Index

Fleet Manager Overview

Fleet manager is ASGI REST API server implementation which can be used to monitor, control fleet of sherpas.

Fleet manager functionalities

1. **Configure/operate fleets**
    1. Configure stations, sherpas with the business use case(Auto-unhitch, Auto hitch, Roller-Top operation)
    2. Operate the fleet by booking/deleting trips
2. **Optimal task assignment**
    1. Assign each trip booking with the best possible sherpa
    2. Configurable dispatch logic to
        - Minimise wait time
        - Maximal throughput
3. **Facilitate any external engagement**
    1. Facilitate communication between two **Ati Sherpas**
    2. Facilitate communication between a **Ati Sherpa** and another **smart device** (For example conveyor present in shop floor or a non Ati bot operating on the same shop floor)
4. **Traffic management**
    1. To prevent deadlocks in constricted pathways, intersections and sherpa blind spots
5. **Dynamic control of fleet**
    1. Fleet level - Initiate end of shift protocol, stop/pause fleet operations
    2. Sherpa level - Pause/Resume sherpa's movement, Stop a particular sherpa from carrying out fleet operations for the time being, facilitate maintenance

Fleet Manager Architecture

There are two types of messages/requests that are excahnged between Fleet manager and other agents like sherpas, dashboard(end consumer), smart devices

```
- Periodic updates/messages
- Aperiodic/event-driven messages
```

Periodic updates/messages

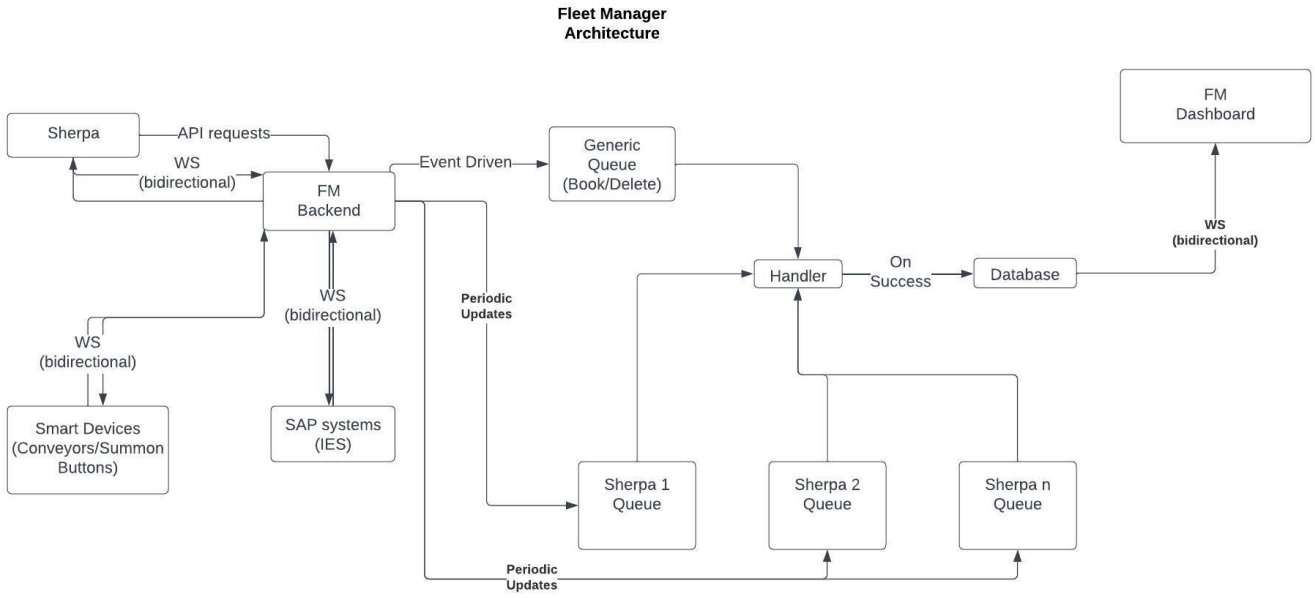
Periodic messages typically include pose/trip updates from sherpas or updates from smart devices(num totes present on conveyor at any point to time). These messages are used to monitors sherpa/smart

devices. All the periodic messages to and from FM are exchanges via websockets. Messages from different sources are processed parallelly.

Aperiodic/event-driven messages

Event driven message/request would usually mean start/end of task like sherpa reaching a station, user booking a trip. The time order of all these message needs to be preserved, have to be processed sequentially. These messages are excahnged in http request-response.

All the messages/requests from multiple agents are queued using redis-queue and processed by a python program **handlers.py** which basically has the business logic, and if the request/message is processed successfully the state changes correponding to the message/requests are recorded in the DB(postgresql).



Examples of prominent messages/requests

1. Sherpa status - Real time update from sherpa to FM via websockets

```
{
   "type": "sherpa_status",
   "timestamp": double
   "sherpa_name": str
   "current_pose": Array containing x,y, theta
   "battery_status": int
   "mode": "manual"/"fleet"/"auto" etc
   "error": bool
   "error_info": str //only if error is True
}
```

2. Trip booking request - HTTP requests from dashboard app/SAP system plugins

```
URL: /api/v1/trips/book
METHOD: post
Parameters required
                1) X-User-Token: string,          in: header
Content-Type: application/json
Request Body Schema:
                Source: string
                Ttl: integer
                Trips: array
                Type: string

                # schema of trips
                "trips": [
                    {
                        "route": ["<station name>", ...],
                        "priority": 0,    // 0 is the lowest priority
                        "metadata": {"scheduled": False,
                                    "scheduled_time_period": time in secs
                                    "scheduled_start_time": "%Y-%m-%d
%H:%M:%S"
                                    "scheduled_end_time": "%Y-%m-%d
%H:%M:%S"
                        }
                    },
                ]
```

3. Trip status request - HTTP request

```
URL: /api/v1/trips/status
METHOD: post
Parameters required
                1) X-User-Token: string,          in: header
Content-Type: application/json
Request Body Schema:
                Source: string
                Ttl: integer
                Booked From: string
                Booked Till: string
                Trip Ids: array
```

4. Delete Trips - HTTP request

```
URL: /api/v1/trips/booking/{booking_id}/{trip_id}
METHOD: delete
Parameters required
                1) Booking Id: integer,           in: path
                2) Trip Id: integer,              in: path
```

```
                3) X-User-Token: string,          in: header
```

5. Disable station - HTTP request

```
URL: /api/v1/station/{entity_name}/disable/{disable}
METHOD: get
Parameters required
                1) Entity Name: string,           in: path
                2) Disable: boolean,              in: path
                3) X-User-Token: string,          in: header
```

6. Get all the information regarding sherpas - HTTP request

```
URL: /api/v1/configure_fleet/all_sherpa_info
METHOD: get
Parameters required
                1) X-User-Token: string,          in: header
```

7. Add/edit a fleet - HTTP request

```
URL: /api/v1/configure_fleet/add_edit_fleet/{fleet_name}
METHOD: post
Parameters required
                1) Fleet Name: string,            in: path
                2) X-User-Token: string,          in: header
Content-Type: application/json
Request Body Schema:
                Source: string
                Ttl: integer
                Site: string
                Location: string
                Customer: string
                Map Name: string
```

8. Pause/Unpause a sherpa, Emergency stop - HTTP request

```
URL: /api/v1/control/sherpa/{entity_name}/emergency_stop
METHOD: post
Parameters required
                1) Entity Name: string,           in: path
                2) X-User-Token: string,          in: header
Content-Type: application/json
Request Body Schema:
                Source: string
```

```
                    Ttl: integer
                    Pause: boolean
```