System logistyki wewnętrznej oparty o RDS

Projektant: Wojciech Młynarczyk
Wersja: 0.1 (draft)

---

1. Streszczenie (Executive Summary)

1.1 Cel projektu

Celem projektu jest zaprojektowanie i wdrożenie systemu logistyki wewnętrznej, który współpracuje z fleet managerem autonomicznych wózków widłowych – RDS (Robot Dispatch System) firmy SEER.

1.2 Czym jest RDS

RDS to oprogramowanie firmy SEER do zarządzania flotą autonomicznych wózków AGV/AMR. System ten zarządza trasami, zadaniami i statusem robotów, ale jego standardowa funkcjonalność nie obejmuje niektórych elementów logistyki wewnętrznej wymaganej w naszym projekcie.

1.3 Problem, który rozwiązujemy

RDS świetnie zarządza robotami i prostymi miejscami, ale „z pudełka" **nie** obsługuje: - buforów o pojemności zależnej od typu towaru, - dynamicznego wyboru dokładnego miejsca odkładania/pobierania w takim buforze, - przejrzystego pokazywania **na jakiej podstawie** została wybrana trasa.

Dlatego dodajemy warstwę logiczną nad RDS, która zapewni prosty w modyfikacji i czytelny proces decyzyjny (z uzasadnieniem widocznym w UI), przy zachowaniu całego stanu w RDS. Ta warstwa logiki nad RDS będzie: - trzymała cały stan **w RDS** (żadnych prywatnych baz obok), - prosta do testowania i rozwoju (podział na małe, niezależne moduły), - wyjaśniająca decyzje (UI zawsze pokaże „dokąd" i „dlaczego").

1.4 Kluczowe pojęcia z RDS

- **Pole procesowe (worksite)** – pojedynczy logiczny punkt procesu w zakładzie (np. pole produkcji, pole owijarki). Każde worksite ma atrybuty wykorzystywane do przechowywania stanu, które są edytowane przez API.
- **Zadanie (task)** – jednostka pracy realizowana przez robota (np. przewieźć paletę z A do B). W RDS rozróżniamy:
- **szablon** zadania (np. przewóz towaru z pola do pola),
- **instancję** zadania – konkretny przejazd, np. z konkretnego źródła do konkretnego celu z określonymi parametrami ustawionymi w momencie tworzenia (typ towaru, profile podnoszenia itp.) oraz z określonym statusem ustawianym w trakcie wykonywania zadania.

1.5 Jedno źródło prawdy – wyłącznie w RDS

Cały stan systemu jest utrzymywany wyłącznie w RDS. Nie ma żadnych dodatkowych baz danych. Stan całego systemu definiują: 1) wartości atrybutów miejsc procesowych (worksite'ów) oraz
2) lista aktualnie wykonywanych tasków w RDS wraz z parametrami wywołania oraz statusami, zmienianymi w trakcie ich działania.

Wszystkie decyzje, podglądy i sygnały zapisujemy w polach worksite'ów, w parametrach tasków i ich polach statusu.

1.6 Reprezentacja „złożonych miejsc" (np. buforów o zmiennej liczbie slotów)

W RDS jedno worksite odpowiada pojedynczemu punktowi odkładczemu. Taki model utrudnia opis miejsc złożonych (np. buforów), bo wymagałby utworzenia setek worksite'ów dla każdego możliwego slotu. Zamiast tego przyjęliśmy, że złożone miejsca będziemy opisywać strukturami JSON przechowywanymi w tekstowych atrybutach worksite'u.

W systemie działają trzy logiczne moduły (UI, Orchestrator, Occupancy Manager), które muszą niezależnie zapisywać dane, dlatego potrzebujemy **trzech niezależnie zapisywanych pól**. Pojedynczy worksite w RDS udostępnia jednak tylko **dwa** takie pola tekstowe: `content` i `tags`. Dlatego **dla jednego fizycznego miejsca** stosujemy **dwa worksite'y** o ustalonych rolach: - **MIEJSCE_CONTROL** – używany do intencji operatora (`content`) i podglądu/feedbacku Orchestratora (`tags`). - **MIEJSCE_OCCUPANCY** – używany do opisu zajętości bufora (`content`).

Taki podział pozwala modelować złożone bufory bez tworzenia setek osobnych worksite'ów dla każdego możliwego punktu odkładania: cała geometria/pojemność jest trzymana wewnątrz jednego pola tekstowego (`MIEJSCE_OCCUPANCY.content`) jako struktura JSON. Pozwala też uniknąć wzajemnego nadpisywania danych przez niezależne moduły.

Używamy następujących pól: - pole **MIEJSCE_CONTROL.content** – zapisuje tam wyłącznie **UI module** (intencje operatora: accept/release, goodsType, tryb AUTO/MANUAL, manualTarget itp.). - pole **MIEJSCE_CONTROL.tags** – zapisuje tam wyłącznie **Orchestrator** (podgląd decyzji „dokąd i dlaczego", lista kandydatów z uzasadnieniem, feedback systemowy). - **MIEJSCE_OCCUPANCY.content** – zapisuje **Occupancy Manager** (zajętość bufora: siatka depth×levels, stackHeights, rezerwacje, blokady).

1.7 Najważniejsze decyzje architektoniczne

1) Implementujemy trzy moduły, luźno powiązane przez RDS: - **UI module** – zapisuje intencje operatora do pola `MIEJSCE_CONTROL.content` (np. „akceptuj przyjęcia", „wyślij dalej", tryb AUTO/MANUAL). - **Orchestrator** – czyta intencje operatora i zajętość buforów a następnie tworzy/kończy taski w RDS; zapisuje do pola `MIEJSCE_CONTROL.tags` podgląd decyzji (dokąd pojedzie i dlaczego) oraz feedback systemowy. - **Occupancy Manager** – utrzymuje zajętość buforów i ich konkretnych pól składowych w osobnych w polu `MIEJSCE_OCCUPANCY.content` osobnych worksitów.

2) Dwa worksite'y na jedno fizyczne miejsce: **MIEJSCE_CONTROL** i **MIEJSCE_OCCUPANCY** wraz z rozdzielonymi polami: - `content` – zapisuje **UI** (intencje) albo **Occupancy Manager** (zajętość) – w

oddzielnych worksite'ach, aby się nie nadpisywać. - `tags` – zapisuje **Orchestrator** (podgląd decyzji, uzasadnienia, status systemowy).

3) **Routing AUTO/MANUAL**: - **AUTO** – docelowe miejsce transportu wyznacza system na podstawie procesu (flow), polityk i zajętości; - **MANUAL** – operator wskazuje cel w granicach polityk (np. manual może być zabroniony na produkcji).
W obu trybach UI pokazuje kandydatów oraz uzasadnienie wyboru.

4) **Eventual consistency z idempotencją i kompensacją (m.in. „dispatchable fix"):**
Orchestrator naprawia stan przy zmianach warunków. Jeśli trzeba coś anulować, kończy tylko taski **bez przypisanego AGV**, aktualizuje podgląd i dba o przywrócenie robota do stanu „dispatchable" po szybkich stopach.

5) **Sygnał** `canInterruptSafely` **w** `status_description` – bezpieczne przerywanie tasków zależnie od fazy.

1.8 Jak działa system – przepływ logiczny

1) Ludzki operator w UI ustawia intencję (np. „przywieź opakowanie", tryb AUTO).
2) Moduł UI zapisuje tę intencję w `MIEJSCE_CONTROL.content`.
3) Moduł Orchestrator odczytuje intencję i zajętość buforów (z `MIEJSCE_OCCUPANCY.content`), po czym: - wybiera cel transportu, - tworzy lub kończy task w RDS, - zapisuje w `MIEJSCE_CONTROL.tags` uzasadnienie decyzji („dokąd i dlaczego"). 4) Moduł Occupancy Manager cyklicznie aktualizuje stan zajętości w złożonych buforach.
5) RDS steruje ruchem robotów zgodnie z utworzonymi zadaniami.

W efekcie operator w UI zawsze widzi:
- dokąd pojedzie ładunek,
- dlaczego taki wybór został podjęty,
- i w jakim stanie jest dany robot lub bufor.

1.9 Dlaczego potrzebujemy odporności na niespójności (eventual consistency)?

W naszym rozwiązaniu kilka modułów jednocześnie zmienia stan w RDS: UI (intencje), Orchestrator (taski i podgląd), Occupancy Manager (zajętość), a czasem także sam RDS. Nie ma globalnych transakcji obejmujących cały system, więc akceptujemy powstawanie chwilowych niespójności i korygujemy je w aplikacji.

Przykłady niespójności i zasad korygowania: - Orchestrator widzi „1 wolne miejsce" i tworzy task, a ludzki operator w tym samym momencie oznacza bufor jako pełny. Decyzję da się cofnąć: kasujemy tylko taski bez przypisanego AGV. - Wyścig z przypisaniem AGV: Orchestrator anuluje świeży task, ale RDS w tej samej chwili przypisuje do niego robota. Robot stał się `undispatchable`, system wykona automatyczny „dispatchable fix". - Zajętość bufora (OCCUPANCY) aktualizuje się z opóźnieniem.

Zasady korygowana stanu: 1) Anulujemy tylko taski **bez AGV** (`agvId == null`). 2) Jeśli trafimy na rzadki wyścig: AGV właśnie przypisany, a my anulowaliśmy, to w RDS robot może przejść stan w

`undispatchable`, który wymaga ręcznej interwencji ludzkiego operatora. **Automatycznie przywracamy** go do „dispatchable" (procedura „dispatchable fix"), aby uniknąć konieczności ręcznej interwencji. 3) Cykliczna rekonsyliacja: Orchestrator w pętli sprawdza warunki i koryguje decyzje; UI widzi zawsze aktualny podgląd z uzasadnieniem. 4) Idempotencja: tworzenie/anulowanie/rezerwacje projektujemy tak, by wielokrotne wywołania były bezpieczne i łatwo było cofnąć decyzję, gdy zmienią się warunki.

1.10 Bezpieczne przerywanie rozpoczętych tasków – sygnał z poziomu taska

Niektóre taski (np. „pobierz z bufora") można przerwać w drodze – zanim wózek fizycznie podejmie paletę. Inne – nie (bo przerwanie w złym momencie wymaga ręcznej weryfikacji stanu robota i ładunku).

Aby Orchestrator mógł rozróżnić te sytuacje w czasie rzeczywistym, wykorzystujemy pole `status_description` taska. Kod taska może ustawiać tam JSON z atrybutami kontrolnymi, m.in.: `{ "canInterruptSafely": true }`.

Zasada działania: - Jeżeli operator/Orchestrator chce przerwać task, a task ma AGV przypisany i `status_description.canInterruptSafely == true`, to Orchestrator: 1) zatrzymuje task, 2) automatycznie przywraca robota do stanu `dispatchable` (bo przerwanie było „bez konsekwencji"). - Jeżeli `canInterruptSafely != true`, Orchestrator **nie** przerywa takiego taska (albo wymaga świadomej decyzji z ostrzeżeniem), by nie wprowadzić robota w stan wymagający ręcznej interwencji.

Dzięki temu to **sam task** (znający swój cykl życia) informuje, kiedy przerwanie jest bezpieczne. Orchestrator nie musi zgadywać fazy realizacji.

1.11 Mechanizmy zapewniania poprawności działania

1) **Idempotencja** – wielokrotne wykonanie tej samej decyzji nie psuje stanu (np. można bezpiecznie ponowić tworzenie lub anulowanie taska).
2) **Rekonsyliacja** – Orchestrator okresowo sprawdza stan i koryguje błędne decyzje.
3) **Dispatchable fix** – automatyczne przywracanie robota do gotowości po błędnym lub przerwanym zadaniu.
4) **Bezpieczne przerywanie (`canInterruptSafely`)** – każdy task określa, czy może być bezpiecznie anulowany w danym momencie.

1.12 Dane i konfiguracja systemu

System wykorzystuje spójny model danych w RDS: - **CONTROL.content** – Intencje operatora (accept/release, goodsType, AUTO/MANUAL itd.) - **CONTROL.tags** – Decyzje i uzasadnienia zapisane przez Orchestratora - **OCCUPANCY.content** – Zajętość bufora w formacie JSON (siatka, rezerwacje, blokady) - **status_description (w tasku)** – Dane o stanie zadania, np. `{ "canInterruptSafely": true }`

1.13 „Bufor pełny" w trakcie dojazdu z ładunkiem – wymagania

- **BLOCK_AND_WAIT** – czekaj i sonduj wolny slot,
- **ASK_OPERATOR** – poproś operatora o decyzję (z timeoutem),
- **FORCE_DROP_ALLOWED** – awaryjne, ale bezpieczne odłożenie na bufor porzucania (jeśli dozwolone),
- **AUTO_REROUTE** – automatyczne przeadresowanie na alternatywny bufor,

- **fallback**: parking.

1.14 Wymagania dla interfejsu na wózku (HMI)

Operacje dostępne z HMI (na bieżącym zadaniu/robocie): - Wstrzymaj zadanie (pause). - Wznów zadanie (resume). - Przerwij zadanie (terminate and set `undispatchable`). - Wyjdź z ruchu (set `undispatchable`). - Wejdź do ruchu (set `dispatchable`). - Opuść widły na 0. - Podnieś widły na wysokość transportową. - Jedź na parking (bez manipulacji ładunkiem) i przejdź w `undispatchable`. - Odłóż aktualny ładunek na zdefiniowany bufor porzucania i przejdź w `dispatchable`.

Widoki/statusy na HMI: - Stan robota: `dispatchable` / `undispatchable`. - Bieżący task: ID, źródło/cel, kluczowe parametry (typ towaru, tryb AUTO/MANUAL, wybrany slot/lokalizacja). - Czy można przerwać bezpiecznie: prezentacja `canInterruptSafely`. - Uzasadnienie trasy: skrót z „explain” (dlaczego ten bufor/cel).

1.15 Co dostarczamy w tej wersji projektu

- **Model danych w RDS**: konwencje JSON dla `CONTROL.content`, `CONTROL.tags`, `OCCUPANCY.content` i konwencje `status_description` w taskach.
- **Konfigurację procesu (`config.json`)**: słownik towarów, jawne kroki flow, polityki routingu, szablony tasków.
- **4 generyczne szablony tasków** (buffer→place, place→buffer, buffer→buffer, place→place) z parametrami robota (profile wideł, rozpoznawanie palet itp.).
- **Mechanizm podglądu decyzji („dokąd i dlaczego”)** – czytelny dla operatora.
- **Podstawy audytu** – spójny format wpisów (EVENT/DECISION) i `traceId`, by można było odtworzyć tok decyzji.

1.16 Zakres dokumentu (na tym etapie)

Skupiamy się na koncepcji, modelu danych, konfiguracji, interakcjach i mapowaniu buforów (3D → punkty mapy RDS). Testy, wdrożenie, bezpieczeństwo i migracje opiszemy po akceptacji designu.

1.17 Korzyści dla klienta

- **Przejrzystość działania** – UI jasno pokazuje, co i czemu się wydarzy.
- **Modyfikowalność** – zmiany procesu realizujemy konfiguracją (nowe towary/flow/miejsca), bez ruszania kodu całego systemu.
- **Prostsze utrzymanie** – trzy małe moduły, stan wyłącznie w RDS, brak dodatkowych baz danych.
- **Odporność operacyjna** – kompensacja decyzji, bezpieczne przerwania tasków, automatyczne przywracanie robotów do „dispatchable” w rzadkich wyścigach.

---

2. Przykład referencyjny

2.1 Zakres i cel przykładu

Przykład przedstawia minimalny, lecz kompletny łańcuch procesu wewnątrzzakładowego z udziałem autonomicznych wózków (RDS SEER). Pokazujemy: - **dwa** typy towaru/opakowania: `EUR_PAL` oraz `BOX`; - **dwa** przepływy (flow), po jednym dla każdego typu towaru; - zestaw **miejsc procesu (places)** obejmujący produkcję (dwa porty linii), bufor pustych, owijarkę, spinanie, bufor wysyłki, sprzedaż oraz bufor odpadu; - **interakcję UI** operatora z miejscami (intencje: accept/release, tryb AUTO/MANUAL, wybór typu towaru, cele manualne); - użycie pary RDS-owych worksite'ów per miejsce: `..._CONTROL` (UI+Orchestrator) i opcjonalnego `..._OCCUPANCY` (Manager Occupancy). - widoczność decyzji dzięki `..._CONTROL.tags.preview/explain`.

Celem jest pokazanie, jak stan systemu (intencje operatora, decyzje Orchestratora, aktywne taski RDS) współgra w praktyce.

2.2 Typy towaru

- `EUR_PAL` — paleta EUR.
- `BOX` — skrzynia.

Wartość `goods.type` jest wybierana przez operatora w UI na danym miejscu i warunkuje domyślny flow, parametry tasków oraz polityki wyboru celu.

2.3 Enum `releaseKind` (skrót celu wysyłki)

Wartości dozwolone (konfigurowalne **per place**): - `"auto"` – cel wynikający z **flow** dla bieżącego `goods.type` (następny krok procesu). - `"waste"` – wywóz techniczny do `WASTE_BUF`. - `"wrap"` – bezpośrednio do owijarki. - `"strap"` – bezpośrednio do spinania. - opcjonalnie `"ship"` – do bufora wysyłki, `"sales"` – bezpośrednio na sprzedaż.

To, które wartości są dostępne, definiujemy w `uiPolicy.allowReleaseKinds` danego miejsca.

2.4 Miejsca procesu (places)

Każde **place** ma odpowiadające mu RDS worksite'y: - **PLACE_CONTROL** - `content`: intencje UI (accept/release, `goods.type`, `routing.mode` = AUTO/MANUAL, ewentualny `manualTarget`, `releaseKind`), - `tags`: feedback Orchestratora (`preview` – dokąd i jaką polityką; `explain` – dlaczego; `flags.safeToInterrupt`). - **PLACE_OCCUPANCY** (opcjonalny) - `content`: stan zajętości (np. `stackHeights`, wolne sloty/rezerwacje); zapisuje **wyłącznie** Manager Occupancy.

W przykładzie używamy: - `EMPTY_BUF` – bufor pustych opakowań, - `PROD1_P1` – port wejściowy linii 1 (puste IN; „waste" OUT), - `PROD1_P3` – port wyjściowy linii 1 (produkt OUT; „waste" OUT), - `WRAP` – owijarka z 2-stanowiskowym buforem, - `STRAP1` – spinanie dla linii 1, - `SHIP_BUF` – bufor wysyłki, - `SALES` – sprzedaż/rolotok, - `WASTE_BUF` – bufor odpadu.

Analogicznie można wprowadzić linię 2: `PROD2_P1`, `PROD2_P3`, `STRAP2`.

> **Uwaga:** stanowisko spinania (`STRAP1`, `STRAP2`) **modelujemy jak alejkę buforową** (identyczny model jak dla `BUFFER_LANE`): zapełnianą od jednej strony i później opróżnianą. Dzięki temu korzysta z tej samej struktury zajętości co bufory (depth/levels/stackHeights).

2.5 Przepływy (flows)

2.5.1 Flow dla `EUR_PAL`

- **Puste → produkcja**: `EMPTY_BUF → PROD1_P1` (na żądanie operatora na P1).
- **Produkt → wyjście**: `PROD1_P3 → WRAP → STRAP1 → SHIP_BUF → SALES`.
- **Wywóz techniczny**: z `PROD1_P1` lub `PROD1_P3` → `WASTE_BUF`.

2.5.2 Flow dla `BOX`

- **Puste → produkcja**: `EMPTY_BUF → PROD1_P1`.
- **Produkt → wyjście**: `PROD1_P3 → WRAP → SHIP_BUF → SALES`.
- **Wywóz techniczny**: jak wyżej → `WASTE_BUF`.

2.6 Zachowanie miejsc (z `releaseKind`)

2.6.1 `PROD1_P1` (puste IN, odpad OUT)

UI: - „Przywieź” puste: `intent.accept=true`, `release=false`. - `routing.mode`: **AUTO** (domyślnie); MANUAL dopuszczalny tylko jeśli `uiPolicy.allowManualTargetsForAccept` wskazuje konkretne bufory/alejki (np. `EMPTY_BUF`). - „Wywieź odpad”: `intent.release=true`, `releaseKind="waste"`.

Blokady: - Po utworzeniu taska (dla accept/release) blokujemy zmianę `goods.type` aż do zakończenia/zatrzymania. - „Anuluj” dozwolony, jeśli task **nie ma AGV** lub task zasygnalizował `canInterruptSafely=true` (w `status_description`).

Orchestrator → ..._CONTROL.tags: - `preview`: źródło/cel (np. z której alejki `EMPTY_BUF`), - `explain`: warunki wyboru (profil pustych, dostępny slot), - `flags.safeToInterrupt`: aktualny status anulowalności.

2.6.2 `PROD1_P3` (produkt OUT, odpad OUT)

UI: - „Wywieź (produkt)”: `intent.release=true`, `releaseKind="auto"` (co oznacza „następny krok z flow dla `goods.type`” – np. WRAP; kolejne kroki tylko informacyjnie w `explain`). - **Skróty ręczne** (jeśli pozwala konfiguracja miejsca w `allowReleaseKinds`): `releaseKind="wrap" | "strap" | "ship" | "sales" | "waste"`.

Blokady/Anuluj: jak wyżej (analogicznie do P1).

Orchestrator → ..._CONTROL.tags (przykłady): - `releaseKind="auto"`:

```
{
  "preview": { "target": "WRAP", "policy": "FLOW_AUTO" },
  "explain": ["auto → następny krok flow: WRAP", "po WRAP → STRAP1 → SHIP_BUF →
SALES"],
  "flags": { "safeToInterrupt": true }
}
```

- `releaseKind="wrap"`:

```
{
  "preview": { "target": "WRAP", "policy": "MANUAL_SHORTCUT" },
  "explain": ["skrót ręczny: bezpośrednio do owijarki"],
  "flags": { "safeToInterrupt": true }
}
```

- `releaseKind="waste"`:

```
{
  "preview": { "target": "WASTE_BUF", "policy": "WASTE" },
  "explain": ["wywóz techniczny (odpad)"],
  "flags": { "safeToInterrupt": true }
}
```

2.6.3 `WRAP` (owijarka + bufor 2-slotowy)

- **Wejście**: gdy stanowisko wolne → cel `WRAP`; inaczej → *late binding* do wolnego slotu bufora (decyzja tuż przed dojazdem).
- **Wyjście**: po potwierdzeniu operatora (`intent.release=true`) towar kierowany dalej (np. do `STRAP1` dla `EUR_PAL` lub wprost do `SHIP_BUF` dla `BOX`, jeśli tak stanowi flow/skróty).

2.6.4 `STRAP1` (spinanie linii 1)

- **Przyjęcie**: tylko gdy `STRAP1_CONTROL.content.intent.accept=true`.
- **Wyjście**: po zakończeniu spinania operator ustawia `intent.release=true` → kierunek wg flow (np. `SHIP_BUF`) lub wybrany skrót.

2.6.5 `SHIP_BUF` (bufor wysyłki)

- **Tryb przyjmowania**: domyślnie `accept=true` (można wstrzymać dla jakości).
- Możliwe **przypisanie** `goods.type` **do pustej alejki**, gdy reguła na to pozwala i alejka jest całkowicie wolna.
- UI może zaznaczyć „kilka pierwszych palet” do wypychania; po ich wyjeździe bufor automatycznie wraca do przyjmowania.

2.6.6 `SALES`

- Odbiera z `SHIP_BUF` (gdy gotowe).

2.6.7 `WASTE_BUF`

- Stały cel dla `releaseKind="waste"`.

2.7 Granice przykładu

- Przykład nie definiuje pełnych parametrów manewrowych (wysokości wideł, rozpoznawanie palet, profile odkładania/pobrania) — to lokalna konfiguracja tasków i/lub Managera Occupancy, która będzie zdefiniowana osobno per miejsce i per towar.
- Przykład pokazuje mechanizm przypisania typu do alejki w `SHIP_BUF` (gdy pusta), ale nie zawiera kompletnego algorytmu zarządzania przestrzenią — to rola Manager Occupancy.

3. Model danych

> Cel: formalny opis struktur danych wymienianych między modułami (UI, Orchestrator, Occupancy Manager) oraz kontraktów z RDS (worksites, taski).

3.1. Encje i odpowiedzialności (skrót)

- **Place** – logiczne miejsce procesu (np. produkcja, bufor, owijarka). Mapowane na parę RDS worksite:
- `<PLACE>_CONTROL` (UI ↔ Orchestrator)
- `<PLACE>_OCCUPANCY` (Occupancy Manager)
- **Worksite (RDS)** – punkt procesowy z polami tekstowymi `content` i `tags`.
- **Task (RDS)** – instancja zlecenia przeniesienia ładunku; parametry wejściowe stanowią część stanu systemu.

3.2. `<PLACE>_CONTROL.content` (JSON)

Zawartość edytowana wyłącznie przez UI. Przykładowe pola:

```
{
  "placeId": "PROD1_P1",
  "ts": "ISO-8601",
  "goods": {
    "type": "EUR_PAL|BOX|null",
    "releaseKind": "auto|waste|wrap|strap|null"
  },
  "intent": {
    "accept": true,
    "release": false,
    "mode": "auto|manual",
    "manualTarget": null
  },
```

```
  "preview": {
    "nextDecision": {
      "targetPlaceId": null,
      "reason": null
    }
  }
}
```

Uwagi: - `preview.nextDecision` – opcjonalna informacja do podglądu w UI (nie wpływa na logikę Orchestratora).

3.3. `<PLACE>_CONTROL.tags` (JSON)

Zawartość edytowana wyłącznie przez Orchestrator. Przykładowe pola operacyjne:

```
{
  "state": {
    "hasActiveTask": false,
    "activeTaskId": null,
    "dispatch": "idle|enqueued|assigned|running|blocked"
  },
  "links": {
    "boundLane": null,
    "flowName": "FLOW_EUR|FLOW_BOX|null"
  },
  "safety": {
    "lastAgv": null,
    "lastUndispatchableFix": null
  }
}
```

$1

$1

3.6 Uniwersalny model CONTROL (content & tags)

Cel: spójny, prosty model sterowany przez UI dla wszystkich rodzajów miejsc (places). **UI zapisuje tylko do** `CONTROL.content`, natomiast **Orchestrator zapisuje wyłącznie do** `CONTROL.tags` (np. podgląd planu, decyzje, błędy). Dzięki temu można zmieniać UI i Orchestrator niezależnie.

3.6.1 Struktura `CONTROL.content` (zapisywana przez UI)

```
{
  "placeId": "<PLACE_ID>",
  "ts": "<ISO8601>",

  "ui": {
    "goodsTypeSelected": "<TYPE|null>",        // wybór operatora; null = brak

    "route": {
      "mode": "AUTO|MANUAL",                  // AUTO – system wybiera TARGET;
MANUAL – operator wskazuje
      "manualTarget": "<PLACE_ID|null>",       // dozwolone tylko dla miejsc z
config.allowManualTargets
      "releaseKind": "auto|waste|wrap|strap|ship|custom",  // skrót intencji
wysyłki
      "customTarget": "<PLACE_ID|null>"        // uzupełnienie dla
releaseKind=custom
    },

    "intents": {
      "accept": true,        // pozwól dowozić (dla buforów/stanowisk
odbiorczych)
      "release": false,      // pozwól wywozić (dla buforów/stanowisk
nadawczych)
      "demand": false,       // jednorazowy impuls (produkcja – przywóz pustych)
      "shipment": false,     // jednorazowy impuls (produkcja – odbiór gotowego)
      "clear": false         // jednorazowy impuls – wyczyść wybór/odblokuj UI
    },

    "safety": {
      "pause": false,        // żądanie pauzy lokalnej (dla przypisanego AGV –
HMI)
      "resume": false        // żądanie wznowienia
    },

    "notes": "<string opcjonalnie>"
  }
}
```

Zasady: - `goodsTypeSelected` – UI blokuje zmianę w chwili, gdy trwa obsługa zgodna z tym wyborem (np. aktywne zadanie na rzecz miejsca) i odblokowuje po zakończeniu. - `route.mode=MANUAL` wymaga, by cel należał do listy `allowManualTargets` z konfiguracji miejsca. - Intenty `demand` / `shipment` / `clear` są impulsowe (UI po zapisie powinno je natychmiast wyzerować na `false` – bądź serwer UI może zerować po przyjęciu).

3.6.2 Struktura `CONTROL.tags` (zapisywana przez Orchestrator)

```
{
  "computedAt": "<ISO8601>",
  "source": "orchestrator@host",

  "preview": {                      // co zostanie zrobione, jeśli warunki będą
spełnione
    "action": "NONE|CREATE_TASK|CANCEL_TASK",
    "task": {
      "label": "<TASK_LABEL>",
      "params": {
        "GOODS_TYPE": "<TYPE>",
        "SOURCE": "<PLACE_ID>",
        "TARGET": "<PLACE_ID>",
        "FLOW_NAME": "<FLOW>"
      }
    },
    "reason": ["<krótkie kody uzasadnień>"],
    "dependsOn": ["<PLACE_ID>"],            // np. musi być wolne miejsce w
STRAP1
    "expiresAt": "<ISO8601|null>"
  },

  "decision": {                     // ostatnia wykonana decyzja
    "action": "NONE|CREATE_TASK|CANCEL_TASK",
    "taskRecordId": "<id|null>",
    "at": "<ISO8601>",
    "result": "OK|ERROR",
    "errorMsg": "<opcjonalnie>"
  },

  "locks": {                        // miękkie blokady anty-duplikacyjne
(idempotencja)
    "createGuardUntil": "<ISO8601|null>",
    "cancelGuardUntil": "<ISO8601|null>"
  },

  "statusHints": {                  // wskazówki dla UI
    "canCancelSafely": true,        // na bazie statusDescription z taska
    "blockedBecause": ["VIEW_STALE","NO_FREE_CAPACITY"],
    "agvDispatchableFixQueued": false
  }
}
```

Zasady: - `preview` nie tworzy skutków – to tylko podgląd. Orchestrator aktualizuje go często, aby UI mogło „widzieć przyszłość”. - `decision` rejestruje ostatnią decyzję faktyczną (utworzenie/usunięcie zadania) wraz z wynikiem i ewentualnym błędem. - `statusHints.canCancelSafely` pochodzi z odczytu `status_description` taska (JSON osadzony przez definicję taska, opisujący np. możliwość bezpiecznego przerwania); jeśli jednocześnie `task.agv_id != null` i `canCancelSafely=true`, po anulowaniu Orchestrator przywraca robota do `dispatchable`.

3.6.3 Minimalne wymagania per rodzaj miejsca

type	Wymagane w `CONTROL.content.ui`	Uwagi
BUFFER_LANE	`goodsTypeSelected`, `intents.accept/release`, `route.mode` (opcjonalnie manual target)	lane przyjmuje/oddaje wg `accept/release`; manual target zgodny z konfiguracją
PROD_SLOT	`goodsTypeSelected`, `intents.demand` (P1) / `intents.shipment` (P3), `route.releaseKind`	P1: dowóz pustych; P3: odbiór gotowego; dodatkowe `releaseKind` dla odpadów itp.
WRAP_STATION	`intents.accept/release`	+ ewentualne `releaseKind=strap`
SHIP_STATION	`intents.accept/release`	+ `releaseKind=ship`

3.6.4 Przykłady CONTROL dla wybranych miejsc

A) EMPTY_BUF_L1 (BUFFER_LANE)

```
{
  "placeId": "EMPTY_BUF_L1",
  "ts": "2025-10-11T11:40:00Z",
  "ui": {
    "goodsTypeSelected": "BOX",
    "route": { "mode": "AUTO", "manualTarget": null, "releaseKind": "auto",
"customTarget": null },
    "intents": { "accept": true, "release": false, "demand": false, "shipment":
false, "clear": false },
    "safety": { "pause": false, "resume": false }
  }
}
```

B) PROD1_P1 (PROD_SLOT – przyjęcie pustych)

```
{
  "placeId": "PROD1_P1",
  "ts": "2025-10-11T11:41:00Z",
  "ui": {
    "goodsTypeSelected": "EUR_PAL",
    "route": { "mode": "AUTO", "manualTarget": null, "releaseKind": "auto",
"customTarget": null },
    "intents": { "accept": true, "release": false, "demand": true, "shipment":
false, "clear": false },
    "safety": { "pause": false, "resume": false }
  }
}
```

C) PROD1_P3 (PROD_SLOT – odbiór gotowego, z opcją odpadu)

```
{
  "placeId": "PROD1_P3",
  "ts": "2025-10-11T11:42:00Z",
  "ui": {
    "goodsTypeSelected": "EUR_PAL",
    "route": { "mode": "AUTO", "manualTarget": null, "releaseKind": "auto",
"customTarget": null },
    "intents": { "accept": false, "release": true, "demand": false, "shipment":
true, "clear": false },
    "safety": { "pause": false, "resume": false },
    "notes": "W razie przezbrojenia użyj releaseKind=waste"
  }
}
```

D) WRAP1 (WRAP_STATION)

```
{
  "placeId": "WRAP1",
  "ts": "2025-10-11T11:43:00Z",
  "ui": {
    "goodsTypeSelected": null,
    "route": { "mode": "AUTO", "manualTarget": null, "releaseKind": "strap",
"customTarget": null },
    "intents": { "accept": true, "release": true, "demand": false, "shipment":
false, "clear": false },
    "safety": { "pause": false, "resume": false }
  }
}
```

E) SHIP1 (SHIP_STATION)

```
{
  "placeId": "SHIP1",
  "ts": "2025-10-11T11:44:00Z",
  "ui": {
    "goodsTypeSelected": null,
    "route": { "mode": "AUTO", "manualTarget": null, "releaseKind": "ship",
"customTarget": null },
    "intents": { "accept": true, "release": true, "demand": false, "shipment":
false, "clear": false },
    "safety": { "pause": false, "resume": false }
  }
}
```

Ten model pozwala Orchestratorowi w sposób jednolity czytać intencje operatora, dobrać parametry taska (`GOODS_TYPE`, `SOURCE`, `TARGET`, `FLOW_NAME`) oraz prezentować w `CONTROL.tags.preview` jasny powód decyzji i planowany cel. UI może na podstawie `preview` pokazywać, **dokąd** pojedzie towar i **dlaczego**.

3.5 Uniwersalny model `OCCUPANCY` (content & tags) — PRZYWRÓCONE

Ten rozdział definiuje **spójny model** opisu zajętości i pojemności dla wszystkich typów miejsc (places). `OCCUPANCY` jest **jedynym** źródłem prawdy o stanie fizycznym miejsca — wyliczanym przez **Occupancy Managera**. Orchestrator i UI **nie liczą** stanu, a jedynie go odczytują.

3.5.1 Struktura `..._OCCUPANCY.content`

```
{
  "placeId": "<PLACE_ID>",
  "ts": "<ISO8601>",
  "type": "BUFFER_LANE|WRAP_STATION|PROD_SLOT|SHIP_STATION|WASTE_BUF",
  "capacity": {
    "depth": 0,
    "maxStack": 0
  },
  "state": {
    "goodsTypeCurrent": "<TYPE|null>",
    "stackHeights": [0],
    "acceptWindow": { "enabled": true, "reason": null },
    "releaseWindow": { "enabled": true, "reason": null }
  },
  "locks": {
    "reserved": 0,
```

```
    "blocked": false,
    "blockedReason": null
  },
  "view": {
    "totalSlots": 0,
    "occupiedSlots": 0,
    "freeSlots": 0,
    "inflightInbound": 0,
    "inflightOutbound": 0
  },
  "stale": false,
  "errors": []
}
```

Wyjaśnienia kluczowych pól - `capacity.depth` – ile pozycji w głąb (liczba „kolumn" od frontu do końca alejki). 0 dla miejsc bez logiki alejki. - `capacity.maxStack` – maksymalna wysokość stosu (liczba „pięter"). 0 dla miejsc bez logiki stosu. - `state.stackHeights` – tablica długości `depth`; każda pozycja to liczba palet w danej kolumnie (0…`maxStack`). - `state.goodsTypeCurrent` – typ obsługiwany aktualnie przez alejkę/miejsce (dla `PROD_SLOT` może być wyprowadzony z `CONTROL.content`). - `acceptWindow` / `releaseWindow` – bramki przyjęcia/oddania z przyczyną zamknięcia (np. kontrola jakości). - `locks.reserved` – liczba **zarezerwowanych** miejsc (przed faktycznym zajęciem), utrzymywana przez Occupancy Managera. - `view.*` – **wyłącznie pochodne** pola, zawsze liczone przez Occupancy Managera: - `totalSlots = depth * maxStack` (dla `BUFFER_LANE`), w innych typach może być 1 lub 0. - `occupiedSlots = sum(stackHeights)`. - `freeSlots = totalSlots - occupiedSlots - reserved`. - `inflightInbound` / `inflightOutbound` – liczby informacyjne na podstawie aktywnych tasków przypisanych do tego `placeId`. - `stale=true` – gdy Occupancy Manager podejrzewa, że obraz nie jest aktualny (np. minął watchdog czasu).

3.5.2 Struktura `…_OCCUPANCY.tags`

```
{
  "lastUpdateTs": "<ISO8601>",
  "health": "OK|WARN|ERROR",
  "calcVersion": "occ-v1",
  "anomaliesSummary": []
}
```

3.5.3 Przykłady

A) Alejka bufora pustych opakowań (`EMPTY_BUF_L1`)

```
{
  "placeId": "EMPTY_BUF_L1",
  "ts": "2025-10-11T10:00:00Z",
  "type": "BUFFER_LANE",
```

```
  "capacity": { "depth": 5, "maxStack": 3 },
  "state": {
    "goodsTypeCurrent": "EUR_PAL",
    "stackHeights": [3, 3, 2, 0, 0],
    "acceptWindow": { "enabled": true, "reason": null },
    "releaseWindow": { "enabled": true, "reason": null }
  },
  "locks": { "reserved": 1, "blocked": false, "blockedReason": null },
  "view": { "totalSlots": 15, "occupiedSlots": 8, "freeSlots": 6,
"inflightInbound": 1, "inflightOutbound": 0 },
  "stale": false,
  "errors": []
}
```

B) Port produkcyjny wejściowy (`PROD1_P1`) – bez kolumn/stosu

```
{
  "placeId": "PROD1_P1",
  "ts": "2025-10-11T10:00:00Z",
  "type": "PROD_SLOT",
  "capacity": { "depth": 0, "maxStack": 0 },
  "state": {
    "goodsTypeCurrent": "EUR_PAL",
    "stackHeights": [],
    "acceptWindow": { "enabled": true, "reason": null },
    "releaseWindow": { "enabled": false, "reason": "input-only" }
  },
  "locks": { "reserved": 0, "blocked": false, "blockedReason": null },
  "view": { "totalSlots": 1, "occupiedSlots": 0, "freeSlots": 1,
"inflightInbound": 0, "inflightOutbound": 0 },
  "stale": false,
  "errors": []
}
```

C) Owijarka – stacja + bufor „frontowy” (`WRAP1`)

```
{
  "placeId": "WRAP1",
  "ts": "2025-10-11T10:00:00Z",
  "type": "WRAP_STATION",
  "capacity": { "depth": 2, "maxStack": 1 },
  "state": {
    "goodsTypeCurrent": null,
    "stackHeights": [1, 0],
```

```
    "acceptWindow": { "enabled": true, "reason": null },
    "releaseWindow": { "enabled": false, "reason": "wrapping" }
  },
  "locks": { "reserved": 0, "blocked": false, "blockedReason": null },
  "view": { "totalSlots": 2, "occupiedSlots": 1, "freeSlots": 1,
"inflightInbound": 0, "inflightOutbound": 0 },
  "stale": false,
  "errors": []
}
```

D) Alejka bufora wysyłki (`SHIP_BUF_L1`)

```
{
  "placeId": "SHIP_BUF_L1",
  "ts": "2025-10-11T10:00:00Z",
  "type": "BUFFER_LANE",
  "capacity": { "depth": 6, "maxStack": 2 },
  "state": {
    "goodsTypeCurrent": "BOX",
    "stackHeights": [2, 2, 1, 0, 0, 0],
    "acceptWindow": { "enabled": true, "reason": null },
    "releaseWindow": { "enabled": true, "reason": null }
  },
  "locks": { "reserved": 2, "blocked": false, "blockedReason": null },
  "view": { "totalSlots": 12, "occupiedSlots": 5, "freeSlots": 5,
"inflightInbound": 1, "inflightOutbound": 1 },
  "stale": false,
  "errors": []
}
```

E) Bufor odpadów (`WASTE_BUF`)

```
{
  "placeId": "WASTE_BUF",
  "ts": "2025-10-11T10:00:00Z",
  "type": "WASTE_BUF",
  "capacity": { "depth": 3, "maxStack": 1 },
  "state": {
    "goodsTypeCurrent": null,
    "stackHeights": [1, 0, 0],
    "acceptWindow": { "enabled": true, "reason": null },
    "releaseWindow": { "enabled": true, "reason": null }
  },
  "locks": { "reserved": 0, "blocked": false, "blockedReason": null },
```

```
  "view": { "totalSlots": 3, "occupiedSlots": 1, "freeSlots": 2,
"inflightInbound": 0, "inflightOutbound": 0 },
  "stale": false,
  "errors": []
}
```

3.6.5 `CONTROL.content` – wymagane pola per typ miejsca

Poniższe mini-szablony pokazują **minimalny** zestaw pól, które UI powinien zapisywać w `..._CONTROL.content` dla danego typu miejsca. Pola dodatkowe (np. `notes`) wolno dodawać – nie wpływa to na kompatybilność.

A) BUFFER_LANE (np. EMPTY_BUF_L1, SHIP_BUF_L2, STRAP1)

```
{
  "placeId": "<BUFFER_LANE_ID>",
  "ts": "<ISO8601>",
  "ui": {
    "goodsTypeSelected": "<TYPE|null>",
    "route": {
      "mode": "AUTO|MANUAL",
      "manualTarget": "<PLACE_ID|null>",
      "releaseKind": "auto"
    },
    "intents": { "accept": true|false, "release": true|false }
  }
}
```

Wymagania: - `goodsTypeSelected` – *wymagane*, gdy lane ma być używany do **accept** (przyjęć) lub **release** (wydań) dla konkretnego towaru. - `route.mode=MANUAL` dozwolone tylko, jeśli miejsce ma w konfiguracji `allowManualTargets`.

B) PROD_SLOT – port wejściowy (np. `PROD1_P1`)

```
{
  "placeId": "PROD1_P1",
  "ts": "<ISO8601>",
  "ui": {
    "goodsTypeSelected": "<TYPE>",
    "route": { "mode": "AUTO", "releaseKind": "auto" },
    "intents": { "accept": true, "demand": true, "release": false }
```

```
  }
}
```

Wymagania: - `goodsTypeSelected` – *wymagane* (typ pustych opakowań dla danej serii). - `intents.demand=true` jest impulsem „przywieź teraz pusty nośnik”. UI powinno wyzerować po zapisie.

C) PROD_SLOT – port wyjściowy (np. `PROD1_P3`)

```
{
  "placeId": "PROD1_P3",
  "ts": "<ISO8601>",
  "ui": {
    "goodsTypeSelected": "<TYPE>",
    "route": { "mode": "AUTO", "releaseKind": "auto" },
    "intents": { "release": true, "shipment": true }
  }
}
```

Wymagania: - `shipment=true` jest impulsem „odbierz gotowy produkt teraz”. - `route.releaseKind` może przyjmować skróty konfiguracyjne (np. `waste`, `wrap`, `strap`, `ship`), jeśli dozwolone.

D) WRAP_STATION (owijarka)

```
{
  "placeId": "WRAP1",
  "ts": "<ISO8601>",
  "ui": {
    "goodsTypeSelected": null,
    "route": { "mode": "AUTO", "releaseKind": "strap" },
    "intents": { "accept": true, "release": true }
  }
}
```

Wymagania: - `accept=true` – pozwól zapełniać wejście (lub bufor WRAP), - `release=true` – operator potwierdza gotowość do wywozu dalej (np. do STRAP).

E) SHIP_STATION (stacja wysyłki / rolotok)

```
{
  "placeId": "SHIP1",
  "ts": "<ISO8601>",
  "ui": {
    "goodsTypeSelected": null,
```

```
    "route": { "mode": "AUTO", "releaseKind": "ship" },
    "intents": { "accept": true, "release": true }
  }
}
```

Wymagania: - `accept=true` – tryb przyjmowania z `SHIP_BUF` (możliwa czasowa pauza jakościowa), - `release=true` – pozwolenie na wywóz (jeśli używamy strefy „ostatniej mili”).

F) WASTE_BUF (bufor odpadu)

```
{
  "placeId": "WASTE_BUF",
  "ts": "<ISO8601>",
  "ui": {
    "goodsTypeSelected": null,
    "route": { "mode": "AUTO", "releaseKind": "auto" },
    "intents": { "accept": true, "release": true }
  }
}
```

Wymagania: - `accept=true` – odbiór odpadów z produkcji, - `release=true` – wywóz odpadu dalej (jeśli przewidziany).

> **Uwaga ogólna:** Intenty impulsowe (`demand`, `shipment`, `clear`) powinny być zerowane przez UI zaraz po zapisie. Blokady zmian `goodsTypeSelected` w czasie aktywnego zlecenia są realizowane po stronie UI (z ewentualnym wsparciem Orchestratora w `CONTROL.tags.statusHints`).

4. Konfiguracja systemu

> Ta sekcja definiuje **statyczną konfigurację** wykorzystywaną przez UI, Orchestrator oraz Occupancy Managera. Konfiguracja jest niezależna od bieżącego stanu miejsc i zadań; opisuje słowniki, rejestr miejsc, biblioteki przepływów i mapowania zadań.

4.1 Słownik towarów (`goods`)

Minimalny schemat wpisu:

```
{
  "id": "EUR_PAL",
  "label": "Paleta EUR",
  "flowName": "FLOW_EUR_STD",
  "recognitionProfile": "eur_pal_v1",
  "stacking": { "maxStack": 3 },
```

```
  "dimensions": { "w": 800, "l": 1200, "h": 145 },
  "notes": "opcjonalne"
}
```

- `flowName` – nazwa z 4.3 (wiąże towar z przepływem). - `recognitionProfile` – profil rozpoznawania palet / ładunku używany przez RDS. - `stacking.maxStack` – domyślny limit pięter dla alejki (może być nadpisany per miejsce).

4.2 Rejestr miejsc (`places`)

Każde fizyczne miejsce ma dwa RDS worksites: `..._CONTROL` i `..._OCCUPANCY`.

```
{
  "id": "EMPTY_BUF_L1",
  "type": "BUFFER_LANE",
  "worksites": {
    "control": "EMPTY_BUF_L1_CONTROL",
    "occupancy": "EMPTY_BUF_L1_OCCUPANCY"
  },
  "capacity": { "depth": 5, "maxStack": 3 },
  "allowManualTargets": true,
  "policies": {
    "accept": { "enabled": true },
    "release": { "enabled": true }
  },
  "routingHints": {
    "defaultTargets": ["PROD1_P3", "PROD2_P3"],
    "priority": 10
  }
}
```

- `capacity` – domyślna pojemność (Occupancy Manager może to nadpisać dynamicznie, jeśli miejsce jest rekonfigurowalne). - `allowManualTargets` – czy UI może ustawić `route.mode=MANUAL`. - `routingHints.defaultTargets` – podpowiedzi dla trybu AUTO (nieobowiązkowe).

Specjalizacje typu: - `PROD_SLOT` może mieć `role: "INPUT"|"OUTPUT"` i stałe powiązanie z linią (np. `lineId: "L1"`). - `WRAP_STATION` może mieć `bufferLaneIds`: np. `{"front": "WRAP1_BUF_L1", "overflow": "WRAP1_BUF_L2"}`. - `SHIP_STATION` może mieć `shipBufferLaneIds`: lista lane'ów powiązanych.

4.3 Biblioteka przepływów (`flows`)

Każdy przepływ składa się z uporządkowanych kroków z warunkami wejścia i regułami mapowania celu.

```
{
  "name": "FLOW_EUR_STD",
  "label": "EUR: standard",
  "steps": [
    {
      "id": "S1_INBOUND_EMPTY",
      "from": { "type": "BUFFER_LANE", "placeRef":
["EMPTY_BUF_L1","EMPTY_BUF_L2"] },
      "to":   { "type": "PROD_SLOT",  "placeRef": ["PROD1_P1","PROD2_P1"] },
      "targetQuery": { "rule": "matchByGoodsType", "args": {} },
      "bind": [ { "key": "lineId", "from": "target.placeId:PROD(\d+)_",
"transform": "extractLine" } ],
      "task": { "label": "bring_goods", "params":
["GOODS_TYPE","SOURCE","TARGET","FLOW_NAME"] }
    },
    {
      "id": "S2_OUT_PROD",
      "from": { "type": "PROD_SLOT",  "placeRef": ["PROD1_P3","PROD2_P3"] },
      "to":   { "type": "WRAP_STATION", "placeRef": ["WRAP1"] },
      "targetQuery": { "rule": "wrapStationWithBuffer", "args": {} },
      "bind": [ { "key": "lineId", "from": "source.placeId:PROD(\d+)_",
"transform": "extractLine" } ],
      "task": { "label": "move_goods", "params":
["GOODS_TYPE","SOURCE","TARGET","FLOW_NAME","lineId"] }
    },
    {
      "id": "S3_WRAP_TO_STRAP",
      "from": { "type": "WRAP_STATION", "placeRef": ["WRAP1"] },
      "to":   { "type": "BUFFER_LANE",  "placeRef": ["STRAP1","STRAP2"] },
      "targetQuery": { "rule": "matchByLineId", "args": {} },
      "bind": [ { "key": "lineId", "from": "prev.lineId" } ],
      "task": { "label": "move_goods", "params":
["GOODS_TYPE","SOURCE","TARGET","FLOW_NAME","lineId"] }
    },
    {
      "id": "S4_STRAP_TO_SHIPBUF",
      "from": { "type": "BUFFER_LANE",  "placeRef": ["STRAP1","STRAP2"] },
      "to":   { "type": "BUFFER_LANE",  "placeRef":
["SHIP_BUF_L1","SHIP_BUF_L2","SHIP_BUF_L3"] },
      "targetQuery": { "rule": "shipLaneByGoodsOrAssignEmpty", "args": {
"autobind": true } },
      "bind": [ { "key": "shipLaneId", "from": "target.placeId" } ],
      "task": { "label": "move_goods", "params":
["GOODS_TYPE","SOURCE","TARGET","FLOW_NAME","shipLaneId"] }
    },
    {
      "id": "S5_SHIPBUF_TO_SHIP",
```

```
      "from": { "type": "BUFFER_LANE",  "placeRef":
["SHIP_BUF_L1","SHIP_BUF_L2","SHIP_BUF_L3"] },
      "to":   { "type": "SHIP_STATION", "placeRef": ["SHIP1"] },
      "targetQuery": { "rule": "shipStationAccept", "args": {} },
      "task": { "label": "move_goods", "params":
["GOODS_TYPE","SOURCE","TARGET","FLOW_NAME"] }
    }
  ]
}
```

- `targetQuery.rule` – odwołuje się do nazwanych reguł selekcji (implementowanych w Orchestratorze lub Occupancy Managerze), np.: `matchByGoodsType`, `wrapStationWithBuffer`, `matchByLineId`, `shipLaneByGoodsOrAssignEmpty`, `shipStationAccept`. - `bind` – jak przenosić kontekst między krokami (np. `lineId`).

Drugi, krótszy przepływ dla BOX:

```
{ "name": "FLOW_BOX_STD", "label": "BOX: standard",
  "steps": [
    { "id": "B1_EMPTY_TO_PROD",  "from": {"type":"BUFFER_LANE","placeRef":
["EMPTY_BUF_L1","EMPTY_BUF_L2"]}, "to": {"type":"PROD_SLOT","placeRef":
["PROD1_P1","PROD2_P1"]}, "targetQuery": {"rule":"matchByGoodsType","args":{}},
"task": {"label":"bring_goods","params":
["GOODS_TYPE","SOURCE","TARGET","FLOW_NAME"]} },
    { "id": "B2_PROD_TO_SHIPBUF","from": {"type":"PROD_SLOT","placeRef":
["PROD1_P3","PROD2_P3"]}, "to": {"type":"BUFFER_LANE","placeRef":
["SHIP_BUF_L1","SHIP_BUF_L2","SHIP_BUF_L3"]}, "targetQuery":
{"rule":"shipLaneByGoodsOrAssignEmpty","args":{"autobind":true}}, "task":
{"label":"move_goods","params":["GOODS_TYPE","SOURCE","TARGET","FLOW_NAME"]} },
    { "id": "B3_SHIPBUF_TO_SHIP","from": {"type":"BUFFER_LANE","placeRef":
["SHIP_BUF_L1","SHIP_BUF_L2","SHIP_BUF_L3"]}, "to":
{"type":"SHIP_STATION","placeRef":["SHIP1"]}, "targetQuery":
{"rule":"shipStationAccept","args":{}}, "task": {"label":"move_goods","params":
["GOODS_TYPE","SOURCE","TARGET","FLOW_NAME"]} }
  ]
}
```

4.4 Mapowanie zadań RDS (`tasks`)

Definicje szablonów tasków oraz ich parametrów wejściowych, które Orchestrator będzie używać przy tworzeniu instancji.

```
{
  "bring_goods": {
```

```
    "rdsLabel": "bring_goods",
    "requiredParams": ["GOODS_TYPE","SOURCE","TARGET","FLOW_NAME"]
  },
  "move_goods": {
    "rdsLabel": "move_goods",
    "requiredParams": ["GOODS_TYPE","SOURCE","TARGET","FLOW_NAME"],
    "optionalParams": ["lineId","shipLaneId"]
  }
}
```

Uzgadniamy, że na tym etapie wystarczą parametry: `GOODS_TYPE`, `SOURCE`, `TARGET`, `FLOW_NAME`. Dodatkowe (np. `lineId`) można dodać w razie potrzeby.

4.5 Uprawnienia ręcznych destynacji (`manualRouting`)

Określa, gdzie UI może kierować ładunek w trybie `MANUAL`.

```
{
  "PROD1_P3": ["WRAP1","STRAP1","WASTE_BUF","SHIP_BUF_L1"],
  "PROD2_P3": ["WRAP1","STRAP2","WASTE_BUF","SHIP_BUF_L2"],
  "SHIP_BUF_L1": ["SHIP1"],
  "WRAP1": ["STRAP1","STRAP2"]
}
```

4.6 Reguły selekcji celu (`targetRules`)

Implementacje nazw używanych w `targetQuery.rule`. Każda reguła deklaruje wymagane dane wejściowe oraz semantykę.

```
{
  "matchByGoodsType": {
    "needs": ["GOODS_TYPE"],
    "desc": "Wybiera miejsca zdolne przyjąć wskazany goodsType i mające
freeSlots>0"
  },
  "wrapStationWithBuffer": {
    "needs": [],
    "desc": "Jeśli WRAP zajęta, wybierz jej bufor; inaczej stację WRAP"
  },
  "matchByLineId": {
    "needs": ["lineId"],
    "desc": "Mapuje lineId→STRAP lane (np. L1→STRAP1, L2→STRAP2)"
  },
  "shipLaneByGoodsOrAssignEmpty": {
```

```
    "needs": ["GOODS_TYPE"],
    "desc": "Jeżeli istnieje lane z przypisanym goodsType i freeSlots>0 –
wybierz ją; w p.p. przypisz pierwszy pusty lane (autobind)"
  },
  "shipStationAccept": {
    "needs": [],
    "desc": "Wybierz stację wysyłki, jeśli ma acceptWindow.enabled=true"
  }
}
```

4.7 Walidacja i domyślne wartości

- Braki w `capacity` dla `BUFFER_LANE` uzupełniamy z `goods.stacking.maxStack` (jeśli zdefiniowane) i z domyślnego `depth` miejsca.
- `allowManualTargets=false` powoduje ignorowanie `route.mode=MANUAL` na UI (UI powinien to blokować, ale Orchestrator też ma to respektować).
- Jeśli `goodsTypeSelected=null` dla lane'u z `accept=true`, Orchestrator może zgłosić `statusHints.errors` (CONTROL.tags) i zablokować automatyczne przyjęcia.
- `targetQuery` bez wyników → brak tworzenia taska + `statusHints.suggestions` z powodem.