



**COMUNE DI GENOVA**  
**DIREZIONE SISTEMI INFORMATIVI**  
**SETTORE PROGRAMMAZIONE E REALIZZAZIONE**

**REQUISITI DI INTEROPERABILITÀ**

## INDICE

1	REQUISITI DI INTEROPERABILITÀ .....	3
1.1	Introduzione.....	3
1.2	Convenzioni utilizzate .....	4
2	ELENCO DEI REQUISITI .....	5
2.1	Requisito #1: Disponibilità di API.....	5
2.2	Requisito #2: Progettazione delle API REST.....	5
2.3	Requisito #3: Documentazione delle API REST.....	5
2.4	Requisito #4: Requisiti obbligatori delle API REST .....	5
2.4.1	Requisito #4.1: Formato dei dati scambiati .....	6
2.4.2	Requisito #4.2: Definizione e naming delle interfacce .....	7
2.4.3	Requisito #4.3: Performance e Robustezza .....	8
2.5	Requisito #5: Requisiti di Privacy e di Sicurezza .....	9
2.5.1	Requisito #5.1: Privacy by Design e Privacy by Default.....	9
2.5.2	Requisito #5.2: Comunicazione dati personali solo su richieste esplicite	9
2.5.3	Requisito #5.3: Veicolazione dei dati tramite protocollo HTTPS.....	9
2.5.4	Requisito #5.4: Accesso circoscritto ai soli dati necessari .....	10
3	PATH DELLE INTERFACCE .....	14
3.1	Caratteristiche generali.....	14

---

# 1 REQUISITI DI INTEROPERABILITÀ

## 1.1 Introduzione

In conformità con le linee guida AGID per l'Informatica nella Pubblica Amministrazione il Comune di Genova considera la tematica dell'interoperabilità un elemento fondamentale della propria architettura di erogazione dei servizi. Il principale punto di riferimento è il Modello di Interoperabilità 2018, in breve ModI 2018, redatto da AGID e disponibile all'indirizzo

<https://docs.italia.it/italia/piano-triennale-ict/lg-modellointeroperabilita-docs/it/v2018.1/>

Ogni nuovo componente applicativo che viene introdotto nel S.I. comunale dovrà quindi produrre ed interagire con un elevato numero di microservizi realizzati da terzi.

Per la gestione dei microservizi il Comune di Genova si è inoltre dotato di una soluzione completa di API Management, con l'obiettivo di gestire l'intero ciclo di vita dei servizi erogati verso tutti i diversi attori del territorio e che:

1. Massimizza la consapevolezza generale dei servizi disponibili nel territorio favorendone la diffusione e il riuso
2. Fornisce gli strumenti per gestire l'intero ciclo di vita delle API sotto il completo controllo dell'organizzazione territoriale
3. Semplifica il compito dei progettisti nel dotare le API delle necessarie caratteristiche, funzionali e non, per rispettare i requisiti normativi vigenti (sicurezza, logging, QoS, ...)

I servizi che compongono l'ecosistema del Comune di Genova comprendono tutti i tipi di interazioni:

- Comune Genova <--> altre PA (A2A)
- Comune Genova <--> Imprese (A2B)
- Comune Genova <--> Cittadini (A2C)

In un contesto così variegato la scelta del Comune di Genova è quella di adottare l'approccio API First in quanto garantisce la massima circolazione e riuso delle interfacce dei servizi, che saranno quindi adottate dalle diverse applicazioni per un accesso regolamentato alle banche dati del territorio.

---

## 1.2 Convenzioni utilizzate

Nella definizione dei requisiti, delle specifiche e delle regole descritte sono utilizzate le parole chiave **DEVE**, **NON DEVE**, **OBBLIGATORIO**, **VIETATO**, **DOVREBBE**, **CONSIGLIATO**, **NON DOVREBBE**, **SCONSIGLIATO**, **PUÒ**, **OPZIONALE** che devono essere interpretate in particolare:

- **DEVE**, **OBBLIGATORIO**, **NECESSARIO** (**MUST**, **REQUIRED**, **SHALL**) significano che la definizione è un requisito assoluto, il requisito deve essere implementato, la consegna è inderogabile.
- **NON DEVE**, **VIETATO** (**MUST NOT**, **SHALL NOT**) significano che c'è proibizione assoluta di implementazione di un determinato elemento.
- **DOVREBBE**, **CONSIGLIATO** (**SHOULD**, **RECOMMENDED**) significa che in particolari circostanze possono esistere validi motivi per ignorare un requisito, non implementare una specifica, derogare alla consegna, ma che occorre esaminare e valutare con attenzione le implicazioni correlate alla scelta.
- **NON DOVREBBE**, **SCONSIGLIATO** (**SHOULD NOT**, **NOT RECOMMENDED**) significano che in particolari circostanze possono esistere validi motivi per cui un elemento è accettabile o persino utile, ma, prima di implementarlo, le implicazioni correlate dovrebbero essere esaminate e valutate con attenzione.
- **PUÒ**, **OPZIONALE** (**MAY**, **OPTIONAL**) significano che un elemento è a implementazione facoltativa.

Le parole chiave nel testo sono segnalate in maiuscolo e neretto (es. "**DEVE**").

---

## 2 ELENCO DEI REQUISITI

### 2.1 Requisito #1: Disponibilità di API

Tutte le nuove implementazioni software e tutti gli interventi evolutivi di software esistenti introdotti nel S.I. comunale **devono** includere interfacce applicative (API) delle risorse gestite dal sistema.

Le interfacce applicative **devono** essere realizzate secondo il paradigma REST che propone una visione del Web incentrata sul concetto di risorsa.

Le interfacce applicative **possono** essere progettate secondo il paradigma SOAP solo nel contesto di servizi preesistenti per i quali sia necessario estendere un set di API SOAP già in uso. In tal caso, comunque, le nuove API SOAP **dovranno** essere conformi alle specifiche del Modello di Interoperabilità 2018 di AGID, che prevede l'aderenza al Basic Profile 2.0.

### 2.2 Requisito #2: Progettazione delle API REST

Per la specifica di tali interfacce, al fine di descrivere le informazioni sulle operazioni disponibili in una API e su come devono essere strutturati i dati della richiesta e della risposta, **deve** essere adottato lo standard OpenAPI 3 (<https://www.openapis.org>).

### 2.3 Requisito #3: Documentazione delle API REST

Le Web API **devono** essere corredate dalla documentazione che fornisca i seguenti dettagli tecnici:

- Il modello e il formato dei dati
- Le operazioni fornite dalla API
- Eventuali caratteristiche aggiuntive quali: sicurezza, qualità del servizio erogato, ecc.

Il requisito sulla documentazione delle Web API **deve** essere soddisfatto allegando una descrizione dell'API formulata mediante il linguaggio di descrizione OpenAPI 3.

### 2.4 Requisito #4: Requisiti obbligatori delle API REST

Le API **devono** soddisfare i successivi requisiti di dettaglio suddivisi nelle seguenti tematiche:

- Formato dei dati scambiati
  - Definizione e naming delle interfacce
  - Performance e Robustezza
  - Schemi di sicurezza per l'accesso
-

### 2.4.1 Requisito #4.1: Formato dei dati scambiati

#### *Formato JSON per i dati:*

La descrizione di dati strutturati **deve** essere nel formato JSON (<http://www.rfc-editor.org/rfc/rfc7159.txt>), adottando il media type "application/json".

#### *Naming Proprietà Oggetti JSON*

Le proprietà degli oggetti JSON **devono** essere denominate utilizzando uno solo a scelta tra i metodi "snake\_case" e "camelCase".

#### *Standard per la nomenclatura*

Per la denominazione delle proprietà, laddove possibile, si **dovrebbe** seguire la nomenclatura stabilita nelle "Linee Guida per la valorizzazione del Patrimonio informativo nazionale" contenente ontologie e vocabolari controllati (<https://github.com/italia/daf-ontologie-vocabolari-controllati>). Esempio di utilizzo di tali standard sono: lingue, nazioni, monete, ecc.

#### *Boolean not null*

Le proprietà booleane **non devono** ammettere il valore NULL.

#### *Formati Data e Ora*

**Devono** essere utilizzati i seguenti formati standard per Data ed Ora (RFC-3339 o UTC) e per gli http headers (RFC-7231).

#### *Tipi con attributo "Format"*

Si **deve** definire l'attributo "format" quando si usano i tipi Number ed Integer.

---

## 2.4.2 Requisito #4.2: Definizione e naming delle interfacce

### *REST Maturity Level 2*

La definizione delle interfacce delle API **deve** seguire le regole del Richardson Maturity Model livello 2, che include:

- Evitare le azioni e ragionare intorno alle risorse
- Evitare i verbi nelle URL
- Usare correttamente i metodi HTTP
- Usare gli status HTTP appropriati

### *Uso corretto dei metodi HTTP*

L'interfaccia di servizio REST **deve** utilizzare l'HTTP verb più adatto all'operazione come indicato in RFC 7231 (<https://tools.ietf.org/html/rfc7231#section-4.3>).

### *Uso corretto degli header HTTP*

Gli header **non devono** essere utilizzati per veicolare informazioni applicative ma solo per informazioni di contesto oppure per supportare funzionalità di protocollo (autenticazione, autorizzazione, ecc). Prima di usare un header si **deve** verificare se è già disponibile nel censimento IANA (<https://www.iana.org/assignments/message-headers/message-headers.xhtml>) ed eventualmente adeguarsi.

### *URL Path*

Si **dovrebbe** utilizzare parole separate da trattino "-" per i Path nelle URL.

### *Query Parameters*

Si **deve** utilizzare uno tra i metodi "snake\_case" e "camelCase" per i Query Parameters.

### *Abbreviazioni*

**Non si devono** utilizzare abbreviazioni o acronimi non universalmente conosciuti.

### *Uso dei nomi al plurale*

Le collezioni di risorse **devono** usare nomi al plurale.

### *Slash finali nei path*

Si **deve** evitare slash finali nei path indicati nel descrittore OpenAPI.

### *Link Headers*

**Non si devono** usare Link Headers RFC5988 se la response è in JSON.

---

### *Restituire URI assoluti*

Si **deve** usare URI assoluti nei risultati al fine di indicare chiaramente al client l'indirizzo delle risorse di destinazione.

### *Formato Errori JSON*

Si **deve** usare lo schema Problem JSON per le risposte di errore. Il payload restituito è di tipo Problem (RFC 7807) con media-type "application/problem+json".

### *HTTP Headers custom*

Si **deve** evitare l'uso di X-Headers (deprecati in RFC-6648). Eventuali header di nuova definizione:

- **Non devono** andare in conflitto con header pubblici esistenti
- **Non devono** iniziare per "X-"
- **Devono** indicare il nome dell'organizzazione che li ha assegnati

## **2.4.3 Requisito #4.3: Performance e Robustezza**

### *Rate Limiting*

I servizi erogati **devono** attuare politiche di rate limiting. Gli eventuali limiti raggiunti devono essere segnalati ai fruitori tramite lo status "HTTP 429 (too many requests)". Inoltre le risposte **devono** sempre contenere gli header di controllo X-RateLimit-\* e **devono** onorare l'header Retry-After, sia nella variante che espone il numero di secondi dopo cui riprovare, sia nella variante che espone la data in cui riprovare.

### *Sovraccarico del sistema o indisponibilità del servizio*

I servizi erogati **devono** esporre un piano di continuità operativa segnalando il sovraccarico del sistema o l'indisponibilità del servizio tramite lo status "HTTP 503 (service unavailable)". **Devono** onorare l'header Retry-After analogamente al punto precedente.

### *Caching*

Le API che supportano il caching **devono** documentare le varie limitazioni e modalità di utilizzo tramite gli header definiti in RFC-7234. Di default il caching è disabilitato tramite indicazione nell'header HTTP "Cache-Control: no-cache".

---



### *Paginazione con parametri standard*

Si **deve** supportare la paginazione delle collezioni tramite:

- Paginazione classica: offset e limit.
- Paginazione con cursore: pagine con "infinite scrolling".

## **2.5 Requisito #5: Requisiti di Privacy e di Sicurezza**

### **2.5.1 Requisito #5.1: Privacy by Design e Privacy by Default**

Le API esposte **devono** essere progettate seguendo i due principi di base di Privacy by Design e di Privacy by Default:

- *Privacy by Design*: La progettazione delle API che gestiscono dati personali **deve** essere condotta applicando i criteri di riservatezza che troveranno di fatto nell'applicazione che fornisce l'implementazione, ma senza affidarsi ad essa. Devono quindi essere identificati chiaramente i dati personali necessari allo svolgimento delle operazioni e definite le misure di protezione in termini dei livelli di accesso necessari.
- *Privacy by Default*: Le operazioni effettuate nelle API **devono** prevedere, come comportamento di default, il livello massimo di protezione e quindi, a fronte di richieste generiche o prive di autorizzazione, condividono l'insieme minimo di informazioni personali e possibilmente nessuna.

### **2.5.2 Requisito #5.2: Comunicazione dati personali solo su richieste esplicite**

In nessun caso **devono** essere presenti API che restituiscono dati personali fuori dal contesto di una esplicita richiesta dell'utente. I dati personali sono trattati da operazioni ben precise che possono essere attivate solo per richiesta esplicita da parte dell'utente o comunque da lui debitamente autorizzate.

### **2.5.3 Requisito #5.3: Veicolazione dei dati tramite protocollo HTTPS**

Le API che trasmettono dati personali **devono** utilizzare per le comunicazioni esclusivamente il protocollo HTTPS, la cui cifratura dei dati previene eventuali intercettazioni delle informazioni sul percorso di trasmissione.

---

#### 2.5.4 Requisito #5.4: Accesso circoscritto ai soli dati necessari

Le interfacce API **devono** consentire l'accesso mirato ai dati personali sulla base dello specifico utilizzo e quindi dell'operazione che deve essere effettuata.

**Devono** essere evitate le soluzioni che veicolano insieme più grandi di dati, non necessari all'esecuzione dell'operazione richiesta, o addirittura la restituzione di un intero profilo di dati personali.

Per ciascuna operazione eseguibile nel contesto della API **deve** essere definito l'insieme minimo di dati personali necessari al suo svolgimento.

Sulla base di ciascun requisito funzionale, **dovranno** essere diversificate le possibili richieste di dati personali in maniera adeguata al tipo di operazione da svolgere.

**Possano** essere utilizzate due differenti tecniche:

1. Associare i dati acceduti a risorse distinte per caso d'uso. Sulla base dei singoli casi d'uso, vengono individuati gli specifici insiemi di dati personali da condividere, si procede di conseguenza alla definizione di singole risorse/metodi per ciascuno dei casi. Ad esempio:
  - a. `/user-profile/{username}/indirizzo-residenza` risorsa per operare (lettura o scrittura) sull'indirizzo ove è domiciliato l'utente.
  - b. `/user-profile/{username}/dati-istruzione` risorse per operare (lettura o scrittura) sulle informazioni inerenti il titolo di studio dell'utente.
2. Parametrizzare la singola risorsa per selezionare i dati acceduti. La distinzione dei dati personali da condividere viene effettuata, a parità di risorsa, tramite specifici parametri da fornire con la richiesta. Ad esempio:
  - a. `/user-profile/{username}` risorsa che consente potenzialmente di operare sull'intero profilo utente ma che stabilisce precisamente cosa condividere in base ai seguenti parametri (tipicamente nella query string):
    - i. `indirizzo-residenza` : boolean
    - ii. `dati-istruzione` : boolean

L'impostazione a "true" di ciascun parametro corrisponde alla richiesta di uno specifico set di dati. Se entrambi i parametri sono "false", non saranno restituiti dati personali.

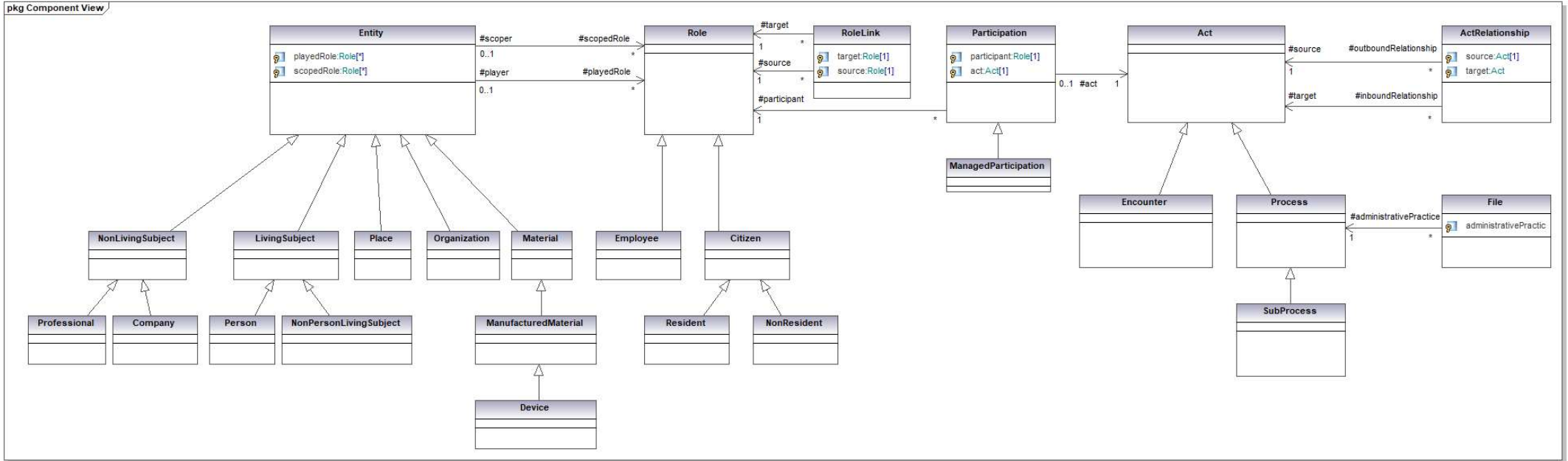
Se i set di dati personali sono associati a risorse/metodi distinti (caso 1), l'autorizzazione delle richieste in ingresso **può** essere effettuata indistintamente dal Resource Server che eroga il servizio o dall'API Gateway. Su quest'ultimo infatti è possibile configurare

opportune politiche di autorizzazione basate sulle risorse accedute e le relative credenziali e scope richiesti.

---

Se i set di dati personali sono distinti tramite parametri, nel contesto di una singola risorsa/metodo, l'autorizzazione **deve** essere effettuata esclusivamente dal Resource Server erogatore del servizio in quanto l'API Gateway non è in grado di effettuare valutazioni autorizzative basate sugli specifici contenuti delle richieste.

---



**Entity:**

E' un soggetto o un oggetto in grado di partecipare (Participation) con un certo ruolo (Role) ad un atto (Act). Comprende persone, imprese, luoghi ecc.

**Act:**

E' un'azione di interesse che viene svolta da una o più Entity in relazione ad un'altra o ad altre Entities. Un' istanza di un Act è un «record» di quell'azione (es. una specifica pratica associata ad un procedimento relativo ad un soggetto o oggetto).

**Role:**

E' una specializzazione di un Entity definita attraverso la relazione fra un «playing» Entity verso uno «scoping» Entity

**Participation:**

E' un'associazione fra un Role e un Act che rappresenta la funzione assunta dal Role nel contesto dell' Act. Un singolo Role può partecipare a multipli Act e un singolo Act può avere multipli participating Roles.

Ogni evento che corrisponde ad un'azione eseguita è un Act (una registrazione anagrafica, la presentazione di una istanza, il pagamento di un tributo, l'iscrizione ad un servizio ecc.)

La Participation definisce il contesto di un Act richiedente, ricorrente, titolare, firmatario, destinatario ecc. I partecipanti all'azione sono i Role (residente, contribuente, dipendente comunale, seconda casa, istituto scolastico ecc.)

I ruoli sono impersonati dalle Entity (persona, impresa, civico, edificio, strada, impianto termico ecc.)

Gli Act sono correlabili fra loro attraverso delle relazioni (ActRelationship) quali: composizione, preconditione, aggiornamento, dipendenza ecc....

I Role sono correlabili fra loro attraverso delle relazioni (RoleRelationship) quali: genitore di, coinquilino di, sposato/a con ecc...

---

### 3 PATH DELLE INTERFACCE

#### 3.1 Caratteristiche generali

Le URL e gli URI devono sempre essere espressi nel formato:

*https://baseUrl/nomeInterfaccia/[versione]/nome\_risorsa/{pathparameters}*

dove:

- Base Url comprende:
  - host (o dominio)
  - porta

nel formato: host:porta

- nome Interfaccia: è il nome assegnato al contesto della risorsa richiesta, ovvero dell'interfaccia API esposta
- versione (opzionale): contiene l'indicazione del numero di versione dell'interfaccia espresso nella forma: m.n.p (es. 1.0.0; 1.1.0; 2.0.1 ecc.)
- nome risorsa: è il nome della risorsa REST che viene coinvolta.

Tutte le stringhe presenti nella URL **devono** essere case-sensitive.

---



