Metodi HTTP nelle API: GET, POST, PUT, DELETE

Cosa sono i metodi HTTP nelle API?

I **metodi HTTP** (come GET, POST, PUT, DELETE) sono azioni standard che un client (browser, applicazione mobile, ecc.) usa per comunicare con un server. In un'API "RESTful", questi metodi sono usati per operazioni CRUD (Create, Read, Update, Delete) su risorse.

- **GET**: ottenere / leggere dati
- **POST**: creare una nuova risorsa o inviare dati
- **PUT**: aggiornare o sostituire una risorsa esistente
- **DELETE**: eliminare una risorsa

Questi metodi costituiscono le basi delle interazioni tra client e server.

Perché sono importanti nei design API?

Usare correttamente i metodi HTTP offre vari vantaggi:

1. **Standardizzazione e chiarezza**
   Ognuno sa che GET è per leggere dati, POST per creare, e così via → rende l'API più intuitiva.
2. **Gestione delle risorse**
   Ogni metodo ha un ruolo chiaro:
   - GET per ottenere informazioni
   - POST per aggiungere nuove risorse
   - PUT per modificare risorse esistenti
   - DELETE per cancellare risorse
3. **Scalabilità e manutenzione**
   Un uso corretto consente al sistema di supportare più richieste e di essere più facile da mantenere.

I metodi HTTP sono fondamentali nelle API REST. Usarli correttamente significa:

- leggere dati con GET,
- creare risorse con POST,
- aggiornare con PUT,
- cancellare con DELETE.

Se segui le buone pratiche (uso corretto dei codici HTTP, idempotenza, chiarezza), la tua API sarà più solida, mantenibile e facile da usare.

Come usare bene i metodi HTTP

Ecco una panoramica pratica + buone pratiche per ciascun metodo:

Metodo	Scopo principale	Buone pratiche	Esempio (richiesta)
GET	Recuperare dati	Usare query parametri per filtrare / paginare (es. ?page=2&limit=10); evitare dati sensibili nell'URL; l'operazione deve essere idempotente (ripetere la stessa richiesta non cambia nulla)	GET /users/123
POST	Creare dati / inviare dati	Usare per operazioni *non idempotenti* (cioè ripetere potrebbe creare duplicati); gestire errori (es. rispondere 400 Bad Request se i dati sono invalidi); restituire l'URL della risorsa creata tramite header Location	POST /users { "name": "Mario", "email": "mario@esempio.com" }
PUT	Aggiornare / sostituire risorsa	Deve essere idempotente (ripetere porta allo stesso risultato); usato per aggiornamenti completi (tutti i campi); rispondere con 200 OK o 204 No Content se non serve corpo di risposta	PUT /users/123 { "name": "Luigi", "email": "luigi@esempio.com" }
DELETE	Eliminare una risorsa	Se la risorsa non esiste, restituire 404 Not Found; ripetere la stessa DELETE non deve produrre errori (idempotenza); uso tipico: 204 No Content per indicare che è stata eliminata con successo	DELETE /users/123

Nota sull'idempotenza
Un'operazione è idempotente se chiamarla più volte produce lo stesso effetto del chiamarla una sola volta. Ad esempio, fare DELETE /users/123 due volte – se l'utente è già stato eliminato, ad esempio, la seconda volta non deve causare errori critici ma restituire un risultato coerente (es. "non trovato" o "già eliminato", eventualmente usando codici di stato).

Molto in breve, ci sono quattro categorie principali di codici di stato:

- Classe 2xx: 200, 201, 202, ... , 299: Status OK, ovvero la richiesta ha funzionato

- Classe 3xx: 300, 301, ..., 399: che riguarda i redirect da una request ad un altro host
- Classe 4xx: errore del client (chi fa la request)
- Classe 5xx: errore sul server (chi fornisce la response)

Errori comuni da evitare

- **Usare GET per inviare dati sensibili**
  I parametri passati nella URL sono visibili (es. password) → è più sicuro usare POST per questo tipo di dati.
- **Non rispettare l'idempotenza**
  Metodi come PUT e DELETE devono restare coerenti anche se invocati più volte — non devono creare effetti collaterali inattesi.
- **Codici di stato HTTP sbagliati**
  Non restituire "200 OK" sempre. Ad esempio:
  - 201 Created quando crei una risorsa con POST
  - 400 Bad Request se i dati inviati non sono validi
  - 404 Not Found se una risorsa che vuoi aggiornare o cancellare non esiste
- **Sovraccaricare il metodo POST**
  Evita di usare solo POST per tutte le operazioni (creazione, aggiornamento, cancellazione). Ogni metodo ha un suo scopo ben specifico.