

# Identity Manager: Chimera API 5.0alpha3



- Sicurezza
  - Usi permessi
  - Credenziali
  - Autenticazione
  - Base-URL per gli endpoint
- Convenzioni
  - Errori
  - Identity e anagrafiche dei servizi federati
  - Eliminazione di Identity (o ritiro del consenso)
  - Sostituzione di Identity
- Guida all'implementazione
  - Casi d'uso
  - Note
    - Insiemi di dati sincronizzati e non
    - Aggiornamento del consenso GDPR
    - Precisazioni sull'uso della funzione di allineamento find\_changed\_identities
- Oggetti in response
  - Identity
  - IdentityHistory
  - Validation
  - ProviderAccount
- Funzioni: gestione delle Identity
  - POST: authenticate
  - GET: get\_identity
  - GET: find\_identity\_uid\_by\_email
  - GET: find\_identity\_uid\_by\_social\_id
  - POST: validate\_updating\_identity
  - POST: update\_identity
  - POST: update\_identity\_consent
  - POST: validate\_new\_identity
  - POST: add\_identity
  - POST: replace\_identity
  - POST: delete\_identity
- Funzioni: gestione del social login
  - POST: add\_provider\_account
  - POST: delete\_provider\_account
  - GET: find\_provider\_accounts
- Funzioni: federazione
  - GET: find\_changed\_identities
  - GET: find\_federations

- Formato dati e valori di lookup
  - Charset
  - <date>
  - <datetime>
  - <timestamp>
  - <socialId>
  - <consentrange>

## Sicurezza

### Usi permessi

Queste API sono progettate esclusivamente per un uso interno alla intranet e VPN Giunti, la loro finalità è esclusivamente il master data management tra servizi federati ("Federations"). Qualsiasi accesso alle funzionalità in modo diretto tramite client (app android, ios, windows) **non è assolutamente permesso**.

### Credenziali

Ogni servizio federato (Federation) è riconosciuto tramite le credenziali assegnate. È previsto che username e password siano unici per ciascun software/servizio che accede a queste API. Per esempio, una società che ha due siti dovrà avere complessivamente 2 credenziali distinte.

A ciascuna credenziale sono associati diritti di accesso diversi. Inoltre, questa differenziazione permettere di tracciare nei **log** le operazioni effettuate, dividendole proprio in base all'accesso.

### Autenticazione

L'accesso alle funzioni elencate è possibile solamente tramite autenticazione di tipo HTTP Basic Authentication. La Basic Authentication consiste in un un header standard HTTP con username e password codificate in base64:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Username e password sono codificati in questo formato:

```
username:password
```

La codifica base64 è reversibile, quindi le credenziali non sono crittografate o protette in alcun modo. Ne consegue che è necessario che la comunicazione sia protetta dagli attacchi di tipo *man in the middle*. È una delle tecniche più semplici per proteggere le risorse REST perché non necessita di cookie, identificatori di sessione o pagine di login.

### Base-URL per gli endpoint

I *base\_url* per quality e produzione sono:

Quality: **identitymanager-test4.intranet.giunti.it:81/chimera/api/05/**

Produzione: **identitymanager-cas4.intranet.giunti.it:81/chimera/api/05/**

## Convenzioni

### Errori

Nel caso di errori, la response HTTP sarà un documento JSON con questi contenuti:

```
{
  "error": {
    "message": "string - messaggio errore",
    "status": "int - http status code"
  }
}
```

UNAUTHORIZED - 401  
 NOT FOUND - 404  
 CONFLICT - 409  
 INTERNAL ERROR - 418  
 UNPROCESSABLE ENTITY - 422

## Identity e anagrafiche dei servizi federati

L'Identity è l'oggetto che rappresenta l'anagrafica dell'Identity Manager. È caratterizzato da una chiave unica **identityUid** utile a riconciliare le anagrafiche distribuite sui servizi federati.

Ciascun servizio federato (Federation) dovrà memorizzare lo **identityUid** abbinato a ciascuna anagrafica locale. Quindi ogni utente del servizio avrà un riferimento al suo **identityUid** dell'Identity Manager.

## Eliminazione di Identity (o ritiro del consenso)

Nel caso in cui sia necessaria la rimozione dei dati di una Identity (che dipenda o meno dal ritiro totale del consenso GDPR), l'Identity Manager non effettuerà una cancellazione fisica del record. Piuttosto, provvederà a rimuovere tutti i valori dai campi (*null*), tranne lo **identityUid**. Ciò è necessario per poter propagare l'informazione ai servizi federati.

## Sostituzione di Identity

Alcuni servizi federati (Federation) hanno la necessità di unire anagrafiche (Identity) che per vari motivi sono duplicazioni. Questo porta, di fatto, a una variazione della chiave **identityUid** che quindi non è mai un valore immutabile. Operativamente, l'Identity Manager effettua la sostituzione impostando come *null* le proprietà dell'oggetto Identity ridondante (è una delete logica). Inoltre, scrive nel suo campo **replacedByUid** lo uid della Identity che lo sostituirà. Di fatto, **replacedByUid** è un puntatore verso l'oggetto valido da uno non più valido.

Nel momento in cui un servizio federato riceve da Identity Manager un oggetto con **replacedByUid** valorizzato deve sostituire lo **identityUid** salvato in locale con il nuovo valore. Da questo momento in poi ogni operazione andrà effettuata facendo riferimento solo al nuovo **identityUid**.

## Guida all'implementazione

### Casi d'uso

Questo paragrafo consiste in un elenco di situazioni-tipo nelle quali il servizio federato effettua chiamate alle API.

Questi casi fungono da linee guida per la federazione dei servizi. Un utilizzo indiscriminato delle funzioni potrebbe danneggiare la base dati condivisa.

- **(A) Passaggi generali per determinare lo **identityUid** aggiornato**
  - Durante l'autenticazione, un servizio federato riceve da **get\_identity\_by\_email** oppure **get\_identity\_by\_social\_id** un oggetto **Id entityHistory** (vedi sezione Oggetti in Response)
  - Se lo **identityUid** contenuto in **IdentityHistory** è già noto al servizio federato, non ci sono altre operazioni da svolgere, lo **identityUid** è da considerare corretto e utilizzabile.
  - Se lo **identityUid** non è noto, verificare che nell'array **replacedIdentityUids** di **IdentityHistory** non ci siano **identityUid** noti, questo è l'elenco degli uid non più validi che sono stati sostituiti
    - Se è così, sostituire localmente il vecchio **identityUid** con il nuovo

- Se lo **identityUid** non è comunque conosciuto, l'utente andrà considerato come nuovo per il servizio federato (proporre registrazione?)
- **Un nuovo utente si registra e crea la propria anagrafica:**
  - 1° passo: il servizio federato chiama **validate\_identity\_data** (con identityUid vuoto) per validare i dati inseriti e segnala gli errori all'utente
  - 2° passo: quando la verifica ha successo, il servizio chiama **add\_identity** e ottiene così l'oggetto **Identity**, in modo da poter abbinare in locale l'anagrafica al **identityUid**
  - 3° passo: il servizio aggiorna le informazioni di consenso GDPR chiamando **update\_identity\_consent**
- **Un utente effettua login classico (email+password) su servizio federato:**
  - 1° passo: il servizio federato ottiene lo **identityUid** chiamando **get\_identity\_by\_email**
  - 2° passo: lo **identityUid** è creato/verificato/allineato tramite la procedura (A)
  - 3° passo: il sistema federato deve allineare i dati utente in locale con quelli dell'identity manager con **update\_identity** e **update\_identity\_consent**
- **Un utente effettua login social su servizio federato:**
  - 1° passo: il servizio federato ottiene lo **identityUid** chiamando **get\_identity\_by\_social\_id**
  - 2° passo: se non è stato trovato un **identityUid** in base al **socialId**, allora questo va cercato con l'email tramite **find\_identity\_by\_email**, l'email è necessaria per riconciliare i social login quando non è possibile con un **identityUid**
  - 3° passo: lo **identityUid** è creato/verificato/allineato tramite la procedura (A)
  - 4° passo: chiamare **add\_provider\_account** per creare l'abbinamento tra login social e **Identity**
- **Un utente modifica la propria anagrafica:**
  - 1° passo: prima di mostrare il form di modifica, i dati dell'anagrafica locali e lo **identityUid** devono essere allineati chiamando **get\_identity**
  - 2° passo: al submit, il servizio federato chiama **validate\_identity\_data** (inclusendo identityUid) per validare i dati inseriti dall'utente
  - 3° passo: se la verifica ha successo, i dati possono essere effettivamente aggiornati chiamando **update\_identity** e, se necessario, anche **update\_identity\_consent**
- **Un utente modifica il consenso GDPR:**
  - Il servizio federato chiama **update\_identity\_consent** (leggere le note riguardo a **consentrange**)
- **Un utente richiede la rimozione del proprio account dal servizio federato:**
  - 1° passo: se l'utente cambia i consensi, il servizio federato deve chiamare **update\_identity\_consent**
  - 2° passo: il servizio rimuove completamente i dati dell'anagrafica dal proprio DB interno
  - 3° passo: è necessario poi chiamare anche la funzione **delete\_identity** che effettua una rimozione dei dati ma non dello **identityUid**, in modo da poter propagare la rimozione su tutti i servizi federati (chiaramente è una operazione distruttiva su tutta la federazione)
- **Allineamento dati ogni X minuti:** operazione sincrona di allineamento dati dall'Identity Manager verso i servizi federati
  - 1° passo: il servizio federato chiama **find\_changed\_identities** ogni X minuti
  - 2° passo: il servizio scorre la lista restituita cercando se vi sono **identityUid** a lui noti. Per ciascuna identity presente in locale:
    - se **changeType="update"** sostituisce i dati locali con quelli dell'identity
    - se **changeType="replace"** significa che lo **identityUid** del record corrente è stato sostituito dal valore in **replacedByUid**
      - se il **replaceByUid** non è presente nel DB locale allora il servizio federato sostituisce i vecchi dati locali compreso il **vecchio identityUid** (è una operazione di update)
      - se il **replacedByUid** è già presente nel DB locale, allora l'anagrafica con il **vecchio identityUid** deve essere eliminata, in modo da far sopravvivere solo quella con il nuovo, che presumibilmente avrà anche i dati più aggiornati
    - se **changeType="delete"** elimina completamente l'anagrafica locale
  - N.B. Per i **identityUid** elencati ma non noti non è necessario effettuare alcuna operazione.
- **C'è una riconciliazione (replace) tra anagrafiche effettuata in locale:**
  - 1° passo: il servizio federato chiama **replace\_identity** passando sia lo **identityUid** ridondante che quello finale
  - 2° passo: se la funzione precedente ha successo, va eliminata completamente l'anagrafica ridondante dal DB locale

## Note

### Insiemi di dati sincronizzati e non

Ciascun servizio federato può scegliere di tenere allineato con l'Identity Manager anche solo un sottoinsieme dei dati che possiede. Per esempio solo email, nome e cognome. In questo caso, il servizio federato **NON deve né scaricare né caricare sull'Identity Manager alcun dato che non appartenga a questo sottoinsieme.**

Questa specifica è fondamentale per evitare che servizi federati (male) propaghino dati che loro stessi non mantengono aggiornati. Si tratta di una perdita di dati irrimediabile.

- Esempio: un servizio che aggiorna **da** Identity Manager solamente le informazioni di Nome e Cognome non deve assolutamente caricare **verso** l'Identity Manager altre informazioni che non siano Nome e Cognome

## Aggiornamento del consenso GDPR

La chiamata ad **update\_identity\_consent** con il parametro **range = "ALL"** propagherà l'update ai consensi GDPR **di tutte le società**.

## Precisazioni sull'uso della funzione di allineamento **find\_changed\_identities**

La funzione **find\_changed\_identities** deve essere eseguita ogni X minuti, in modo da ottenere dall'Identity Manager l'elenco dei dati modificati. Per chiamarla, è necessario fornire un timestamp numerico che indichi l'istante temporale del passato (**start\_timestamp**) a partire da cui ricevere gli aggiornamenti. Il valore è di tipo intero long. La response a questa chiamata conterrà un nuovo timestamp **current\_timestamp** che potrà essere utilizzato come istante temporale iniziale per la chiamata successiva (dopo i canonici X minuti).

In sintesi, è consigliato memorizzare il **current\_timestamp** restituito da ogni chiamata per utilizzarlo come **start\_timestamp** di quella successiva, assicurandosi così che non esistano intervalli temporali scoperti

Considerare che la chiamata a **find\_changed\_identities** da parte di un servizio federato XYZ non riporta le identity modificate dal servizio XYZ stesso, per evitare loop.

## Oggetti in response

### Identity

È l'oggetto che contiene l'anagrafica identificato dallo **identityUid**. Quando un oggetto Identity è stato unito a un altro (replace) tutte le sue proprietà sono null, come per una delete logica, tranne **replacedByUid** che indica quale nuova Identity rappresenta realmente l'anagrafica cercata.

```

{
  "identityUid": "string - è lo identity_uid più recente",
  "replacedByUid": "string - è lo uid della anagrafica che ha
sostituito questa, quando è popolato tutte le altre proprietà sono
vuote",
  "changeTime": "<datetime>",
  "email": "email",
  "lastName": "string",
  "firstName": "string",
  "sex": "m|f",
  "birthDate": "<date>",
  "addressStreet": "string",
  "addressZip": "string",
  "addressProvinceId": "string - codice provincia",
  "addressTown": "string - località",
  "telephone": "string",
  "codiceFiscale": "string",
  "partitaIva": "string",
  "interest": "string",
  "job": "string",
  "school": "string",
  "newsletters": [ "newsletter1", "newsletter2", ...],
  "consent": [
    {
      "range": "<consentrange>",
      "tos": "true|false",
      "marketing": "true|false",
      "profiling": "true|false",
      "tosDate": "<date>",
      "marketingDate": "<date>",
    },
    ...
  ]
}

```

## IdentityHistory

L'oggetto restituito contiene lo **identityUid** di una data **Identity** e i suoi vecchi **identityUid**. Queste informazioni facilitano la riconciliazione dell'anagrafica locale, vista la possibile variazione dello **identityUid**

```
{
  "identityUid": "string - identificativo dell'utente",
  "replacedIdentityUids": [
    "oldUid1",
    "oldUid2",
    ...
  ]
}
```

## Validation

Oggetto che contiene il risultato della validazione con l'elenco di messaggi.

```
{
  "success": "true|false",
  "assignedIdentityUid": "string - eventuale uid assegnato",
  "messages": [
    "property1": "string - messaggio d'errore di validazione",
    "property2": "string - messaggio d'errore di validazione",
    ...
  ]
}
```

## ProviderAccount

Informazioni di social login collegate a un dato **Identity**

```
{
  "identityUid": "string - identificativo dell'utente",
  "socialId": "<socialId>"
}
```

## Funzioni: gestione delle Identity

### POST: authenticate

Restituisce un **Identity** se le credenziali fornite sono corrette.

Endpoint POST: [http://base\\_url/authenticate](http://base_url/authenticate)

#### REQUEST:

```
{
  "email": "string - email utente (oppure username per compatibilità)",
  "password": "string - password"
}
```

#### RESPONSE:

Restituisce un oggetto **IdentityHistory**, contenente lo **identityUid** e i vecchi **identityUid**. Queste informazioni facilitano la riconciliazione dell'anagrafica locale, vista la possibile variazione dello **identityUid**

#### GET: get\_identity

Permette di ottenere un **Identity** in base all'**identityUid**.

Endpoint POST: `http://base_url/get_identity/{identityUid}`

**{identityUid}** string - identificativo dell'utente

#### RESPONSE:

Restituisce un oggetto **Identity**.

Se i dati non corrispondono ad alcuna **Identity**, la response conterrà un errore.

#### GET: find\_identity\_uid\_by\_email

Restituisce un **identityUid** in base all'email fornita.

Endpoint POST: `http://base_url/find_identity_uid_by_email/{email}`

**{email}** string - email utente (oppure username per compatibilità giuntiscuola)

#### RESPONSE:

Restituisce un oggetto **IdentityHistory** che contiene lo **identityUid** attuale e i vecchi **identityUid**. Queste informazioni facilitano la riconciliazione dell'anagrafica locale, vista la possibile variazione dello **identityUid**

#### GET: find\_identity\_uid\_by\_social\_id

Permette di ottenere lo **identityUid** abbinato al **socialId** fornito.

Endpoint POST: `http://base_url/find_identity_uid_by_social_id/{socialId}`

**{socialId}** string - <socialId>

#### RESPONSE:

Restituisce un oggetto **IdentityHistory** che contiene lo **identityUid** attuale e i vecchi **identityUid**. Queste informazioni facilitano la riconciliazione dell'anagrafica locale, vista la possibile variazione dello **identityUid**



## POST: validate\_updating\_identity

Necessita dei diritti **canUpdate**

Permette di validare i dati utente prima di una eventuale chiamata a **update\_identity**

Endpoint POST: `http://base_url/validate_updating_identity`

REQUEST:

Vige la logica del vuoto per pieno:

```
{
  "identityUid": "string(32) - rappresenta l'identificativo unico
dell'identity",
  "email": "string(64) - è necessaria l'unicità",
  "password": "string(64) - sarà memorizzata internamente come hash
md5",
  "lastName": "string(64) - cognome o ragione sociale",
  "firstName": "string(32) - nome",
  "sex": "string(1) - può assumere solo i valori <m|f>",
  "birthDate": "<date> - data di nascita",
  "addressStreet": "string(64) - indirizzo",
  "addressZip": "string(16) - cap",
  "addressProvinceId": "string(2) - sigla provincia",
  "addressTown": "string(64) - località",
  "telephone": "string(32) - telefono",
  "codiceFiscale": "string(16)",
  "partitaIva": "string(16)",
  "interest": "string(256) - (?)",
  "job": "string(256) - (?)",
  "school": "string(256) - (?)",
  "newsletters": [ "newsletter1", "newsletter2", ...]
}
```

RESPONSE:

Restituisce un oggetto **Validation**.

## POST: update\_identity

Necessita dei diritti **canUpdate**

Permette di aggiornare i dati di una **Identity**.

Endpoint POST: `http://base_url/update_identity`

REQUEST:

```
{
  "identityUid": "string(32) - rappresenta l'identificativo unico
dell'identity",
  "email": "string(64) - è necessaria l'unicità",
  "password": "string(64) - sarà memorizzata internamente come hash
md5",
  "lastName": "string(64) - cognome o ragione sociale",
  "firstName": "string(32) - nome",
  "sex": "string(1) - può assumere solo i valori <m|f>",
  "birthDate": "<date> - data di nascita",
  "addressStreet": "string(64) - indirizzo",
  "addressZip": "string(16) - cap",
  "addressProvinceId": "string(2) - sigla provincia",
  "addressTown": "string(64) - località",
  "telephone": "string(32) - telefono",
  "codiceFiscale": "string(16)",
  "partitaIva": "string(16)",
  "interest": "string(256) - (?)",
  "job": "string(256) - (?)",
  "school": "string(256) - (?)",
  "newsletters": [ "newsletter1", "newsletter2", ...]
}
```

Sono gli stessi valori di **validate\_identity\_data**, ma con **identityUid** obbligatorio.

#### RESPONSE:

Restituisce un oggetto **Validation**.

#### POST: update\_identity\_consent

[Necessita dei diritti canUpdate](#)

È obbligatorio chiamare questa funzione dopo **add\_identity**, oltre a quando c'è un cambiamento nel consenso da parte dal cliente.

Impostando "range" con il valore "ALL" i valori del consenso associati a tutte le società vengono modificati con una sola chiamata. È equivalente a chiamare questa funzione più volte, passando in ciascuna come range "GE", "GS", "GAP"...

Endpoint POST: [http://base\\_url/update\\_identity\\_consent](http://base_url/update_identity_consent)

#### REQUEST:

```
{
  "identityUid": "string - rappresenta l'identificativo unico
dell'identity (obbligatorio)",
  "range": "<consentrage> - valore che indica la società per cui si
stanno modificando i consensi",
  "tos": "true|false - consenso termini del servizio",
  "marketing": "true|false - consenso marketing",
  "profiling": "true|false - consenso profilazione",
  "tosDate": "<date> - obbligatorio se tos è true",
  "marketingDate": "<date> - obbligatorio se marketing è true"
}
```

#### RESPONSE:

Restituisce un oggetto **Identity**.

#### POST: **validate\_new\_identity**

Necessita dei diritti **canUpdate**

Permette di validare i dati utente prima di una eventuale chiamata a **create\_identity**

Endpoint POST: [http://base\\_url/validate\\_new\\_identity](http://base_url/validate_new_identity)

#### REQUEST:

Vige la logica del vuoto per pieno:

```
{
  "email": "string(64) - è necessaria l'unicità",
  "password": "string(64) - sarà memorizzata internamente come hash
md5",
  "lastName": "string(64) - cognome o ragione sociale",
  "firstName": "string(32) - nome",
  "sex": "string(1) - può assumere solo i valori <m|f>",
  "birthDate": "<date> - data di nascita",
  "addressStreet": "string(64) - indirizzo",
  "addressZip": "string(16) - cap",
  "addressProvinceId": "string(2) - sigla provincia",
  "addressTown": "string(64) - località",
  "telephone": "string(32) - telefono",
  "codiceFiscale": "string(16)",
  "partitaIva": "string(16)",
  "interest": "string(256) - (?)",
  "job": "string(256) - (?)",
  "school": "string(256) - (?)",
  "newsletters": [ "newsletter1", "newsletter2", ...]
}
```

#### RESPONSE:

Restituisce un oggetto **Validation**.

#### POST: add\_identity

Necessita dei diritti **canUpdate**

Crea una nuova **Identity** e la restituisce.

Endpoint POST: [http://base\\_url/add\\_identity](http://base_url/add_identity)

#### REQUEST:

```
{
  "email": "string(64) - è necessaria l'unicità",
  "password": "string(64) - sarà memorizzata internamente come hash md5",
  "lastName": "string(64) - cognome o ragione sociale",
  "firstName": "string(32) - nome",
  "sex": "string(1) - può assumere solo i valori <m|f>",
  "birthDate": "<date> - data di nascita",
  "addressStreet": "string(64) - indirizzo",
  "addressZip": "string(16) - cap",
  "addressProvinceId": "string(2) - sigla provincia",
  "addressTown": "string(64) - località",
  "telephone": "string(32) - telefono",
  "codiceFiscale": "string(16)",
  "partitaIva": "string(16)",
  "interest": "string(256) - (?)",
  "job": "string(256) - (?)",
  "school": "string(256) - (?)",
  "newsletters": [ "newsletter1", "newsletter2", ...]
}
```

Sono gli stessi valori di **validate\_identity\_data**, ma senza **identityUid**.

#### RESPONSE:

Restituisce un oggetto **Validation**.

#### POST: **replace\_identity**

[Necessita dei diritti canReplace](#)

Permette di marcare una identity come doppione di un'altra ed effettuare la deduplica.

Endpoint POST: [http://base\\_url/replace\\_identity](http://base_url/replace_identity)

#### REQUEST:

```
{
  "redundantIdentityUid": "string - identificativo unico dell'identity da eliminare perché ridondante",
  "finalIdentityUid": "string - identificativo unico dell'identity finale (risultato dell'accorpamento)"
}
```

#### RESPONSE:

Restituisce un oggetto **Identity**.

### POST: delete\_identity

Necessita dei diritti **canDelete**

Questa funzione rimuove completamente i dati nell'identity, tranne per lo identity\_uid e la data di aggiornamento. Ciò è necessario per tenere traccia dell'operazione e propagare la rimozione ai siti federati.

Endpoint POST: `http://base_url/delete_identity`

REQUEST:

```
{
  "identityUid": "string - identificativo dell'utente"
}
```

RESPONSE:

Restituisce un oggetto **Validation**.

## Funzioni: gestione del social login

### POST: add\_provider\_account

Necessita dei diritti **canUpdate**

Permette di associare l'account di un provider esterno (social login) a una identity esistente. L'abbinamento consente all'utente di accedere anche tramite social login. La funzione ha successo solamente se all'identity non era già associato un social\_id relativo allo stesso provider.

Endpoint POST: `http://base_url/add_provider_account`

REQUEST:

```
{
  "identityUid": "string - identificativo dell'utente",
  "socialId": "<socialId> - identificativo dell'account social nel
formato nomeServizio#9999999999"
}
```

RESPONSE:

Restituisce un oggetto **ProviderAccount**.

## POST: delete\_provider\_account

Necessita dei diritti [canUpdate](#)

Permette di rimuovere l'abbinamento tra un Identity e il socialId di un provider esterno.

Endpoint POST: `http://base_url/delete_provider_account`

REQUEST:

```
{
  "identityUid": "string - identificativo dell'utente",
  "socialId": "<socialId> - identificativo dell'account social nel
formato nomeServizio#9999999999"
}
```

RESPONSE:

Restituisce un oggetto **Validation**.

## GET: find\_provider\_accounts

Permette di ottenere gli account dei provider social abbinati a una identity.

Endpoint GET: `http://base_url/find_provider_accounts/{identityUid}`

**{identityUid}** string - identificativo dell'utente

RESPONSE:

```
{
  "providerAccounts": [
    ProviderAccount, ProviderAccount, ...
  ]
}
```

## Funzioni: federazione

### GET: find\_changed\_identities

Restituisce l'elenco delle **Identity** nuove, modificate o rimosse.

Endpoint GET: `http://base_url/find_changed_identities/{startTimestamp}`

**{startTimestamp}** <timestamp> - istante temporale da cui elencare le novità

*La richiesta con un timestamp trascorso da più di una settimana restituirà un errore.*

#### RESPONSE:

Restituisce un oggetto che contiene un array di **Identity** ed il timestamp relativo all'istante attuale. Eseguendo il prossimo *find\_changed\_identities* con questo nuovo timestamp i nuovi risultati non si sovrapporranno a quelli già ottenuti senza rischio di ignorare dei gap temporali.

```
{
  "currentTimestamp": "<timestamp>",
  "identities": [
    Identity, Identity ...
  ]
}
```

#### GET: find\_federations

Restituisce l'elenco dei servizi federati con identity manager, con i loro identificativi.

Endpoint GET: [http://base\\_url/find\\_federations](http://base_url/find_federations)

#### RESPONSE:

```
{
  federations: [
    {
      "name": "string - nome servizio",
      "federationUid": "string - identificativo servizio"
    },
    ...
  ]
}
```

## Formato dati e valori di lookup

### Charset

I contenuti delle request e response HTTP sono previsti esclusivamente con charset UTF-8.

### <date>

I dati indicati come 'date' sono nel formato **yyyy-MM-dd**



### <datetime>

I dati indicati come 'datetime' sono nel formato **yyyy-MM-dd'T'HH:mm:ss.SSS'Z'**

Il formato è leggermente diverso rispetto alle API precedenti e segue lo standard ISO 8601.

Esempio: 20170801T220101+0200 (lo url escape di '+' è '%2B')

### <timestamp>

I dati indicati come 'timestamp' sono interi long che identificano un istante temporale. Le funzioni restituiscono questo valore come stringa per evitare l'arrotondamento.

E' importante che un 'timestamp' non sia gestito sul client come numero floating point, perché potrebbe essere soggetto ad arrotondamento.

### <socialId>

I dati di tipo 'socialId' sono stringhe che rappresentano l'identificativo dell'account social nel formato **nomeServizio#9999999999**

Nelle richieste di tipo GET è necessario effettuare un URL escape di '#' utilizzando '%23' al suo posto:

FacebookProfile#0000002817 FacebookProfile%230000002817

Attualmente questi sono i prefissi permessi nel valore **socialId**:

| prefisso socialId      | servizio     |
|------------------------|--------------|
| FacebookProfile        | Facebook     |
| Google2Profile         | Google+      |
| TwitterProfile         | Twitter      |
| CasOAuthWrapperProfile | CentroStella |

### <consentrange>

I valori elencati servono a definire l'area per cui si stanno assegnando le informazioni di consenso secondo il GDPR.

| consentrange | descrizione  |
|--------------|--|
| ALL          | estende le informazioni di consenso a tutte le società |
| GE           | società Giunti Editore                                 |
| GS           | società Giunti Scuola                                  |
| GAP          | società Giunti al Punto                                |
| GEDU         | società Giunti EDU                                     |