ChatGPT

Indents and Vendor Portal – FLM vs. MM (IST) Workflows

Overview

In **ProjectX**, an *indent* represents a request sent to external vendors to assign a vehicle (and driver) for a trip. The **Vendor Portal** allows vendors to view and respond to these indents. There are two main trip categories: - **FLM Trips** (often called "endpoint" trips in code, e.g. first/last mile or direct shipment trips). - **MM (IST) Trips** – Inter Stock Transfers (internal multi-modal or milk-run movements).

The indent handling is similar for both, but runs on separate models (regular Trip vs. InterStockTransfer) with some differences in flow, validations, and configuration. Below, we detail the end-to-end indent process for each type, covering how indents are generated, how vendors interact via the portal (accepting or rejecting requests and assigning vehicles), and what validations and rules apply at each step.

FLM Trip Indent Workflow (Endpoint Trips)

1. Indent Creation and Vendor Notification (Auto-Allocation)

When an FLM trip is created and requires a vendor's vehicle, the system automatically generates an **indent** to one or more eligible vendors: - **Vendor Selection:** The trip's required vehicle type and route are used to find serviceable vendors. The code builds a priority list of vendors (optionally filtering by vendor capacity) [1] [2]. A `VendorTrip` entry is created for each invited vendor with status **"created"** [3] [4]. This marks an active indent request for that vendor. (The `VendorTrip` model tracks vendor-trip links; `tripStatusEnum` defines states like *created*, *accepted*, *assigned*, *rejected* [5].) - **SMS Notification:** For each vendor, an SMS is sent containing a link to the Vendor Portal and trip details. The system uses a template URL (`VENDOR_PORTAL_URL`) with the organisation ID injected [6] [7]. If multiple trips are bundled in one indent (not typical for FLM), they are listed; otherwise a single trip reference number is included [8] [9]. The SMS content includes origin hub/location and possibly destination or vehicle details, directing the vendor to respond on the portal. An **expiration alert** SMS will also be scheduled a bit later (see step 4).

Validation: Only vendors with available capacity for the required vehicle type are invited. The system checks a vendor's allotted quantity for that vehicle type; if none available, that vendor is skipped [10]. If no vendors are found for the vehicle type, the indent process does not proceed [11].

2. Vendor Portal – Viewing and Responding to Indents

On the Vendor Portal, an authenticated vendor can see all active vehicle requests (indents) sent to them: - **Listing Requests:** Vendors use endpoints like `/vehicle-requests/get` to fetch pending requests. The backend fetches all active `VendorTrip` records for that vendor (both trip IDs and IST IDs) [12] and returns details such as trip reference number, origin/destination, etc., with pagination and filtering [13] [14]. (The portal supports filters and "bucket" counts – e.g. how many requests are new, accepted, etc. [15] [16] – but all indents initially appear in a pending state.) - **Vendor Actions:** For each

indent, the vendor typically has options to **Accept**, **Reject**, or proceed directly to assign a vehicle. These actions correspond to specific API endpoints: - **Accept Indent:** `POST /indent/event/accept` – indicates the vendor is willing to fulfill the request (without yet providing vehicle details). This is used only if *Indent Acceptance* is enabled in the configuration [17]. - **Reject Indent:** `POST /vehicle-requests/reject` – the vendor declines the request with a reason. - **Assign Vehicle and Worker:** `POST /assign/vehicle` – provide vehicle registration and driver details to fully commit to the trip.

The Vendor Portal APIs enforce required fields and vendor authenticity: - The vendor must be logged in (each endpoint is protected by `authenticateVendorRemote` hooks) [18]. - The **accept/reject** calls require the trip ID and will error if the request isn't actually pending for that vendor (back-end double-checks that the `VendorTrip` is active and corresponds to that vendor) [19] [20]. - The **assign vehicle** call requires all details: vehicle number, driver name, contact, and license expiry in `YYYY-MM-DD` format [21] [22]. Missing or badly formatted fields will be rejected with an error.

3. Indent Acceptance and Timeout Logic

Indent Acceptance is a configurable step. If enabled, the workflow is two-stage (Accept then Assign): - When a vendor hits **"Accept"** on the portal, the system marks that vendor's indent as *accepted*. The `VendorTrip` status is updated to **"accepted"** [23] and an `IndentAcceptedEvent` is recorded for audit [24]. This acceptance does **not** yet assign a vehicle; it signals intent. Other vendor requests for the same trip remain active but the system will pause auto-allocation to the next vendor because one vendor has accepted [25] (the auto-allocation job checks if any vendor has accepted and, if so, will not immediately reassign [25]). - **Timeout for Assignment:** Once accepted, the vendor is given additional time (configured as `indent_expiry_time`) to actually provide vehicle details. The system cancels the original auto-allocation timer and schedules a new one for the remaining time window [26] [27]. Specifically, it calculates how much of the total expiry time is left since the indent was first offered [28] and enqueues a new job `ASYNC_AUTO_ALLOCATE_PRIORITY_VENDOR` with a `skipIndentAcceptance: true` flag [29] [30]. This job will trigger re-allocation if the vendor fails to assign a vehicle in time (see step 4). An expiration alert SMS is also scheduled to warn the vendor before the final expiry [31] [32].

If **indent acceptance is disabled** (the configuration `enable_indent_acceptance` is false), the vendor must directly assign a vehicle without a preliminary accept step. In that case, the portal likely guides the vendor straight to the assignment form. The system's indent timer (`indent_expiry_time`) then serves as the total time allowed for the vendor to assign a vehicle. The back-end will treat a lack of assignment by expiry as a failure and move to the next vendor.

Important validations: The accept call will fail if the organisation has not enabled indent acceptance [17]. Also, if the indent request already expired or was canceled, the system returns a "Vehicle Request expired" error when validating the trip/vendor relation [33] [34].

4. Vendor Assignment of Vehicle & Driver (Fulfillment)

Once ready, the vendor provides the vehicle and driver details via the **Assign Vehicle** API. This is the critical step that finalizes the indent: - **Data Validation:** The system re-validates the trip status and vendor's right to assign. It ensures the trip is still pending assignment and the `VendorTrip` is active for that vendor [35] [33]. It also runs any pending business validations (e.g. consignment or document checks) associated with the "indent accepted" stage [36] [37]. If any pre-assignment validations fail (for example, if the load was canceled by the client in the meantime, or required documents are missing), the assignment is blocked. - **Vehicle Selection:** The system looks up the provided

`vehicleRegistrationNumber` or internal `vehicleId`. If the vehicle record exists but is linked to a different vendor, it updates the vehicle to belong to the current vendor (this allows vendors to add new trucks on the fly) [38]. If the vehicle doesn't exist, a new vehicle record is created for the vendor [39]. - **Driver (Worker) Handling:** If a `workerId` is given, that existing driver is fetched and must exist under the organisation [40]. If only details are given, the system creates a new worker or updates an existing one by `workerCode` [41]. The worker's license expiry is expected in proper date format (validated earlier) and stored. - **Uniqueness Checks:** The back-end ensures the same vehicle isn't already assigned to another active trip. For FLM trips, it tracks assignments through a **Medium** entity (linking a vehicle and driver): if the driver is currently engaged in a different trip, or their associated medium has a `current_trip_id`, the assignment might be disallowed. Specifically, if the worker was free (no ongoing trip), the system associates them to the new trip's hub; if they had a "home hub" and were free, it updates their hub to the trip's origin hub for this assignment [42] [43]. This ensures the driver is now tied to the correct origin location. (For IST, a similar check is done: see differences below.) - **Assigning to Trip:** Finally, the trip is marked as assigned to the vendor's vehicle/driver. Internally, this involves calling the Trip management logic to attach the vehicle and worker to the trip tasks. The `VendorTrip` status is updated to **"assigned"** [44] [45]. An `IndentAssignedEvent` is recorded for the trip [44], which can trigger downstream processes (e.g. notifying the client that a vendor has been secured). The trip's `third_party_vendor_id` field is set to this vendor (so the trip is now associated with a specific vendor) [46].

Upon a successful assignment, the system sends a confirmation SMS to the vendor (and possibly the vendor's parent account) to acknowledge that the vehicle has been assigned. This uses a "Vehicle Request Accepted" SMS template including details of the truck and driver that have been assigned [47]. At this point, the indent cycle for this trip is complete – the trip will proceed with the assigned vehicle.

5. Rejection, Cancellation, and Expiry Handling

An indent might not always end in assignment. The system handles the following outcomes: - **Vendor Rejects:** If the vendor explicitly rejects the indent via the portal (providing a rejection reason), the back-end immediately marks that vendor's `VendorTrip` as **rejected** and inactivates it [48]. A `IndentRejectedEvent` is logged [49]. Crucially, the system will **trigger re-allocation** to the next vendor without waiting for the original timeout: - It consults the auto-allocation job that was scheduled for this trip, removes this vendor/trip from that job's data, and prepares a new job payload targeting the next priority vendors [50] [51]. It calls the auto-allocation processor immediately within the same transaction [52], passing `skipIndentAcceptance: true` to skip waiting (since a vendor already rejected). This will create new indents to the next vendor(s) in line [52]. Essentially, a rejection fast-forwards the allocation to the next candidate. - The vendor's active indent is deactivated (no longer shown in their portal list once the rejection is processed). The rejection reason and a timestamp are stored in the `VendorTrip.extra_details` for record-keeping [53]. - **Indent Expiry (Vendor No-Response):** If a vendor neither accepts nor rejects within the initial **acceptance timeout** window, the scheduled **auto-allocation job** (`ASYNC_AUTO_ALLOCATE_PRIORITY_VENDOR`) will fire. This job will consider the vendor as unresponsive and move to the next vendor on the priority list [54] [55]. Before doing so, it "deactivates" the current vendor's indent (marking it rejected due to timeout) and notifies them that the request expired. The system sends a *vehicle requisition de-requisition SMS* to the vendor informing that the request expired due to no response [56] [57]. Then it proceeds to allocate the next vendor(s) in the list, similar to a rejection flow. - Additionally, halfway through the expiry period, the **Indent Expiration Alert** job triggers. By default ~12 hours (configurable) before the final expiry, the system sends an SMS reminder to the vendor with the indent's reference and the expiration deadline [32] [58]. This serves as a warning: *"Your pending vehicle request will expire by [time] if not accepted."* For IST, a similar alert is sent (the logic is analogous, using the IST reference number) [59] [60]. - **Client Cancels Indent:** In some cases, the shipper or system may cancel an indent (e.g. the load was canceled

or they found a vehicle elsewhere). The **Cancel Indent** API (`POST /trip/event/indent_cancel`) allows an integration or client dashboard to cancel the vendor request [61] [62]. On cancel: - The system finds the trip by reference number and ensures it has an active third-party vendor request [63] [64]. It then marks all active `VendorTrip` invites for that trip as **rejected** (or simply inactive) with a reason [65]. It clears the `third_party_vendor_id` from the trip's data [46] (since the trip is no longer vendor-assigned). - If a vehicle or driver had been tentatively assigned, it unassigns them from the trip (calls the internal routine to detach worker/vehicle from the trip tasks) [66]. An `IndentCancelledEvent` is emitted for audit [67]. - The net effect is the indent process is halted. No new vendors will be auto-allocated (the scheduled jobs would be removed or no-ops), and vendors will no longer see the request. If a vendor tries to accept/assign after cancellation, the `validateTrip` check will throw a "Vehicle Request expired" or invalid trip error [33] [68].

MM (IST) Trip Indent Workflow (Inter-Stock Transfers)

For **MM (IST) trips**, the indent workflow is conceptually similar but involves the **InterStockTransfer (IST)** model and has a few key differences in handling:

1. Indent Generation for IST

When an IST is created requiring an external transporter, the system raises indents to vendors in a manner analogous to FLM: - **Vendor Matching:** The IST handler finds vendors based on the route's origin/destination hubs and required vehicle type. For example, it fetches the origin and destination city codes for the IST route and ensures they are available [69]. Vendors serviceable on that route and vehicle category are retrieved. If a special carrier allocation mapping is enabled, it may use a priority mapping list; otherwise it falls back to a default vendor priority list per vehicle category [70] [71]. The result is a mapping of vendor IDs to the IST trip (or trips) they should be notified for [72]. - **Raise Indent:** For each vendor, a `VendorTrip` record is created linking the vendor to the IST (using `ist_id` instead of `trip_id`) with status "created" [4]. The IST's `extra_details` field is updated to record which vendors were "assigned" (invited) and the timestamp [73] [74]. The system then triggers a `VendorVehicleRequisitionISTEvent`, which handles notifying vendors [75] [76]. - **Notification:** Vendors receive an SMS similar to FLM, with a link to the portal. The content will include IST details (often an IST reference number and route info). For multiple ISTs combined (if ever applicable), the logic can handle multiple reference numbers in one message [77] [78]. In practice, usually one IST corresponds to one indent. The portal URL again has the organisation placeholder replaced and is included in the SMS [6].

The system also schedules two background jobs for the indent: - `ASYNC_AUTO_ALLOCATE_PRIORITY_VENDOR_IST` – for auto re-allocation if needed. - `ASYNC_INDENT_EXPIRATION_ALERT_IST` – to send the reminder SMS before expiry [79] [80].

(Note: The configuration keys for indent timings are shared – e.g., `indent_expiry_time` and `indent_acceptance_timeout` – and are fetched from the organisation's VENDOR_CONFIG settings in both cases [81] [82]. Thus, FLM and IST indent timeouts are usually aligned unless configured otherwise.)

2. Vendor Portal – IST Requests and Acceptance

On the Vendor Portal, IST requests appear in a separate section (often labeled as "MM" or similar). The portal provides analogous endpoints for IST: - `POST /ist/vehicle-requests/get` – list active IST indents for the vendor [83]. This uses the vendor's active `VendorTrip.istIdList` to fetch relevant IST details, likely including origin hub, destination, etc. - `POST /ist/indent/event/accept` –

vendor accepts the IST indent [18] [84] . - `POST /ist/assign/vehicle` – vendor submits vehicle and driver for the IST [85] [86] . - `POST /ist/vehicle-requests/reject` – vendor rejects the IST request [87] [88] .

The frontend workflow for a vendor is analogous: they can view the IST request, and if indent acceptance is enabled, optionally hit **Accept** before providing full details. The same validations for input fields apply (vehicle number, driver info, etc.), since the implementation reuses the handler with IST-specific parameters. For example, the assign endpoint for IST requires `istId` and all the same driver/vehicle fields, and checks that `istId` is present [89] [90] .

One difference is that IST indents include route hub information. The vendor will see the origin hub for the load. In fact, the backend fetches the origin hub data to include in SMS and in some portal responses (so that the vendor knows where to report) [91] [92] .

3. Acceptance and Rejection for IST

The indent acceptance logic for IST mirrors FLM with minor differences in data handling: - If a vendor **accepts** an IST indent (`acceptISTIndent`), the system validates the IST and vendor association, then updates the `VendorTrip` status to **accepted** [93] . It logs an `ISTIndentAcceptedEvent` in the IST's event log [94] . The same config check ensures indent acceptance is enabled [95] (else an error is thrown). A new auto-allocation job is scheduled with `skipIndentAcceptance: true` for the remaining time until full expiry [96] [97] – meaning the vendor now has until `indent_expiry_time` to assign a vehicle. - If the vendor **rejects** (`rejectISTVehicleRequest`), the system immediately prepares to notify the next vendor. It removes this IST from the pending job's data and calls `autoAllocatePriorityVendorIST` within the transaction [98] [99] . The current vendor's indent is marked rejected, and an `ISTIndentRejectedEvent` is recorded [100] . The behaviour is essentially the same as FLM: rejection triggers instant re-allocation. - If the vendor times out (no response), the `ASYNC_AUTO_ALLOCATE_PRIORITY_VENDOR_IST` job will execute after the acceptance timeout or expiry and handle moving to the next vendor (deactivating the current one). Before final expiry, an **expiration alert SMS** is sent for IST just like for FLM [59] [60] – warning the vendor of the deadline.

Differences: The IST model has its own status enumeration and allowed states for vendor actions. The `validateIST` method checks that the IST's status is still "CREATED" (or pending) before allowing accept/reject [101] . If the IST has progressed or the vendor wasn't actually invited (no active `VendorTrip`), it will throw "Invalid IST Id" or "Vehicle Request expired" [102] [103] .

Also, IST indent events include the vendor as a "transporter" in their extra details for tracking [104] . The events (ISTIndentAccepted/Rejected) are analogous to trip events but stored in the `ISTEvent` log.

4. Vehicle Assignment for IST and Validation

When a vendor assigns a vehicle to an IST (`assignVehicleToIST`): - The system again checks the IST is valid and pending assignment for that vendor using `validateIST` [105] [106] . - **Vehicle & Driver Checks:** It looks up the vehicle similar to FLM, updating the vendor association if needed [107] . A critical validation for IST is that the vehicle is **not already assigned to another IST** – the code verifies `requiredVehicle.current_ist_id` is null before proceeding [108] . This ensures one vehicle isn't double-booked on two IST shipments at the same time. If that check fails, it throws an error "Vehicle is already assigned to an IST" [108] . - The driver (worker) is handled similarly: if the worker exists and is tied to a different hub, the system updates the worker's hub relation to this IST's origin hub (since in IST context, the origin hub is where the trip starts) [109] [110] . This is analogous to how FLM updates the

worker's hub to the trip's hub, but here specifically for the IST's origin. - **Assigning to IST:** The IST is then updated – the IST handler's `assignVehicle` method is called to attach the vehicle and driver details to the IST record (this likely moves the IST status forward in its workflow) [111]. Simultaneously, the vehicle's status is updated via `vehicleHandler.assignIST`, linking that vehicle to this IST as its current assignment [112]. The `VendorTrip` status for that vendor/IST is set to **"assigned"** [113]. - As with FLM, if indent acceptance was part of the flow, an `ISTIndentAssignedEvent` would be logged (not explicitly shown above, but the pattern follows). The code triggers a confirmation SMS to the vendor after successful assignment, confirming the IST has been assigned to them [114]. This SMS uses the IST details and is analogous to the FLM accept SMS.

After assignment, the IST's `vehicle_id` and `driver` info are set, and the indent cycle concludes. The IST will move to execution with that vendor's vehicle.

5. Differences Summary – FLM vs. IST Indents

While FLM (endpoint) and MM (IST) indent workflows share the same core logic (vendor invitations, accept/reject, timeout, etc.), here are the key differences and rules distinguishing them: - **Data Model:** FLM indents operate on the `Trip` object, whereas IST indents use the `InterStockTransfer`. Thus, the linking model (`VendorTrip`) uses a `trip_id` for FLM vs. `ist_id` for IST [115]. Events are logged in `TripEvent` vs `ISTEvent` tables respectively (different event classes like `IndentAcceptedEvent` vs `ISTIndentAcceptedEvent`). - **Trip Status Constraints:** An FLM trip can involve multi-stop logic (primary vs secondary leg), but from the indent's perspective it's available if the trip's status is in a pending state (one of the allowed pending statuses) [116]. An IST is typically either "CREATED" or not; the validateIST specifically checks for status `CREATED` [117] as allowable for indent actions. Once an IST is assigned (status might change to e.g. "ASSIGNED" internally), no further vendor can accept it. - **Route and Hub Info:** IST indents are tied to specific origin/destination hubs. The vendor portal and SMS notifications emphasize the hub locations. FLM trips might have an origin hub (e.g. a warehouse) or just a city; the portal code for FLM fetches the trip's route info and origin for display as well [118] [119], but IST has a clearer concept of an origin hub for pickup. - **Vehicle Uniqueness:** In IST, a vehicle cannot be assigned to two ISTs at once (checked via `current_ist_id`) [108]. In FLM, a vehicle also shouldn't be double-booked, but the check is indirect (via Medium/worker current trip) since a Trip is the primary record – a vehicle's `vehicle.current_trip_id` is handled through the medium/assignment process. Essentially both ensure one active indent assignment per vehicle. - **Indent Acceptance Timing:** Both use the same config values for timeouts. However, the code sets the auto-allocation job type differently: `ASYNC_AUTO_ALLOCATE_PRIORITY_VENDOR` for FLM [120] vs. `..._IST` for IST [121], and similarly for expiration alert. This separation means the jobs operate on the respective model contexts. The expiration alert calculation is identical, computing the alert time (~halfway) before the full `indent_expiry_time` [122] [123]. - **Portal Separation:** The Vendor Portal provides distinct API endpoints and possibly UI sections for IST requests (often labeled "MM Requests"). This keeps the two workflows separate for the vendor's convenience. For example, `fetchVehicleRequestsIST` is a different endpoint returning IST-specific fields [83]. In practice, a vendor user would toggle between regular trip requests and IST requests in the portal.

Conclusion

Both FLM and MM(IST) indent flows ensure that the right vendor is selected and given a fair chance to accept and fulfill a trip, with automated fallback to alternate vendors if they decline or don't respond. The system enforces validations at each step – from verifying input data (like driver license formats and vehicle IDs) to ensuring a vendor cannot hijack an expired request or assign a vehicle already in use. It also logs key events (Indent Accepted/Rejected/Cancelled/Assigned) for auditing the workflow.

By separating the workflows for FLM and IST while reusing core components (such as the Vendor Portal handler and common timing configurations), ProjectX provides a robust and parallel indent management system for different trip types. The above covers the complete process flow for indents in both modules, highlighting how vendors interact via the portal and how the backend rules guarantee a smooth (and automated) allocation of trips to transporters.

Sources:

- ProjectX Repository – *VendorTrip model and status definitions* 124 3
- ProjectX Repository – *Indent auto-allocation logic (FLM)* 125 126
- ProjectX Repository – *Indent auto-allocation logic (IST)* 127 128
- ProjectX Repository – *Vendor Portal APIs (assign, accept, reject)* 21 18
- ProjectX Repository – *Vendor assignment handler (FLM)* 38 44
- ProjectX Repository – *Vendor assignment handler (IST)* 108 129
- ProjectX Repository – *Reject and cancel flows* 50 52 46
- ProjectX Repository – *Expiration alert SMS* 122 58 123

---

1 2 25 56 57 auto-allocate-process-consumer.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/tms-consignment/auto-allocation/auto-allocate-process-consumer.js

3 4 5 10 53 115 124 vendor-trip.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/models/vendor-trip.js

6 7 31 54 77 78 79 80 81 121 127 128 vendor-vehicle-requisition-ist-event.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/vendor-vehicle-requisition-handler/vendor-vehicle-requisition-event/vendor-vehicle-requisition-ist-event.js

8 9 55 120 125 126 vendor-vehicle-requisition-event.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/vendor-vehicle-requisition-handler/vendor-vehicle-requisition-event/vendor-vehicle-requisition-event.js

11 69 70 71 72 73 74 75 76 91 92 tms-indent-handler.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/tms-handler/tms-handler-class/tms-handler-parts/tms-indent-handler.js

12 13 14 15 16 83 118 119 view-vehicle-requests-api.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/models/vendor-portal-parts/vehicle-requests-module/view-vehicle-requests-api.js

17 19 20 23 24 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 65 66 67 68 116 vendor-vehicle-requisition-handler-class.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/vendor-vehicle-requisition-handler/vendor-vehicle-requisition-handler-class/vendor-vehicle-requisition-handler-class.js

18 21 22 84 85 86 87 88 89 90 assign-vehicle-and-worker-api.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/models/vendor-portal-parts/vehicle-requests-module/assign-vehicle-and-worker-api.js

26 27 28 29 30 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 117 129 vendor-vehicle-requisiton-handler-ist-parts.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/vendor-vehicle-requisition-handler/vendor-vehicle-requisition-handler-class/vendor-vehicle-requisiton-handler-ist-parts.js

32 58 122 vendor_indent_expire_alert.js

https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/trip-manager/vendor_indent_expire_alert.js

59 60 82 123 ist-vendor-indent-expire-alert.js

https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/tms-handler/auto-allocation-ist/ist-vendor-indent-expire-alert.js

61 62 63 64 cancel-indent-api.js

https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/models/client-integration-parts/vendor-parts/cancel-indent-api.js