nome do banco MySQL principal
`$_ENV['WOSK_DB_HMG']`	`WOSK\Common`	Flag de homologação para conexões WOSK (`false` em produção)
`APPPATH`	`controle.php`	Constante CodeIgniter — base dos `require_once`

Logs de requisição

O trait `WOSK\Commons\WebService` salva automaticamente toda chamada no banco **integrador** (`{base_db}_integrador`):

Tabela	Conteúdo
`wosk_webservice`	Registro principal da chamada
`wosk_webservice_parametros`	Parâmetros recebidos (JSON)
`wosk_webservice_retorno`	Resposta enviada (JSON)
`wosk_webservice_callback`	Tokens de callback

Diagnóstico via log Em qualquer dúvida sobre o que retornamos em uma chamada anterior, consultamos `wosk_webservice_retorno` no banco integrador filtrando pelo timestamp e pelo token `<TOKEN_FULLSTORE>`.

Troubleshooting e Testes

Problemas comuns, checklist de diagnóstico e como testar o endpoint manualmente.

Problemas comuns

1. `vendas_ecommerce` sempre retorna vazio

Causas possíveis — verificar em ordem

1. O `cnpjLoja` não tem entidades ativas no MySQL: `SELECT e.* FROM entidade e JOIN juridica j ON e.id_pessoa=j.id WHERE j.cnpj='CNPJ' AND e.situacao='ATIVO'`
2. A filial está como `TIPO_FILIAL = 'FRANQUIA'` no SQL Server — excluímos franquias.
3. A view `W_BI_FAT_VENDEDOR_VDK_OMNI_V0` não tem dados para o período/filial informados.
4. Falha na conexão SQL Server — verificar porta `1435` e extensão `pdo_sqlsrv`.

2. Erro de nenhum registro por período além do limite

Causa

O período informado ultrapassa **200 dias no passado**. A implementação atual monta um `fallback` internamente, mas a resposta final retorna `success:false` com mensagem de nenhum registro encontrado. Com a data de hoje **2026-06-09**, o limite de calendário recua até **2025-11-21**; por causa da comparação com horário atual, use **2025-11-22** como primeira data segura para evitar a borda.

```
// dataInicio: "2025-01-01" → excede → retorna success:false
// dataInicio: "2025-11-22" → dentro do limite com margem de segurança
```

3. Erro `success: false` — campo faltante

Solução

Verificar se o payload contém os três campos dentro de `parametros`: `dataInicio`, `dataFim` e `cnpjLoja`. Valores `null` ou string vazia também disparam a validação.

4. Trocas não aparecem mesmo com devoluções registradas

Causas possíveis

- A troca não está com `situacao = 'FINALIZADO'` — buscamos apenas finalizadas.
- O `tipo_estoque` não é `'TROCA'` — outros tipos de entrada são ignorados.
- A data de emissão da troca está fora do período solicitado.

Checklist de diagnóstico

#	Verificação	Comando / Query
1	MySQL acessível?	`SELECT 1` via PDO
2	CNPJ existe em `juridica`?	`SELECT * FROM juridica WHERE cnpj = 'CNPJ'`
3	CNPJ tem filiais ativas?	`SELECT e.codigo FROM entidade e JOIN juridica j ON e.id_pessoa=j.id WHERE j.cnpj='CNPJ' AND e.situacao='ATIVO'`
4	Existem vendas PDV no período?	`SELECT COUNT(*) FROM movimentacao WHERE modulo='PDV' AND tipo='SAIDA' AND data_emissao BETWEEN '...' AND '...'`
5	SQL Server acessível?	`new \db_sqlserver(false, 'LX_ZERO_300', true)` — exception = falha de conexão
6	View tem dados?	No SSMS: `SELECT TOP 10 * FROM W_BI_FAT_VENDEDOR_VDK_OMNI_V0 WHERE EMISSAO >= 'data'`
7	Log da requisição salvo?	`SELECT * FROM wosk_webservice_retorno ORDER BY id DESC LIMIT 5`

Como testar o endpoint

Via cURL

```
curl -X POST https://SEU_SERVIDOR/bibliotecas/<TOKEN_FULLSTORE> \
  -H "Content-Type: application/json" \
  -d '{
    "acao": "getVendas",
    "parametros": {
      "dataInicio": "2026-05-01",
      "dataFim":    "2026-05-31",
      "cnpjLoja":   "12345678000190"
    }
  }'
```

Forçar erro (ambiente de testes)

```
{
  "acao":   "getVendas",
  "falha":  "teste",
  "parametros": {
    "dataInicio": "2026-05-01",
    "dataFim":    "2026-05-31",
    "cnpjLoja":   "12345678000190"
  }
}
```

Interpretando a resposta

Cenário	success	fallback	Arrays	Ação
Funcionamento normal	true	`null`	Preenchidos	Processar normalmente
Sem movimento no período	false	—	—	Validar se realmente não há dados no período
Período além do limite	false	Não exposto na resposta atual	—	Ajustar datas da consulta
Parâmetro faltando	false	—	—	Corrigir payload
Falha de banco de dados	false	—	—	Verificar conectividade e logs PHP

Regra de ouro: verificar success antes dos arrays Toda integração deve checar o campo top-level `success` antes de processar qualquer array. Na implementação atual, resposta sem nenhum registro retorna `success:false`; arrays vazios só aparecem quando pelo menos o envelope de dados é retornado.