ChatGPT

Shopifyへの発送データ反映API設計に関する調査レポート（2025年10月時点）

1. Shopify発送更新API構成の比較と推奨パターン（2025年）

Fulfillment Orders API（新方式）: 2025年現在、**Shopifyは Fulfillment Orders API ベースのワークフローを推奨**しています。注文が生成されると、自動的に「Fulfillment Order（FO）」という単位でロケーションごとに発送指示が作成されます [1] [2]。開発者はこのFOを取得し、対応する**Fulfillment（実発送）**を作成することで注文の商品を発送済みにできます [3] [4]。Fulfillment Orders APIは部分発送（ラインアイテム単位の一部数量発送）や複数ロケーション発送をネイティブにサポートし、Shopify上で部分発送時に注文ステータスが「Partially Fulfilled」になるなど正確に反映されます [5] [6]。

Fulfillment API（旧方式）: 旧来は**注文API上で直接フルフィルメントを作成**する手法（RESTエンドポイント: `POST /orders/{order_id}/fulfillments.json` など）がありました。しかし**2023年頃に非推奨**となり、Shopifyは新しいFulfillment OrdersベースのAPIへの移行を促しています [7] [8]。古いFulfillment APIはシンプルにラインアイテムIDを指定して発送済みにできましたが、マルチロケーションや部分発送時の制御が不足していました。また**Shopifyは2024年10月以降REST Admin APIをレガシー扱い**としており、**2025年4月以降の新規公開アプリはGraphQL Admin APIのみ使用**する必要があると発表しています [9]。そのため将来的な互換性を考えると、新方式（Fulfillment Orders API）に沿った実装が望ましいです。

Order Editing API: こちらは注文内容（ラインアイテムの追加・削除・数量変更等）を編集するためのAPIです。**発送ステータス更新自体には直接関与しません**。部分発送を行うために注文を分割する、というアプローチも考えられますが、Shopifyでは上記のFulfillment Order機構がその役割を担うため、基本的にOrder Editing APIは不要です。例外的に、発送不能なアイテムを注文から削除する（キャンセル返金する）場合などにはOrder Editing APIやRefund API等が関わりますが、通常の**発送処理にはFulfillment Orders/Fulfillments APIで十分**です。

GraphQL API: 前述のとおり2025年以降はGraphQL利用が推奨されており、GraphQLでも `fulfillmentOrder` や `fulfillmentCreate` ミューテーションを使って同様の操作が可能です。GraphQLでは一度のミューテーションで複数トラッキング番号を扱うこともでき（REST APIでは1フルフィルメント=1トラッキング番号制限）、柔軟性があります [10] [11]。もっとも、本レポートでは理解しやすいRESTエンドポイントで解説します。

推奨パターン: マーチャント管理ロケーションを前提として、注文ごとに自動生成されるFulfillment Orderを取得し、該当アイテムに対するFulfillment（発送）を作成するパターンが基本です [12] [6]。ベンダーが一部のラインアイテムのみ発送する場合でも、Fulfillment作成時に対象とするFOラインアイテムと数量を指定すれば**部分的に発送**できます（残りは未発送のまま同一のFulfillment Orderに残ります）。複数ベンダー＝複数ロケーションの場合、注文にはロケーション別に複数のFulfillment Orderが存在し、それぞれ独立に発送処理を行います。小規模かつロケーション数を増やしたくない場合は**単一ロケーション上で部分発送**しても問題ありません（Shopifyは一つの注文に対して複数回の発送更新＝複数Fulfillment作成をサポートし、その都度「Partially Fulfilled」→「Fulfilled」にステータスが変化します）。

> **補足:** Basicプランでは作成可能なロケーション数に上限があります。複数ベンダーをそれぞれ別ロケーションにする設計は、ベンダー数が上限を超えない範囲であれば可能です。それ以上に分かれる場合はロケーションを共有しつつSKUや内部データでベンダー判別し、単一ロケー

ション内で部分発送を行う方針が現実的です（この場合もFulfillment Orders APIで問題なく対応可能です）。

2. 必要なOAuthスコープ、エンドポイント、リクエスト例（トラッキング番号・配送業者コード含む）

OAuthスコープ: 発送更新には**注文関連の読み書き権限**に加え、Fulfillment Order管理用のスコープが必要です。具体的には、`write_merchant_managed_fulfillment_orders`（マーチャント管理ロケーションのFO権限）は必須です。また将来的に他社提供のフルフィルメントサービスのFOも扱う可能性があるなら`write_third_party_fulfillment_orders`も検討します [13] [14]。一般的な**Order管理アプリ**では上記2つをリクエストすることで**全てのFulfillment Orderを包括的に扱えます** [14]。なお2025年現在でも`read_orders, write_orders`スコープを持っていれば基本的なFulfillment操作は可能ですが、最新のAPI仕様に沿った粒度で権限を与えるため、FO関連スコープの付与が推奨されます。加えて、発送情報取得のために`read_orders`（注文およびその子リソースの参照権限）や、Webhook利用時には`read_fulfillments`相当の権限も必要になる場合があります。自社ストア内の在庫ロケーションを使う限りは上記の「merchant_managed」スコープで十分です。

主要エンドポイント:
- **Fulfillment Order取得:** `GET /admin/api/{version}/orders/{order_id}/fulfillment_orders.json` – 特定注文に紐づくすべてのFOを取得 [15]（部分発送や複数ロケーションでFOが複数存在するケースに対応）。
- **フルフィルメント作成:** `POST /admin/api/{version}/fulfillments.json` – **Fulfillment Orderに対応する発送を作成**します [16]。このエンドポイントで、対象とするfulfillment_orderのIDやトラッキング情報をJSONで渡します。
- **フルフィルメントキャンセル:** `POST /admin/api/{version}/fulfillments/{fulfillment_id}/cancel.json` – 誤って発送を確定した場合に**キャンセル**し、注文を未発送状態に戻すAPIです [17]（UIで「未発送に戻す」操作をした際に利用）。
- **トラッキング情報更新:** `POST /admin/api/{version}/fulfillments/{fulfillment_id}/update_tracking.json` – すでに作成済みのFulfillmentに後から追跡番号や配送業者を追加・変更するAPIです [18]。初回で情報を付与できるなら通常不要ですが、配送業者コードのタイプミス修正や後追い登録に備えて使用できます。

リクエストボディ例: Fulfillment（発送）作成時のJSON例を以下に示します。

```
POST /admin/api/2025-10/fulfillments.json
{
 "fulfillment": {
  "line_items_by_fulfillment_order": [
   {
    "fulfillment_order_id": 1046000818,
    "fulfillment_order_line_items": [
     { "id": 1072503286, "quantity": 1 }
    ]
   }
  ],
  "tracking_info": {
   "number": "SG123456789JP",
   "company": "Sagawa (JA)"
  },
  "notify_customer": true
```

```
  }
}
```

上記では、注文のFulfillment Order（ID: 1046000818）のうち特定のラインアイテム（FOラインアイテムID: 1072503286）数量1点を発送し、追跡番号`SG123456789JP`および配送業者「佐川急便」を指定しています。`notify_customer`を`true`にすると、Shopifyがこの発送に関するメール通知（発送通知メール）を自動送信します [19]。**配送業者コード**は`tracking_info.company`フィールドで指定し、**Shopify公式のサポート社名**を正確に記載します [20]。例えば日本の主要配送業者であれば、Shopify指定の文字列「Sagawa (JA)」（佐川急便）、「Yamato (JA)」（ヤマト運輸）、「Japan Post (JA)」（日本郵便）などを使用します [21] [22]。**大文字小文字や表記揺れも区別**されるため、**リスト通りの表記**にする必要があります [23]。公式名に一致する会社を指定した場合、Shopify側で**自動的に追跡URLが生成**され、注文詳細画面でクリック可能になります [21]。また日本郵便・ヤマト・佐川など主要キャリアであれば、配達状況に応じてShopifyが`shipment_status`（配送状況）を自動更新してくれるため（例：「in_transit」「out_for_delivery」「delivered」等）運用上も便利です [24] [25]。万一、マイナーな運送会社や社内便等で**公式サポート外**の場合は、`tracking_info`に`url`を含めて追跡ページのURLを直接指定します（URLを指定すれば未知の会社名でもそのリンクが管理画面に表示されます） [10]。なお、REST API経由では一度のFulfillment作成につき**追跡番号は1つまで**ですが、複数荷物に分割発送する場合は**Fulfillment自体を分けて複数回POST**するか、後から`update_tracking`で追加登録します [11]。GraphQL APIでは複数追跡番号をまとめて登録することも可能ですが、本システムでは基本RESTで逐次送信すれば問題ありません。

補足: 上記JSONでは`fulfillment_order_line_items`を明示的に指定しています。これは**部分発送**時に必要で、もしFulfillment Order全体を丸ごと発送する場合（そのFOに含まれる全ラインアイテムを一括発送する場合）は、この配列を省略して`{"fulfillment_order_id": 1046000818}`と書くだけで**全数量を発送**できます [26] [27]。Shopifyはリクエストを受け取ると、指定FO内の該当ラインアイテムのfulfillable数量を減算し、Fulfillmentオブジェクトを作成します。以降、注文ステータスや残りのFOの状況は自動で更新されます。

3. Supabaseデータ（shipments 等）とShopify APIのマッピング

本システムではSupabaseに以下のようなテーブルを持つと想定されます: **`orders`**, **`line_items`**, **`shipments`**, **`shipment_line_items`**。ShopifyからWebhook/APIで同期した**`line_items`テーブルには Shopifyの `line_item_id`**（注文内ラインアイテム固有ID）が保存されており、各行にSKUや数量などが入っているでしょう。**`shipments`**はベンダーごとの発送情報（追跡番号や配送会社、発送日など）、**`shipment_line_items`**はその発送に含める商品行（複数行）を関連付ける中間テーブルです。**SKU**は各商品変種を一意に識別する社内管理コードですが、Shopify APIとのやりとりでは**直接SKUは使用せず**、SKUを元に突き止めた**`line_item_id`**や関連IDを用います [28]。

マッピング戦略: Shopifyへの反映時は、まず対象注文のFulfillment Orderを取得し、その中から**発送対象とするラインアイテムを特定**します。具体的には以下の手順になります。

1. **注文のFO取得:** 該当注文のShopify注文IDを元に、API経由でその注文の全Fulfillment Orderを取得します [15]。レスポンスには各FOのID（`fulfillment_order.id`）と、含まれるラインアイテム一覧（`fulfillment_order.line_items`）が含まれます。
2. **ラインアイテム対応付け:** 各FOの`line_items`配列内に、Shopifyの`line_item_id`フィールドが含まれており、元の注文のラインアイテムと対応付けられています [29]。Supabase上の`shipment_line_items`が持つ`line_item_id`と照合し、**そのラインアイテムが属するFO ID**および**FO内でのラインアイテムID**（`fulfillment_order_line_item.id`）を取得します。例えば、FOのJSON一例では次のようにオリジナルの`line_item_id`とFO固有のラインアイテムIDが対応しています [29]:

```
"fulfillment_orders": [
 {
  "id": 2613406630044,
  "status": "open",
  "line_items": [
   {
    "id": 7383807459484,
    "fulfillment_order_id": 2613406630044,
    "quantity": 1,
    "line_item_id": 6151601488028,
    "inventory_item_id": 38438906921116,
    "fulfillable_quantity": 1,
    ...
   }
  ]
 }
]
```

上記ではShopify注文内ラインアイテムIDが `6151601488028` のアイテムが、このFOではラインアイテムID `7383807459484` として含まれていることが読み取れます。**SKU**はこの対応付けには使われませんが、SKUから該当する `line_item_id` を引くこともできます（本システムでは注文同期時にSKUからベンダー振り分けをしているため、SKU→ベンダーの対応は内部で完結しており、ShopifyAPI呼び出し時にはline_item_idベースで処理するのが確実です [30] [31] ）。

1. **FOとshipmentの対応:**　通常、**1つのShipment（ベンダーの発送単位）につき1つのFulfillment**をShopifyに作成します。Shipmentに紐づくすべてのラインアイテムが**同一のFulfillment Orderに属する場合**（例えば単一ベンダーの発送で、該当アイテム群が同じロケーションから発送される場合）、そのFOに対してまとめてFulfillmentを作成できます。一方、もし1つのShipmentに含まれる商品が**複数のFOに跨る場合**（基本的に起こらない設計が望ましいですが、例えば異なるロケーションの在庫を一つにまとめて発送するような特殊ケース）、Shopify上はそれぞれのFOごとにFulfillmentを別送信する必要があります。同じFulfillment作成API呼び出しで複数のFOを同時に指定することもできますが、**すべて同一注文・同一ロケーションのFOでなければエラー**になります（異なる注文やロケーションを混ぜることは不可） [32] [33] 。

2. **数量のマッピング:** `shipment_line_items` には各ラインアイテムの発送数量が入っているはずです。通常はそのラインアイテムの未発送数量すべてを発送するでしょうが、**一部数量のみ発送する場合**もあります。ShopifyのFOラインアイテムには `fulfillable_quantity` （未発送残数）が管理されています [34] [35] 。Fulfillment作成時に指定する `quantity` は、今回発送する個数です。例えば在庫不足で本来5個中2個のみ先に発送する場合、`fulfillment_order_line_items: [{ "id": <FO_line_item_id>, "quantity": 2 }]` のように指定します。送信後、Shopify側では当該FOラインアイテムの `fulfillable_quantity` が3に減り、FOはまだ「open」（未完了）のままとなります。残り3個が後日発送可能になったら、同じFOに対し再度Fulfillment　APIを呼び出して残りを発送します。**Shopifyは1つのFOにつき複数回のFulfillment作成を許容**しており、そのたびに残数量を管理します [36] [11] 。

3. **その他のフィールド:** `shipments` テーブルの追跡番号や配送会社は前項の `tracking_info` にマップします。ベンダーやSKU情報はShopifyには送らず、あくまで自社システム内で管理するメタ情報です（Shopify管理画面上では各Fulfillmentに「担当ロケーション」や「Fulfillmentサービス」が表示されますが、Basicプランでは通常「販売元（マーチャント名）」または「手動」として扱われます [37]

[38] ※Advanced以上でロケーション名表示）。SKUはShopifyの商品管理にも存在する項目ですが、発送APIでは利用せず、主キーとしての役割はline_item_idやvariant_idが担っています [39]。

まとめると: Supabaseで管理している**line_item_id**をキーに、**FulfillmentOrder ID**と**FulfillmentOrder Line Item ID**を取得し、**shipment_line_itemsの数量**と**shipmentsの追跡情報**を組み合わせてShopifyのFulfillment作成APIに送る、というマッピングです。このときSKUは内部照合に留め、Shopify APIには登場させない形となります [28]。適切にマッピングすれば、**SKU単位での配送ステータス同期**が可能です [40]（一商品が複数発送にまたがる場合でも、line_item_idとFOラインアイテムIDで一意に管理されます）。

4. 「発送済み」ステータス更新のワークフローとリアルタイム連携

ベンダーが管理UI上で注文のステータスを「発送済み」に切り替えたタイミングで、即座にShopifyに反映させるフローを設計します。重要な点は**即時性**と**信頼性**であり、ユーザー操作に対するフィードバックを早く返しつつ、通信エラーなどによる不整合を防ぐことです。

基本ワークフロー:
1. ベンダーがUIで「発送済み」ボタンをクリック。追跡番号や配送会社を入力して確定します。
2. フロントエンドはバックエンド（Next.js APIルートやSupabase Edge Functionなど）にその情報を送信します。バックエンドではまず**Supabaseのデータを更新**し、該当 `shipments` および `shipment_line_items` のステータスを「shipped」にします。
3. 続いてバックエンドからShopify Admin APIに対して前項のマッピングに従った**Fulfillment作成リクエスト**を送ります。
4. **Shopify APIレスポンスを確認**し、成功した場合はSupabase側に「同期済み」フラグを立てたり、発送日時を記録します。フロントエンドには成功レスポンスを返し、ユーザーには「Shopifyに同期完了」等の通知を表示します。
5. **エラー発生時**（ネットワーク不通、Shopify APIからエラー応答 など）は、バックエンドでエラーログを記録し、フロントエンドにはエラー内容を返します。UI上でユーザーには「同期に失敗しました：○○（エラー内容）」等と知らせ、必要に応じて**再試行（リトライ）**ボタンや案内を出します [41]。

エラーハンドリング:
- Shopify APIからの典型的なエラーとして、**ステータス401/403（認証・権限エラー）**、**404（対象リソースなし）**、**422（リクエスト内容不備）**、**429（レート制限超過）**などが考えられます [42] [43]。例えば「Not Found」エラーはFO ID間違いや権限不足で発生しうるので [44] [45]、まず**FO取得→ID確認**のプロセスを踏み、正しいID・権限でリクエストすることが重要です。また、429レート制限に対しては**指数バックオフ付きのリトライ**を実装します（Shopify推奨は**1秒以上の待機**です [46]）。自動再試行は3回程度までとし、それ以上失敗する場合はユーザーに手動再送を促す設計が望ましいです [47]。
- エラーが発生した場合、**UI上のステータスは「未同期」状態**としておきます。例えば「発送済み（未反映）」のようにマーキングし、ユーザーが再送ボタンで再試行できるようにします。再送時にはエラー内容（例: 権限エラーなら管理者にアプリ再インストール促す、422ならデータ不整合をチェックする等）を考慮したメッセージを出すと親切です。

リトライ方法:
- **手動リトライ:** ユーザーが「再送信」ボタンを押すと再度API送信を試みる。特にCSV一括処理時などは、失敗した行だけ再送するUIを用意すると便利です。
- **自動リトライ:** 瞬間的なネットワーク障害などの場合、自動で数回リトライすることも検討します。ただし連続失敗時は以降成功しない可能性が高いため、無限ループを避けて**上限回数を決める**べきです [47]。また、バッチ処理で大量注文を一括同期する場合、一件失敗が他の処理をブロックしないよう非同期キュー化する手法もあります。SupabaseならDBの行にフラグを立ててCronやTriggerで後ほど処理することもできます。

UI上の考慮:
- ユーザーには**リアルタイムに結果を通知**します。成功時は「Shopify側も発送済みに更新されました（メール通知も送信済み）」など、失敗時は「同期失敗: ○○。再試行してください。」のようにフィードバックします [41]。
- 再送ボタンは、エラー内容に応じて**活性/非活性**や**注意文言**を変えると良いでしょう。例えば権限エラーであれば「管理者にお問い合わせください」、一時的なエラーなら「再送する」で良い、といった案内です。

Webhookやイベント駆動の代替策:
今回のケースでは、基本的に当システムがデータの主導権を持っており、Shopifyへアウトバウンド連携する流れです。そのためShopifyのWebhookは直接役立ちませんが、**設計次第ではイベント駆動型の同期**も可能です。例えば、Supabaseの行変更トリガーを用いて、`shipments`テーブルのステータスが「shipped」に更新されたら自動でShopify API送信を行うサーバーレス関数を走らせることもできます。この方法ならフロントエンドから直接Shopify通信せずにすみ、エラー時の再送キュー管理もバックエンドで一元化できます。ただし即時性は若干低下する可能性があります（トリガー処理の遅延や失敗を考慮する必要あり）。現状MVPではユーザー操作後すぐ結果を出す同期方式で問題ないと考えられますが、将来的に注文量が増えた場合や完全自動化する場合には、こうした**非同期キュー + バックグラウンド処理**への移行も視野に入れてください。

また別アプローチとして、Shopify側で**Fulfillment Service**を登録し、Shopify→アプリへのFulfillment Order生成通知・承認リクエストを受けてから処理する方法もあります [48]。具体的には各ベンダーをShopifyの「カスタム発送サービス（FulfillmentService）」として設定し、注文発生時にShopifyから当アプリにFO作成のWebhook（もしくは登録Callback）を送り、アプリ側でベンダーUIに反映→ベンダー承認後、アプリがFulfillment APIで確定させるというフローです [49]。この方式だとShopify上でも「リクエスト送信中」などのステータス管理ができますが、**実装が複雑**になるのと、現在の要件では自社システム主導で十分対応可能なため、MVP段階では採用しなくてよいでしょう。ただ、**将来的な拡張**としてShopifyのfulfillment_orders系Webhook（例: `FULFILLMENT_ORDER_CREATE`, `FULFILLMENT_ORDER_HOLD`, `FULFILLMENT_ORDER_FULFILLED`等）を購読し、Shopifyと在庫連携や自動承認を行うことも検討できます [50]。

5. Shopify APIからの代表的なエラーと運用上の制限事項

Shopify側への発送同期で遭遇し得るエラーや、設計段階で留意すべきポイントを列挙します。

- **権限エラー (401/403 Forbidden):** アクセストークンやOAuthスコープが不十分な場合に発生します。例えばFulfillment Order関連のスコープを付与していないとFO取得やFulfillment作成で403エラーとなります [42]。解決策は必要なスコープを追加付与して再認可することです [14]。Basicプランでもアプリのスコープには影響ありませんので、遠慮なく要求して問題ありません。

- **リソース未検出エラー (404 Not Found):** 指定した注文IDやFO IDが存在しない場合、あるいは**アプリにそのリソースを読む権限がない場合**に起こります [51] [52]。たとえばFO ID間違いで`{"errors": "Not Found"}`というレスポンスが返るケースがあります [51]。この場合は前述のとおり、まず注文IDから正しいFO IDを取得して使用する、権限も含め確認することが必要です。

- **検証エラー (422 Unprocessable Entity):** リクエストボディの内容に問題がある場合のエラーです。Shopify APIでは意味的に不正な操作も422で返されることがあります [53] [54]。代表的な例:

- **「Cannot create fulfillment for the fulfillment order」**というエラー: 対象のFulfillment Orderが現在Fulfillment作成できない状態の場合に起こります [55] [49]。典型的には、そのFOが**第三者ロケーションに割り当てられていて承認待ち**（`request_status: submitted`）の場合です [56] [48]。このケースではFulfillment Service側が`accept_fulfillment_request`操作を行わないとFulfillmentを作

れません [48] 。本設計ではマーチャント管理FOを使う想定のため基本起こりませんが、もし将来外部FulfillmentServiceと連携する際には留意してください。

- **「The fulfillment order is not in an open state」エラー:** 既にクローズ済み（完了またはキャンセル済み）のFOに対してFulfillmentを作成しようとすると起こります [57] 。例えば一度全品発送してFOがcloseされた後に再度Fulfillment APIを叩くとこのエラーになります。運用上は**同じ注文に対して二重送信しない**ように気をつけ、万一発生した場合はアプリ側で「既に同期済み」とみなしてエラーを無視してよいでしょう。

- **配送業者名の不備:** `tracking_info.company` にサポート外の値を入れてもエラーにはならず無視されますが、`tracking_info.number` が極端に長すぎる等フォーマット不正の場合は422になります。運送会社コードは先述のリスト通りであれば問題ありません。

- **レート制限エラー (429 Too Many Requests):** 短時間にAPIを呼びすぎた場合に発生します。BasicプランのAPIレートはRESTで**40リクエスト/分（2リクエスト/秒のリークレート）**程度が上限です [58] 。CSV一括同期などで大量のFulfillment作成を一度に行うと429になる可能性があります。この場合、**1秒以上の待機を挟んで再試行**する必要があります [46] 。対策として、連続リクエストにはキューイングと**指数バックオフ**を導入してください [47] [59] 。幸い、発送更新は各注文ごとに独立した処理なので、逐次実行でも大きな問題にはなりません。ユーザーには大量の発送同期を指示した際でも、バックグラウンドで順次処理し進捗を表示するようなUIにすると安心です。

- **重複発送の制約:** 同じ注文の同じラインアイテムを重複してFulfillmentすると、2回目のリクエスト時に前述の422エラーが出ます（既にfulfillable_quantityが0のため） [60] [61] 。運用上、**一度発送済みにしたものを再度発送済みにしようとしない**ことが重要です。UI上で「未発送に戻す」を実行した場合は、まずShopify側で該当Fulfillmentをキャンセルする処理を行ってからでないと、再度fulfillable_quantityが戻らず発送できません。したがって、「未発送に戻す」ボタンのアクションでは `fulfillments/{id}/cancel.json` を呼び出し、成功したらSupabase側もステータスを戻すようなフローにします [62] 。なお、キャンセルしたFulfillment Orderは再び「open」状態になるので、残量分について後からFulfillment可能になります。

- **配送会社コードの扱い:** 先にも触れましたが、**Shopifyが認識する運送会社名**であれば自動追跡リンク生成・配達状況更新が行われます [21] 。日本の「ヤマト運輸」「佐川急便」「日本郵便」はいずれもサポートリストに含まれており、それぞれ `"Yamato (JA)"`, `"Sagawa (JA)"`, `"Japan Post (JA)"` を指定可能です [22] [63] （※末尾の(JA)は日本語表記対応の意味ですが、システム上どちらを使っても追跡URL生成自体は同じです）。もし誤った値を入れるとリンク生成されないだけでなく、Shopify側には「その他」の配送業者として扱われ、顧客に送信されるメールにも追跡リンクが載りません。必ず正確な値を使うよう注意してください。また、tracking_number自体はShopify内でユニーク制約等はありませんが、同一注文内で全く同じ追跡番号を重複登録すると混乱を招くため、通常はユニークになるはずです。万一間違った番号で送信した場合、`update_tracking` エンドポイントで後から修正可能です [64] 。

- **Deprecated API呼び出し:** 旧来の `/orders/{id}/fulfillments.json` などのエンドポイントは上述の通り非推奨であり、**2023-04以降のAPIバージョンでは利用すると警告が出る**可能性があります [7] 。一部のコミュニティ記事では非推奨APIを引き続き使う裏技的な言及もありますが [65] 、将来的にサポート終了するリスクがあります。可能な限り新しいFulfillmentOrder+Fulfillmentのフローを採用してください。BasicプランだからといってDeprecated APIを使わなければ実現できない機能は特にありません。

- **その他制限事項:** Shopify側で一つの注文に作成できるFulfillment（部分発送）の**最大数に明確な上限は公表されていません**が、非常に細かく分割すると管理も煩雑になるため、運用的に現実的な範囲で分割してください。また、Fulfillment作成後の履歴は消去できない（キャンセルは可能ですがログは残る）ため、テスト環境ではダミー注文で検証し、本番では不要な発送操作をしないよう注意しま

しょう。Shopifyの管理画面上でも部分発送や再発送の操作は制限されています（例：注文編集でアイテムを追加すると新たなFOができるが、既存Fulfillmentには影響しない等）。こういった管理上の仕様も把握しておくとトラブルシューティングに役立ちます [66] [67]。

6. 参考資料（公式ドキュメント・コミュニティ記事・実装例）

本調査で参照した情報源および実装の指針となるドキュメントを以下にまとめます。

- **Shopify公式APIリファレンス: FulfillmentとFulfillmentOrder** – REST Admin APIにおけるFulfillment関連エンドポイントの最新仕様 [26] [68]。FulfillmentOrderの自動生成やライフサイクルについても記載 [1] [2]。特に「FulfillmentOrderは注文作成時に自動生成され手動作成不可」である点は重要 [2]。また、トラッキング情報のフォーマットやサポート運送会社一覧 [20] [22]、APIエラーコード一覧 [53] [54]も参照しました（2025-10版ドキュメントに基づく）。

- **Shopify公式開発者ドキュメント: APIアクセススコープ** – Fulfillment Orderに関連する権限スコープの説明 [13] [14]。Order管理アプリは`write_merchant_managed_fulfillment_orders`等を要求すべきと明示されており、本アプリのOAuth設定にも反映済みです。

- **Shopifyコミュニティ Q&A**:

- 「FulfillmentOrders APIへの移行相談」 [69] [8] – 旧API廃止に伴う疑問への回答。旧APIは簡潔だが新APIへ段階的に移行する必要がある旨示唆されています。
- 「Fulfillment Orderの422エラー事例」 [55] [49] – Fulfillment Service未承認時に発生するエラーの共有と解決策（GraphQLの`acceptFulfillmentRequest`を使う必要がある点） [48]。
- 「Not Foundエラーの原因」 [70] [52] – FO IDの指定ミスや権限不足によるエラー時のチェックポイントが参考になります。

- その他、部分発送に関する質疑応答（SKUマッチング不要になった等） [71] やREST APIでの複数追跡番号問題などもコミュニティ上で議論されています。

- **HulkAppsブログ「Mastering Order Fulfillment on Shopify」** [12] [6] – Fulfillment Ordersを使った部分発送や追跡情報更新についてのFAQ形式の解説。FO ID取得方法（注文IDからFulfillmentOrderをGET）や部分Fulfillmentの可否など、本設計にマッチする情報が掲載されています。一般的なベストプラクティス（エラー処理や在庫連携）も網羅されており、実装時のヒントになります。

- **Scale Shopifyブログ「Error Handling in Shopify API: Best Practices」** [42] [47] – 2025年3月公開の記事で、Shopify API利用時のエラー処理について詳述。認証エラーや422/429の対処、リトライ戦略（バックオフ）など本システム実装にも役立つテクニックが紹介されています。特に「一度の不備が連鎖的な問題を引き起こしうるので丁寧なハンドリングを」との指摘は肝に銘じるべきでしょう [72] [73]。

- **Shopify公式ブログ「Automated Order Fulfillment (2025)」** [74] – フルフィルメント業務の自動化に関する記事。Fulfillment Orders APIを使えば外部の物流システムと受注・在庫・発送データをシームレスに連携できる、といった記述があり、本システムのような**受発注プラットフォームとShopifyの橋渡し**にも当てはまります [75]。直接的な実装コードはありませんが、コンセプト整理の参考になります。

以上の情報は**2025年10月時点**で最新のものを参照しています。それぞれ信頼性の高い公式ドキュメントや実績のある開発者コミュニティの知見に基づいており、本設計・実装を進める上で有用な根拠となります。不明

点が出た際は、該当する公式ドキュメントの最新版やパートナー向けフォーラムを再確認し、将来的なAPI変更にも対応できるよう留意してください。 9 14

---

1 2 3 4 13 14 15 34 35 50 68 FulfillmentOrder
https://shopify.dev/docs/api/admin-rest/latest/resources/fulfillmentorder

5 6 12 66 67 Mastering Order Fulfillment on Shopify: How to Use the Shopify API to
https://www.hulkapps.com/blogs/shopify-hub/mastering-order-fulfillment-on-shopify-how-to-use-the-shopify-api-to-create-fulfillment-orders

7 8 32 33 65 69 Using Fulfillment Orders API instead of Fulfilment API - Shopify Community
https://community.shopify.com/t/using-fulfillment-orders-api-instead-of-fulfilment-api/174557

9 10 11 16 17 18 19 20 21 22 23 24 25 26 27 36 37 38 39 60 61 62 63 64 Fulfillment
https://shopify.dev/docs/api/admin-rest/latest/resources/fulfillment

28 11_requirements_bridge_app.md
file://file_00000000405c61fa8e424859c944676d

29 48 49 55 56 422 Error Creating Fulfillment for Order - Shopify Community
https://community.shopify.com/t/422-error-creating-fulfillment-for-order/4590

30 31 40 sku-vendor-spec.md
file://file_000000009e0861faa241d66a1ca75f08

41 10_requirements_app.md
file://file_00000000895861fa95f8ef3675dbca84

42 43 47 58 59 72 73 Error Handling In Shopify API: Best Practices | Scale Shopify
https://scaleshopify.com/2025/03/03/error-handling-in-shopify-api-best-practices/

44 45 51 52 70 Why am I getting a 'Not Found' error when fulfilling an order via API? - Technical Q&A - Shopify Community
https://community.shopify.com/t/why-am-i-getting-a-not-found-error-when-fulfilling-an-order-via-api/218616

46 Shopify API limits
https://shopify.dev/docs/api/usage/limits

53 54 Shopify API response status and error codes
https://shopify.dev/docs/api/usage/response-codes

57 REST Admin API reference - Shopify developer documentation
https://shopify.dev/docs/api/admin-rest

71 New Workflow for fulfilmentOrders with Tracking Information - #5 by prg74 - Shopify Community
https://community.shopify.com/t/new-workflow-for-fulfilmentorders-with-tracking-information/188921/5

74 75 Automated Order Fulfillment: Streamlining Order Processing (2025) - Shopify
https://www.shopify.com/blog/automated-order-fulfillment