ChatGPT

Middle Mile Trip Status Transitions and Logic

Trip and Vehicle Status Changes (Gate In → Loading → Departed → Unloading → Closed)

Gate In: When a vehicle carrying a trip *gates in* to a hub, the system records the vehicle's entry and (if the vehicle is tied to an active inter-stock transfer or IST trip) updates the trip's location. Specifically, the trip's `current_hub_id` is set to the hub where the gate-in occurred [1]. No explicit change to the trip's *status* is made on gate-in (the trip may still be in **Created** status at origin, or in **In-Transit** status if this is an intermediate stop). The system validates that the gate belongs to the same hub and (for **PTL** "Partial Truck Load" trips) ensures the vehicle can only gate in at the origin if configured [2]. On the vehicle side, gating in sets the vehicle's status to **GateIn** and its `current_hub_id` to the hub (with the previous hub recorded) [3]. This effectively marks the vehicle as present at that hub. The time of gate-in is captured as the event's timestamp (`eventTime`) and recorded in the trip/vehicle event log (often stored in `last_event_time` or event extra details) [4] [5].

Start Loading: When users click **"Start Loading"** at the origin hub, the system transitions the trip to **Loading** status. The backend `markISTStartLoading` handler sets the trip's `status` to `"loading"` and timestamps the change [6]. Validations ensure the trip is at the expected hub (the trip's `current_hub_id` must match the loading hub) and that loading is allowed (e.g. cannot load at the destination hub) [7] [8]. If certain org settings are enabled, additional rules apply – for example, loading may be disallowed if the trip hasn't been "received" at that hub (for intermediate hubs) or if the trip was already departed or previously loaded [9] [10]. Once **Loading** status is set, the UI moves the trip into the **"Loading" tab** for that hub. Internally, a *Loading* event record is created and the trip's `last_status_change_time` is updated [6]. The vehicle linked to the trip is also marked as **Dock In/ Loading**: `VehicleHandler.markVehicleStartLoading` sets the vehicle's status to a docked/ loading state at that hub [11]. (The vehicle's `dock_id` is recorded when it's assigned a dock; during loading the `current_dock_id` on the vehicle is set and used in KPI calculations [12].)

During Loading: as shipments are scanned, they are added to the trip's **loading draft** (`extra_details.loading_draft`). The system may auto-save this draft on each scan [13] [14]. There are validations to prevent improper loading – e.g. scanning a bag already assigned to another trip will error out ("Bag Already Assigned to Trip…") [15] [16], or scanning items not meant for this route if not allowed (destination hub mismatch) [17].

Complete/Freeze Loading: Once all intended items are loaded, the user or system "freezes" the loading. The `ISTHandler.freeze` operation finalizes the load list: it assigns all draft waybills to the trip and updates trip status to **Loaded** [18]. In this step, the trip's status changes from `loading` to `loaded` (and `last_status_change_time` is set) [19], and the `loading_draft` is cleared out (moved into permanent relations) [20]. The total loaded weight may be calculated and stored (e.g. `transfer_weight`) [21]. Validation ensures no partial consignments remain unless shortage loading is allowed [22] [23]. After a trip is **Loaded**, it remains on the **Loading tab** (since it's still at origin hub) until departure. The system essentially treats **Loaded** as "loading done, waiting to depart."

Depart: When the trip is dispatched from the origin hub (**Depart action**), multiple things happen atomically. The IST trip's status is updated to **Departed** [24]. This is handled in `ISTHandler.gateOut`, which sets `status: 'departed'` and records the departure time (`last_status_change_time = eventTime`) [24]. All pending drafts are cleared – for instance, `extra_details.loading_draft` and any `unloading_draft` are reset to empty arrays [25] [26], ensuring no further loading/unloading can be recorded at the origin after departure. A validation prevents departure if loading isn't finished: if there are still items in the loading draft, *"Depart not allowed before IST Loading."* [27]. Similarly, if an unloading was in progress (for a multi-leg trip), departure is blocked until that's completed [28]. The Depart action also creates a **GateOut** event for the trip (an IST Depart event) with any provided comments. On the **vehicle side**, departing triggers `VehicleHandler.markVehicleGateOut`: the vehicle's status goes to **GateOut** (in transit), its `previous_hub_id` is set to the origin hub, and crucially its `current_hub_id` is now set to the *next hub* (the trip's destination or next stop) [29]. This means the vehicle is immediately associated with the receiving hub upon departure. An expected arrival time is calculated based on route transit time and stored in the vehicle's record (`extra_details.expected_time_of_arrival`) [30] [31]. After departure, the trip will move to a **"Departed" tab** on the origin hub's UI, indicating it left. The origin hub user cannot modify it further except view it in Departed/ongoing trips. At this point, the destination hub may start to see the incoming vehicle (since the vehicle's `current_hub_id` now equals the destination), but the trip itself isn't marked at the destination yet (see visibility below).

Arrival (Receive at Destination): Once the vehicle reaches the destination hub, the user performs a **Receive** or gate-in action there. The backend `ISTHandler.receive` is invoked to mark that the trip arrived. This updates the trip's status to **Received** (essentially "Arrived") [32]. The trip's `current_hub_id` is set to the destination hub at this point [32] (if not already) and `last_status_change_time` is stamped. The system captures details like the seal number and end KM reading for the trip: it validates the seal (ensuring it matches what was recorded at origin, if applicable) [33] [34] and verifies the odometer reading progression [35] [36]. It also prepares for unloading by computing which waybills are to be offloaded at this hub. All shipments on the trip with `destination_hub_id` matching this hub are fetched into `extra_details.way_bills_to_unload` [37] [38]. At this stage, the trip will appear in the **"Arrived" tab** (or equivalent) of the destination hub's interface – it's at the hub but unloading hasn't started. The arrival event is logged as an **ISTReceivedEvent** with any comments, seal, and KM details [39] [32]. (Under the hood, if configured, the system can also mark consignments as "Arrived in Vehicle" for tracking – logging that each package reached the hub in the vehicle [40] [41].)

Start Unloading: When the hub begins offloading the trip, the user clicks **"Start Unloading."** This triggers `ISTHandler.startUnloading`, which transitions the trip's status to **Unloading** [42]. The system validates that the trip is indeed at that hub (`current_hub_id` must equal the hub) and that unloading is allowed (it may require that a Receive event was done first, depending on config) [43] [44]. Starting Unloading creates an **ISTUnloadingEvent** and updates `last_status_change_time` to the unload start time [42]. The trip now moves into the hub's **"Unloading" tab**. As bags and consignments are scanned off the vehicle, they populate an `unloading_draft` (similar to loading draft) in the trip's extra details [45]. The system ensures that unloading at this stage is valid – e.g. it won't allow unloading before an arrival ("Unloading not allowed before IST Receive" if configured) [46], or after a departure (not applicable at final dest), etc. The counts of total waybills to unload are recorded for the event (for logging KPI: number of bags, etc. to unload) [47] [48].

Finish Unloading: Once all listed items are removed, the unloading is finalized. The `ISTHandler.finishUnloading` operation marks the trip as **Unloaded** (signifying all consignments destined for this hub have been offloaded) [49]. It clears the `unloading_draft` and moves any

remaining not-unloaded waybills into `extra_details.way_bills_to_unload` (these would represent shortages or exceptions) [50]. Trip `status` is set to `"unloaded"` and `last_status_change_time` updated [51]. The system may automatically apply exceptions for any packages that were not offloaded (e.g. mark them as shortages) [52] [53]. All the consignments/bags scanned in unloading are formally disassociated from the trip: for each item in the draft, `unloadBagFromIST` or `unloadConsignmentFromIST` is called to update their relationship status to unloaded [54] [55] (and possibly mark them as inscanned into the hub's stock records [56]). After **Unloaded** status, the trip typically remains on the **Unloading tab** (since it's still at the hub, just waiting closure). At this point, the vehicle can potentially be released or moved – the vehicle's status would have been set to GateIn at arrival (so it's in yard at destination). If the vehicle is leaving empty or for another trip, separate vehicle actions (gate out without an IST, etc.) would be taken, but that's outside the trip's lifecycle.

Close Trip (Complete): Finally, the trip is **closed** by the user (often labeled "Close Trip" or "Complete Trip"). This triggers `ISTHandler.complete`, which marks the IST trip as **Completed** [57]. The system validates that the trip is ready to close: the trip must be at its destination (`current_hub_id == destination_hub_id`) unless it's a special PTL case [58] [59], and it checks that no shipments remain loaded (it looks for any IST relations still in status 'loaded'; if found, it ensures an Unloaded event exists) – essentially "IST must be unloaded before completion" [60]. It may also require an end-of-trip odometer reading (`endKMReading`) and validate it (must be >= last reading) [61] [62]. On completion, trip `status` is set to `"completed"` and a final ISTCompletionEvent is recorded [63] [64]. Extra details like the `end_km_reading` and derived distances are saved for record [65]. Once a trip is **Completed**, it is removed from active lists (the trip would no longer appear on Loading/Unloading/Departed/Arrived tabs). The origin hub user can see that the trip eventually completed (if they have access to a completed trips log), and the destination can consider the inbound trip closed. For PTL trips (which might represent a return or partial-load scenario), note that the system only allows closing at origin hub if configured – *"PTL Trips can only be closed at Origin Hub."* [59].

Trip and Vehicle Visibility Across Hubs

Visibility of a trip or vehicle at a given hub is primarily governed by the `current_hub_id` field (and to some extent by origin/destination for in-transit trips). In general, a **vehicle** is considered "at" a hub if its `current_hub_id` equals that hub's ID. A **trip** (IST) is visible at the hub that currently holds or last held the trip. Key rules are:

- **While at a Hub:** When a trip is created or arrives at a hub, its `current_hub_id` is set to that hub. Thus, it will appear in that hub's dashboard. For example, gating in a trip at a hub sets `InterStockTransfer.current_hub_id` to that hub [1], making it visible there. The trip remains associated with that hub through Loading and until it departs.

- **During Transit (Departed):** After a trip departs a hub, it **still remains associated with the hub it departed from until it officially arrives at the next hub.** Notably, when an IST trip's status goes to **Departed**, the code does **not** immediately change the trip's `current_hub_id` to the destination [66]. It leaves it as the last known hub. This means the trip is initially still visible under the origin hub's "Departed" list after departure, and the destination hub does not yet see it in their Arrived list. (In contrast, the **vehicle** does switch over its hub association on depart: `VehicleHandler.markVehicleGateOut` sets the vehicle's `current_hub_id` to the next hub upon departure [29]. The vehicle is essentially "handed off" to the destination for tracking en route, with an expected arrival time [5] [29]. This allows the destination hub's vehicle view to know an incoming vehicle is on the way, even if the trip itself isn't marked arrived.)

- **Arrival at Destination:** At the moment the trip is received at the destination hub (i.e. **Arrived/ Receive action**), the trip's `current_hub_id` is updated to the destination [32]. At this point, the trip becomes visible on the destination hub's dashboard (in Arrived/Unloading tabs). Correspondingly, it will no longer be listed under the origin hub's active trips because it's no longer "current" at the origin. Essentially, **control of the trip shifts hubs upon arrival**. The origin hub can still see that trip in historical departed lists or via its static `origin_hub_id` reference, but not in active unloading since `current_hub_id` is now the destination.

- **Intermediate Hubs:** If a route has multiple legs (hubs en route), the same mechanism applies for each leg. The trip's `current_hub_id` always reflects the hub where the trip presently is. When departing an intermediate hub, `current_hub_id` stays as that intermediate hub until the next arrival. Upon arriving at the next hub, `current_hub_id` updates to that new hub. Thus, at any given time a trip is typically visible at only **one hub's active view** – the hub noted in `current_hub_id`. There isn't an overlap where two hubs see the trip simultaneously in their active lists, except possibly during the moment of status change. For example, during unloading at the destination, only the destination hub has `current_hub_id` = that hub, so only they see it in Unloading. The origin hub no longer sees it (its `current_hub_id` is now different). The phrase "visible at both hubs during unloading" would not apply under normal circumstances – once the trip's `current_hub_id` switches to the destination on arrival, the origin's UI moves it out of active view.

- **Origin vs Destination Hub Fields:** The fields `origin_hub_id` and `destination_hub_id` are static for the trip (set when the IST is created). These determine where the trip started and where it's supposed to end. They do not dynamically change. Visibility logic can use them for filtering in some cases (for instance, an origin hub user might view all trips that originated at their hub, even after departure, in a history or departed section). However, active operational screens primarily rely on `current_hub_id`. For example, a trip in transit might still appear under the origin's "Departed" tab **because** `current_hub_id` hasn't yet changed from the origin hub until arrival. Once arrived, the destination hub gets it in their tabs (and the origin's departed list would drop it if filtering by current hub). Vehicles use a similar pattern: a vehicle's `previous_hub_id` and `current_hub_id` get updated on each leg. When a vehicle leaves Hub A for Hub B, `previous_hub_id` is set to A and `current_hub_id` to B [29]. So Hub A will no longer list that vehicle as currently available, while Hub B can now see it incoming (often with a status indicating en route). When the vehicle gates in at B, it remains `current_hub_id = B` and status gatein, showing as arrived at B [3]. In summary, **a vehicle or trip "belongs" to exactly one hub at a time in the UI's eyes**, determined by `current_hub_id` (with the one exception that an origin hub might still consider a departed trip until it's officially received elsewhere).

- **PTL Trip Visibility:** Partial Truck Load trips (where the truck returns to origin) are an edge case. A PTL trip might depart the origin and eventually come back to the origin instead of having a different destination. The system's config can enforce that such trips are only gated in or closed at the origin [2] [59]. In such cases, the origin remains the focal hub throughout. Generally, for standard trips, once a trip is completed at the destination, it will not show on either hub's active dashboards.

Tab Filtering Logic (Loading, Departed, Arrived, Unloading, etc.)

Both the backend and frontend segregate trips into status-based tabs to reflect their stage in the middle-mile process. The filtering works roughly as follows, using trip fields like `status`, hub associations, and timestamps:

- **"Loading" Tab (Origin Hub):** Shows trips that are in the process of being loaded at the hub. These are trips with `current_hub_id` = *[current hub]* and status **Loading** or **Loaded**. In practice, as soon as "Start Loading" is initiated, the trip status becomes `loading` and it appears in this tab [6]. It remains under Loading until departure. Even if the loading is finished (status internally moves to `loaded` after freeze), the trip still hasn't left the hub, so it stays in the Loading section up to the point of departure. (There may not be a separate "Loaded" tab; a loaded-but-not-departed trip is typically still grouped with loading trips.) Any trip in **Created** status that has a vehicle gated in could be treated similarly – in many cases the UI will prompt to start loading immediately, but if a trip were sitting in Created status at a hub, it would likely be considered in the Loading queue as well. The key condition is that the trip's `current_hub_id` equals the hub and it has not departed yet. For example, a trip at origin with status *Created/Loading/Loaded* qualifies for the Loading tab (it's at origin, pending departure).

- **"Departed" Tab (Origin or En Route):** This tab lists trips that have left the hub but not yet arrived at their destination. For an **origin hub user**, once they click Depart, the trip (status **Departed**) moves from Loading to the Departed tab. The backend set status to `'departed'` [24] and cleared loading info, indicating it's en route. The UI will include trips in Departed tab typically if `status = departed` **and** the trip hasn't been completed. There are a couple ways the filter might be implemented:

- For origin hubs, it may simply show trips with `status = departed` and `origin_hub_id` = current hub (meaning trips that started here and are currently in transit). Because `current_hub_id` is still the origin until arrival, an origin could also filter by `current_hub_id = hub AND status = departed` – which amounts to the same set. These trips disappear from the Departed tab once they arrive at the next hub (at that moment the trip's `current_hub_id` changes and status changes to received/unloading, so they no longer meet the filter).

- For a **destination hub user**, there might also be an interest in trips that are on the way to that hub. Since prior to arrival the trip's `current_hub_id` is not yet the destination, the front-end might explicitly query for trips where `destination_hub_id = myHub AND status = departed` to show incoming in-transit trips. However, in many implementations, the inbound "Arrived" tab only shows things once they're marked arrived. In our system, because the vehicle object was transferred to the dest, the dest operations team would know an incoming trip is expected from the vehicle list, even if the trip itself isn't yet in an Arrived tab. Thus, the **Departed tab is mainly used by the source/origin hub** to track trips that have left. (If the UI does list inbound departed trips for the dest, it would use `destination_hub_id` and status to filter them.)

- **"Arrived" Tab (Destination Hub):** This shows trips that have reached the hub but unloading has not begun. In the backend, when a trip is received at the hub, its status becomes **Received** (essentially an "arrived" state) [32]. Such trips will have `current_hub_id = hub` and `status = received`. The Arrived tab filter would include trips with status `received` at that hub. These are waiting for unloading to commence. The presence of data in

`extra_details.way_bills_to_unload` after the receive action indicates items to be offloaded [37], but the status itself is what UI uses. As soon as the user hits "Start Unloading", the trip leaves the Arrived tab.

- **"Unloading" Tab (Destination Hub):** Trips that are actively being unloaded or have finished unloading but not closed appear here. These have `current_hub_id = hub` and status **Unloading** or **Unloaded**. When "Start Unloading" is clicked, status goes to `unloading` [42], moving the trip from Arrived to Unloading tab. It remains in this tab throughout the offload process. If unloading is completed (status `unloaded` is set via finishUnloading [51]), the trip can either stay in the Unloading tab (often as a "Pending Closure" entry) until the final close is done. The system might not have a distinct "Unloaded" tab; instead, an unloaded-but-not-closed trip is typically still shown under Unloading (since the trip is technically still at the hub awaiting closure). Only after the trip is marked **Completed** will it drop off.

- **Other Tabs/Panels:** Once a trip is **Completed**, it usually goes to a historical list (not part of active operations tabs). Some UIs have a "Closed" or "Completed" filter to review past trips, but operationally it's no longer on the live dashboard. There may also be an error/exception tab if applicable (e.g., trips with exceptions), but the question focuses on the main lifecycle. The **Vehicles page** likely has its own tabs (like *Available, In Transit, At Dock* etc.). For example, vehicles at a hub with status gatein/dockassigned might show as "At Hub", whereas vehicles with status gateout (en route) might show under an "En Route" or "In Transit" category. Since the question specifically mentions the Vehicles Page, note that:

- A vehicle that is **Gate-In** (inside the hub) or docked appears in the hub's active vehicle list. When it goes **Gate-Out**, the vehicle moves off the current hub's list and appears under the next hub's inbound list. For instance, after marking Gate Out, the vehicle's status = `gateout` and its `current_hub_id` is already set to the next hub [29], so it will show up on that hub's vehicle page (often under an "En Route" or incoming section). Once it **gates in** at the destination, its status goes back to `gatein` and it's firmly listed in that hub's "At Hub" vehicles.
- The vehicle page might be filtered by statuses like `gatein` (on premises), `gateout` (left hub, possibly filterable by previous_hub or next_hub). For example, at any hub, vehicles with `current_hub_id = hub` and status *gatein/dockassign/dockin* would be in an "On Premise" or "Idle/Available" list, whereas vehicles with `current_hub_id = hub` and status *gateout* might be in an "Inbound (on trip)" list – indicating they are currently on a trip coming to that hub.

In summary, the **backend handlers set the fields that drive tab logic**, and the front-end uses those fields to filter: - *Loading tab:* trips at this hub with status loading(loaded). - *Departed tab:* trips that left this hub (status departed, and either origin_hub or current_hub still this hub) and not yet arrived elsewhere. - *Arrived tab:* trips at this hub with status received (arrived, not unloading yet). - *Unloading tab:* trips at this hub with status unloading or unloaded (in offload process). - *Completed:* trips that have status completed are usually excluded from the main tabs (they might be available via a separate filter or report if needed).

Throughout these transitions, the system enforces logical **validations** at each step to prevent out-of-sequence actions: - You **cannot Depart** a trip from a hub before starting and freezing loading (depart validation fails if loading isn't done or if any unload at that hub is pending) [27]. - You **cannot Start Unloading** at a hub before the trip is marked arrived/received there (validation will demand a prior "received" event if configured) [46]. - You **cannot Gate In** a vehicle twice in a row without a Gate Out in between (vehicle gate-in validation ensures if it already has a current hub and isn't in gateout status, a new gate in is not allowed) [67]. - A **Gate Out** at the destination hub is disallowed (since that's the end of

the line) [68]. - PTL trips have hub-specific restrictions (e.g., *"PTL Trips are not allowed to gate in at another hub."* [2] and must be closed at origin [59]).

By following the field updates (`status`, `current_hub_id`, `previous_hub_id`, `extra_details` flags, etc.) and the above conditions, the system ensures that each trip and vehicle appears in the correct hub's view at the correct time, and moves through the UI tabs corresponding to its lifecycle stage.

Sources:

- ProjectX Backend – TMS Handler and IST Handler logic for status updates [6] [18] [24] [32], and validations [27] [69].
- ProjectX Backend – Vehicle Handler logic for hub transfers (gate in/out) [29] [3].
- ProjectX Backend – PTL special-case checks [2] [59].

---

[1] [2] [4] [24] [25] [26] [66] [68] ist-handler.js
https://github.com/shipsy/projectx/blob/c1cee2f6a187baed0fde81a1325e82e94ada9335/common/domain-models/ist-handler/ist-handler.js

[3] [5] [29] [30] [31] [67] vehicle-handler-class.js
https://github.com/shipsy/projectx/blob/c1cee2f6a187baed0fde81a1325e82e94ada9335/common/domain-models/vehicle-handler/vehicle-handler-class/vehicle-handler-class.js

[6] [7] [8] [9] [10] [13] [14] [18] [19] [20] [21] [22] [23] [27] [28] [69] ist-outbound.js
https://github.com/shipsy/projectx/blob/c1cee2f6a187baed0fde81a1325e82e94ada9335/common/domain-models/ist-handler/ist-handler-parts/ist-outbound.js

[11] [12] [15] [16] [17] tms-handler-class.js
https://github.com/shipsy/projectx/blob/c1cee2f6a187baed0fde81a1325e82e94ada9335/common/domain-models/tms-handler/tms-handler-class/tms-handler-class.js

[32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52] [53] [54] [55] [56] [57] [58] [59] [60] [61] [62] [63] [64] [65] ist-inbound.js
https://github.com/shipsy/projectx/blob/c1cee2f6a187baed0fde81a1325e82e94ada9335/common/domain-models/ist-handler/ist-handler-parts/ist-inbound.js