Specifiche tecniche di integrazione

1. Introduzione

Il presente documento ha lo scopo di definire le specifiche tecniche necessarie all'integrazione tra i sistemi dell'applicazione IF – Infomobilità Firenze del Comune di Firenze e i sistemi dei servizi di sharing mobility da collegare alla piattaforma.
Le specifiche, i requisiti e i flussi descritti di seguito costituiscono il livello minimo richiesto per garantire l'interoperabilità e consentire l'utilizzo dei mezzi di sharing all'interno delle funzionalità MaaS dell'app IF.

2. Standard interscambio dati

MDS - Mobility Data Specification
https://github.com/openmobilityfoundation/mobility-data-specification
GBFS - General Bikeshare Feed Specification
https://github.com/MobilityData/gbfs

3. Requisiti minimi del flusso GBFS

Le funzioni dedicate alla mobilità (MaaS) all'interno dell'app IF, utilizzano le API dei provider esterni per permettere agli utenti di fruire in modo semplice e dinamico di molteplici servizi connessi alla mobilità urbana. Tra questi, rivestono grande importanza tutti i servizi dedicati al noleggio di mezzi di sharing per gli spostamenti.
A prescindere dalla modalità con cui gli utenti potranno trovare, visualizzare, acquistare ed utilizzare i servizi proposti, esistono alcuni requisiti necessari per consentire il funzionamento dell'app e l'integrazione dei mezzi di sharing del provider.
Nel seguito sarà presentata una panoramica delle specifiche tecniche. I requisiti necessari per l'integrazione tra il nostro sistema e quello di un provider di sharing generico sono:

- **GBFS** → Necessario per il funzionamento del motore di pianificazione viaggio, per la ricezione delle disponibilità, della posizione dei mezzi etc.;
- **API Operatore standard** → Necessarie per l'interazione con il mezzo;
- **MDS Provider** → per eventi, telemetria, analisi (opzionale).

Ciascuna di queste macro-componenti deve essere resa disponibile almeno in due ambienti distinti:

- **Staging/Test** → Ambiente di test per sviluppo e validazione della soluzione;
- **Production/Live** → Ambiente di produzione per l'utilizzo reale.

Il flusso **GBFS** deve rispettare dei requisiti minimi che consentano un corretto svolgimento ed erogazione del servizio. In particolare si richiedono le seguenti caratteristiche minime:

- Le informazioni contenute nel flusso, in particolar modo quelle contenute nel feed *vehicle_status.json* del GBFS v3.0, devono essere affidabili, precise e puntuali in modo tale che le informazioni riguardanti stato del mezzo, disponibilità e posizione siano aggiornate e corrette. Stante la strategicità di tale livello informativo, possono essere tollerate informazioni parziali o incorrette per un periodo di tempo limitato (10 giorni) e comunque per una percentuale non superiore al 5% del totale del parco mezzi messo a disposizione.
- La frequenza di aggiornamento del flusso deve essere sufficientemente elevata, per permettere un'esperienza utente conforme alle aspettative. Ci si attende che l'aggiornamento dei dati avvenga almeno ogni 60 secondi, considerando che comunque una frequenza di aggiornamento più elevata (< 30 secondi) è preferibile;
- La latenza nella risposta deve essere sufficientemente inferiore alla frequenza di aggiornamento del flusso GBFS e in nessun caso essere superiore;
- La tolleranza al numero e alla frequenza delle richieste deve essere sufficientemente elevata ovvero tra le 8 e le 16 chiamate al minuto;
- Il flusso GBFS deve seguire in modo quanto più preciso possibile lo standard di riferimento internazionale. E' condizione necessaria che venga assegnato un identificativo univoco per ogni veicolo (vehicle_id), che questo si mantenga inalterato nel tempo e soprattutto che vada a coincidere con quello utilizzato nelle API Operatore. Più in generale, è richiesto che gli identificativi utilizzati all'interno dei flussi GBFS debbano essere coerenti tra di loro e corrispondere con quelli utilizzati nelle API fornite dal provider per la gestione dei mezzi.

La fedeltà tra quanto riportato nel GBFS e quanto disponibile attraverso le API Operatore è un punto cruciale, in quanto entrambi verranno usati in modalità sinergica e dovranno permettere un'esperienza utente fluida e unificata.

Il provider dovrà esporre un feed GBFS conforme alla specifica ufficiale: https://github.com/MobilityData/gbfs
La versione minima richiesta è la 3.0: https://github.com/MobilityData/gbfs/blob/v3.0/gbfs.md.

La validazione del feed può essere eseguita tramite lo strumento ufficiale: https://gbfs-validator.mobilitydata.org/

4. Feed minimi richiesti

I seguenti feed sono considerati obbligatori per l'integrazione:

- gbfs.json (feed principale)
- system_information.json
- vehicle_status.json
- vehicle_types.json
- system_pricing_plans.json
- geofencing_zones.json

Nel caso siano da integrare servizi *station based* sono obbligatori anche i seguenti feed:

- station_information.json
- station_status.json

Tutti gli altri feed, sono da considerarsi comunque potenzialmente necessari, in funzione del tipo di integrazione che deve essere implementata e potranno essere concordati e richiesti al provider in fase di implementazione tecnica, sulla base delle esigenze specifiche del servizio.
Gli attributi minimi richiesti per ciascun file sono documentati nella specifica GBFS v3.0.
Attributi e/o Feed aggiuntivi (https://github.com/MobilityData/gbfs)

5. Disponibilità del GBFS e delle API in ambiente di test

Per consentire una corretta integrazione tra i servizi messi a disposizione dall'app IF e quelli forniti dal generico provider di sharing, deve essere messo a disposizione un ambiente di test/staging che rispecchi fedelmente le caratteristiche e i comportamenti dell'ambiente di produzione.
Questo ambiente di test dovrà permettere di simulare scenari e casi di test realistici, nei quali poter testare le principali funzioni come la verifica della disponibilità dei veicoli e l'esecuzione di prenotazioni, sblocchi e gestione dei viaggi. I dati messi a disposizione in ambiente di test dovranno permettere di effettuare test ripetibili in maniera automatizzata con esiti prevedibili, in modo da poter verificare la correttezza dell'integrazione in ogni fase del processo.

6. Dettaglio sulle zone operative

Per abilitare la funzionalità riguardante le zone operative dei mezzi di sharing messi a disposizione dal provider è essenziale che venga esposto correttamente il file:

- *geofencing_zones.json*

secondo le specifiche contenute nello standard GBFS 3.0.

La gestione operativa come blocco/sblocco del mezzo o fine corsa fuori area consentita, è completamente a carico del provider, che dovrà implementarla secondo le proprie logiche interne.

Al momento dell'integrazione dell'operatore con IF, sarà necessaria una fase di progettazione tecnica dell'integrazione volta a definire:

- quali regole intende applicare sulle zone di operatività del veicolo (es. blocco sblocco o fine corsa fuori area, etc.);
- se tali informazioni verranno esposte tramite le API Operative.

7. Specifiche API Operatore Standard

Le API dell'operatore sono fondamentali per gestire le funzionalità core del servizio di micromobilità, come la prenotazione, il controllo del veicolo (lock/unlock) e la gestione della corsa (start/end ride).
Verranno utilizzate per integrare il sistema di backend con l'infrastruttura dell'operatore, permettendo agli utenti di interagire con i veicoli in modo fluido e sicuro.
Tramite i servizi di IF deve essere possibile avviare, terminare e storicizzare le sessioni di noleggio, oltre a permettere la gestione della logica di business legata alla tariffazione e al monitoraggio delle corse.
Nel seguito si riportano i Moduli API minimi obbligatori da esporre, necessari alla gestione dei viaggi MaaS; per quanto attiene gli endpoints sono da intendersi come esempi di best practice:

Modulo	Endpoint	Finalità
Availability	GET /vehicles	Stato flotta (complementare a GBFS)
	GET /vehicles/{id}	Stato del singolo mezzo
Vehicle Control	POST /vehicles/{id}/unlock	Sblocco del mezzo
	POST /vehicles/{id}/lock	Blocco del mezzo
Ride Management	POST /rides/start	Inizio viaggio
	POST /rides/end	Fine viaggio
	GET /rides	Lista paginata e filtrabile
	GET /rides/{id}	Dettaglio
User Management	POST /users	Creazione/recupero utente

Si richiama inoltre l'attenzione sulla strategicità dei successivi moduli che permettono un netto miglioramento nella user experience dagli utenti e pertanto, seppur non obbligatori è auspicabile che vengano implementati

Modulo	Endpoint	Finalità
Reservation	POST /reservations	Prenotazione del mezzo
	DELETE /reservations/{id}	Cancellazione prenotazione
	GET /reservations	Lista delle prenotazioni paginata e filtrabile
	GET /reservations/{id}	Stato prenotazione

Eventuali funzionalità aggiuntive potranno essere concordate con l'operatore in fase di implementazione tecnica, sulla base delle esigenze specifiche del servizio e sulla base della tipologia del mezzo messo a disposizione (un esempio potrebbe essere una gestione avanzata delle zone operative tramite API dedicate).

In generale comunque, tutte le API dovranno essere:

- RESTful
- JSON
- HTTPS
- Autenticate (preferibilmente OAuth2 Bearer Token)

E' infine necessario che, per ogni endpoint, sia assicurata una gestione appropriata degli errori standard HTTP 400, 401, 403, 404, 429 e 500. Altri ed eventuali codici di errore specifici potranno comunque essere necessari e stabiliti in fase di implementazione tecnica. Per facilitare il debug, ogni risposta di errore deve includere un request_id (se gestito lato operatore) univoco e un timestamp, oltre che un codice di errore e un messaggio descrittivo.

8. Esempio di implementazione e best practice

Al fine di agevolare ulteriormente la lettura del presente documento si riportano degli esempi utili alla comprensione generale.

a) Gestione degli errori

400 Bad Request

```
{
  "error": {
    "code": "XYZ",
    "message": "Validation Error: wrong parameter abcd",
    "details": [
      { "field": "abcd", "issue": "value_too_small", "expected": ">=30" }
    ],
    "request_id": "6f1b8a6e-...-e2d4",
    "timestamp": "2025-02-01T12:35:00Z"
  }
}
```

Il messaggio di errore deve essere chiaro e utile per il debug. Il codice di errore deve essere univoco per ogni tipo di errore e verrà utilizzato per fornire un messaggio parlante all'utente finale, ove necessario.

b) Availability API

Serve a sincronizzare lo stato dei veicoli con il sistema, andando oltre a ciò che GBFS espone.

GET /vehicles

Restituisce la lista dei mezzi con stato aggiornato.

Esempio di risposta:

200 OK


```
[
  {
    "vehicle_id": "op12345",
    "type": "bike",
    "form_factor": "bicycle",
    "propulsion_type": "electric",
    "status": "available",
    "battery_level": 80,
    "location": { "lat": 45.4654, "lon": 9.1859 },
    "last_update": "2025-02-01T12:34:56Z"
  },
  {
    "vehicle_id": "op23456",
    "type": "bike",
    "form_factor": "bicycle",
    "propulsion_type": "electric",
    "status": "available",
    "battery_level": 60,
    "location": { "lat": 45.4754, "lon": 9.1859 },
    "last_update": "2025-02-01T12:34:56Z"
  }
]
```


volendo cercare di mantenere una certa omogeneità tra API Operatore e flusso GBFS, di seguito si riporta un esempio di alcuni possibili stati standardizzati (GBFS, MDS):

- available
- reserved
- in_use
- out_of_service
- maintenance
- low_battery

GET /vehicles/{id}

Restituisce il dettaglio di un singolo mezzo.

Esempio di risposta:

200 OK


```
{
   "vehicle_id": "op12345",
   "type": "bike",
   "form_factor": "bicycle",
   "propulsion_type": "electric",
   "status": "available",
   "battery_level": 80,
   "location": { "lat": 45.4654, "lon": 9.1859 },
   "last_update": "2025-02-01T12:34:56Z"
 }
```


c) Vehicle Control API

Gli endpoint di vehicle control permettono il controllo remoto del mezzo.

POST /vehicles/{id}/unlock

Permette di sbloccare un mezzo.

Esempio di richiesta:


```
{
  "reservation_id": "res789",
  "user_id": "our_user_678"
}
```


Esempio di risposta:

200 OK


```
{
  "status": "unlocked",
  "timestamp": "2025-02-01T12:35:00Z"
}
```


POST /vehicles/{id}/lock

Permette di bloccare un mezzo.

Esempio di richiesta:


```
{
  "reservation_id": "res789",
```

```
  "user_id": "our_user_678"
}
```

Esempio di risposta:

200 OK

```
{
  "status": "locked",
  "timestamp": "2025-02-01T12:35:00Z"
}
```

d) Ride API

Endpoints che consentono di gestire le corse di un utente e di ottenerne le informazioni.

POST /rides/start

Permette l'inizio di una corsa per un utente.

Esempio di richiesta:

```
{
  "vehicle_id": "op12345",
  "reservation_id": "res789",
  "user_id": "our_user_678",
  "client_session_id": "abc-xyz-000"
}
```

Esempio di risposta:

201 CREATED

```
{
  "ride_id": "ride123",
  "vehicle_id": "op12345",
  "start_time": "2025-02-01T12:36:10Z"
}
```

POST /rides/end

Consente la chiusura di una corsa per un utente.

Esempio di richiesta:

```
{
  "ride_id": "ride123",
  "location": { "lat": 45.4679, "lon": 9.1921 }
}
```

Esempio di risposta:

200 OK

```
{
  "ride_id": "ride123",
  "start_time": "2025-02-01T11:48:00Z",
  "end_time": "2025-02-01T12:48:00Z",
  "duration_seconds": 710,
  "distance_meters": 3230,
  "pricing": {
    "total": 3.20,
    "currency": "EUR"
  }
}
```


GET /rides/{id}

Permette di recuperare le informazioni di una corsa tramite il suo identificativo.

Esempio di risposta:

200 OK


```
{
  "ride_id": "ride123",
  "start_time": "2025-02-01T11:48:00Z",
  "end_time": "2025-02-01T12:48:00Z",
  "duration_seconds": 710,
  "distance_meters": 3230,
  "pricing": {
    "total": 3.20,
    "currency": "EUR"
  }
}
```


Per allineare dati di sessione e tariffazione. I dati saranno popolati a seconda dello stato della corsa.

GET /rides

Garantisce il recupero delle corse attive o storicizzate, con supporto a filtri e paginazione.
Alcune opzioni di filtro necessarie, potrebbero essere relative a:

- vehicle_id → restituisce le corse relative a un determinato veicolo
- user_id → filtra le corse per utente
- status → filtra le corse per stato
- from / to → intervallo temporale (es. created_at, expires_at, etc)
- asc/desc → ordinamento temporale

L’ aggiunta di un filtro per ride_id potrebbe rappresentare una valida alternativa all’ implementazione separata dell endpoint GET /rides/{id}.

Alcuni suggerimenti riguardo alla paginazione, potrebbero riguardare:

- Parametri standard: limit (default 50), offset oppure page/page_size
- Inclusione nella risposta di parametri utili per l'interpretazione della paginazione come total, limit, offset (o page, page_size)

Un tipico caso d’ uso potrebbe essere quello del recupero dell’ultima corsa effettuata da un utente.

Esempio di risposta:

200 OK


```
{
  "data": [
    {
      "ride_id": "ride123",
      "vehicle_id": "op12345",
      "user_id": "our_user_678",
      "status": "ended",
      "start_time": "2025-02-01T12:36:10Z",
      "end_time": "2025-02-01T12:48:00Z",
      "duration_seconds": 710,
      "distance_meters": 3230,
      "pricing": {
        "total": 3.20,
        "currency": "EUR"
      }
    }
  ],
  "pagination": {
    "total": 240,
    "limit": 50,
    "offset": 0
  }
```


e) User API

Consistono negli endpoints che permettono la creazione o il recupero dell’utenza lato operatore, fondamentale per poterne utilizzare le api.

POST /users

Endpoint che consente la creazione di un nuovo utente lato operatore oppure restituisce l'utente esistente se lato operatore l'utenza esiste già. L'Api deve essere in grado di gestire unitamente sia la creazione che il recupero dell'utenza.
Il caso d' uso tipico e quello di creazione o recupero dell'identificativo dell'utente lato operatore necessario per interagire con gran parte delle api operator. E' importante che la creazione dell'utenza lato operatore richieda come unico parametro obbligatorio l'email, mentre eventuali metadati aggiuntivi siano lasciati opzionali (es: nome, cognome, telefono, etc).

Esempio di richiesta:

```
{
  "email": "user@example.com",
  "metadata": {
    "first_name": "Mario",
    "last_name": "Rossi"
  }
}
```

Esempio di risposta:

201 CREATED

```
{
  "user_id": "usr_123",
  "email": "user@example.com",
  "metadata": {
    "first_name": "Mario",
    "last_name": "Rossi"
  },
  "created_at": "2025-02-01T12:00:00Z"
}
```

se l'utente è stato creato con successo.

200 OK

```
{
  "user_id": "usr_123",
  "email": "user@example.com",
  "metadata": {
    "first_name": "Mario",
    "last_name": "Rossi"
  },
  "created_at": "2024-11-10T08:22:00Z"
}
```

se l'utente esiste già.

f) Reservation API

Gli endpoint di prenotazione consentono di riservare per un dato utente la disponibilità di un mezzo per un certo periodo di tempo. Di fondamentale importanza strategica, questi permettono di erogare un servizio di gestione operativa che sia perfettamente integrato con la fase di pianificazione.

POST/reservations

Permette di effettuare la prenotazione di un mezzo.

Esempio di richiesta:

```
{
  "vehicle_id": "op12345",
  "user_id": "our_user_678",
  "ttl_seconds": 300,
  "client_session_id": "abc-xyz-000"
}
```

Esempio di risposta:

201 CREATED

```
{
  "reservation_id": "res789",
  "vehicle_id": "op12345",
  "expires_at": "2025-02-01T12:39:00Z"
}
```

DELETE /reservations/{id}

Consente di cancellare una prenotazione non ancora attiva.

Esempio di risposta:

204 No Content

GET /reservations/{id}

Permette di verificare lo stato di una prenotazione.

Esempio di risposta:

200 OK

```
{
  "reservation_id": "res789",
  "vehicle_id": "op12345",
  "expires_at": "2025-02-01T12:39:00Z",
  "status": "active"
```

```
}
```

Come ulteriore suggerimento alcuni possibili stati potrebbero essere:

- active
- cancelled
- expired

GET /reservations

Garantisce il recupero delle prenotazioni attive o storicizzate, con supporto a filtri e paginazione.
Alcune opzioni di filtro necessarie, potrebbero essere relative a:

- vehicle_id → restituisce le prenotazioni relative a un determinato veicolo
- user_id → filtra le prenotazioni per utente
- status → filtra le prenotazioni per stato (es: active, cancelled, expired, etc)
- from / to → intervallo temporale (es. created_at, expires_at, etc)
- asc/desc → ordinamento temporale

L' aggiunta di un filtro per reservation_id potrebbe rappresentare una valida alternativa all' implementazione separata dell endpoint GET /reservations/{id}

Alcuni suggerimenti riguardo alla paginazione, potrebbero riguardare:

- Parametri standard: limit (default 50), offset oppure page/page_size
- Inclusione nella risposta di parametri utili per l'interpretazione della paginazione come total, limit, offset (o page, page_size)

Un tipico caso d' uso potrebbe essere quello del recupero dell'ultima prenotazione effettuata da un utente.

Esempio di risposta:
200 OK

```
{
  "data": [
    {
      "reservation_id": "res789",
      "vehicle_id": "op12345",
      "user_id": "our_user_678",
      "status": "active",
      "expires_at": "2025-02-01T12:39:00Z",
      "created_at": "2025-02-01T12:34:00Z"
    }
```

```
  ],
  "pagination": {
    "total": 128,
    "limit": 50,
    "offset": 0
  }
}
```