**Cos'è REST?**

**REST** (Representational State Transfer) è uno stile architetturale per lo sviluppo di applicazioni distribuite, che si basa sul trasferimento di rappresentazioni di risorse.

REST è un modello di architettura che usa il protocollo HTTP come base per la comunicazione tra client e server.

Le risorse (entità) vengono identificate tramite URL e l'interazione con queste risorse avviene attraverso i metodi HTTP.  
  
Rest è una architettura utilizzata per costruire webservice che si basa sul protocollo HTTP ed in base al tipo di request fai un tipo di operazione.

**Caratteristiche principali di REST**

1. **Stateless**: Ogni richiesta HTTP contiene tutte le informazioni necessarie per essere compresa e processata dal server. Il server non mantiene stato tra una richiesta e l'altra.
2. **Client-Server**: L'architettura è separata in client e server. Il client invia richieste e il server le elabora, rispondendo con i dati richiesti o l'esito dell'operazione.
3. **Cacheable**: Le risposte devono essere etichettate come "cacheable" o "non cacheable" per evitare che client o intermediari usino dati obsoleti.
4. **Uniform Interface**: Il sistema deve avere una interfaccia uniforme che semplifica l'architettura. Questo implica che le operazioni siano standardizzate e facili da comprendere.
5. **Layered System**: L'architettura può essere composta da più livelli (proxy, server intermediari, ecc.) che separano il client dal server effettivo.
6. **Code on Demand (opzionale)**: I server possono inviare codice eseguibile (come JavaScript) che il client può eseguire, ma questa caratteristica non è obbligatoria e viene usata raramente.

**Risorse in REST**

In un'architettura REST, **tutto è una risorsa**. Le risorse rappresentano entità o dati che il client vuole manipolare (ad esempio, utenti, articoli, prodotti, etc.). Ogni risorsa è identificata da un **URI** (Uniform Resource Identifier) che agisce come un indirizzo univoco.

Ad esempio, un URL come:

|  |
| --- |
| https://api.example.com/utenti/123  ​ |

identifica un utente specifico (l'ID 123).

**Metodi HTTP in REST**

I metodi HTTP sono utilizzati per interagire con le risorse. I metodi più comuni in REST sono:

* **GET**: Recupera una risorsa o una rappresentazione di una risorsa.
* **POST**: Crea una nuova risorsa.
* **PUT**: Aggiorna o sostituisce una risorsa esistente.
* **DELETE**: Elimina una risorsa.
* **PATCH**: Modifica parzialmente una risorsa esistente.
* **HEAD**: Recupera solo l'intestazione della risposta, senza il corpo.
* **OPTIONS**: Ottiene le opzioni di comunicazione supportate dal server per la risorsa specificata.

**Principi RESTful**

Un'architettura **RESTful** segue questi principi fondamentali:

1. **Indipendenza dalla piattaforma**: REST non è legato a un particolare linguaggio di programmazione o sistema operativo. I client possono essere costruiti con qualsiasi tecnologia in grado di inviare richieste HTTP.
2. **Indipendenza dai dati**: Le risorse possono essere rappresentate in vari formati (come JSON, XML, HTML, ecc.) e il client non deve preoccuparsi del formato di rappresentazione.
3. **Interazione uniforme**: Le operazioni sulle risorse devono essere omogenee e usare sempre gli stessi metodi HTTP.

**Modelli di comunicazione in REST**

1. **Request-Response**: Il client invia una richiesta HTTP e il server risponde con uno status e un corpo della risposta. La risposta contiene i dati richiesti o l'esito dell'operazione.
2. **Client-Server**: Il client invia richieste al server, ma non dipende dal server. Il server, a sua volta, risponde senza conoscere lo stato del client.

**Codici di stato HTTP**

I codici di stato HTTP sono un altro aspetto importante di REST, che informano il client riguardo all'esito di una richiesta:

* **1xx** (Informational): La richiesta è stata ricevuta e il processo è in corso.
* **2xx** (Successo): La richiesta è stata completata con successo (ad esempio, 200 OK, 201 Created).
* **3xx** (Redirezione): È necessaria una nuova azione per completare la richiesta (ad esempio, 301 Moved Permanently).
* **4xx** (Errore client): La richiesta è malformata o contiene errori da parte del client (ad esempio, 400 Bad Request, 404 Not Found).
* **5xx** (Errore server): Il server ha incontrato un errore e non può completare la richiesta (ad esempio, 500 Internal Server Error).

**Operazioni su Risorse in REST**

Le operazioni CRUD (Create, Read, Update, Delete) sono mappate sui metodi HTTP come segue:

* **Create**: POST
* **Read**: GET
* **Update**: PUT o PATCH
* **Delete**: DELETE

Ad esempio:

* **GET /utenti**: Ottieni la lista di tutti gli utenti.
* **POST /utenti**: Crea un nuovo utente.
* **GET /utenti/{id}**: Ottieni i dettagli di un singolo utente.
* **PUT /utenti/{id}**: Aggiorna un utente esistente.
* **DELETE /utenti/{id}**: Elimina un utente.

**Comunicazione e Formati dei Dati**

In REST, la comunicazione avviene attraverso **formati di rappresentazione** dei dati, come JSON, XML, HTML, ecc. Il formato più comune per RESTful APIs è **JSON** per la sua leggerezza e facilità d'uso.

Esempio di risposta JSON per un'API RESTful:

|  |
| --- |
| {   "id": 123,   "nome": "Giovanni",   "email": "giovanni@example.com"  }  ​ |

**RESTful vs SOAP**

REST e SOAP sono due stili di architettura che si usano per realizzare web services. Le principali differenze tra i due sono:

* **REST** è più semplice e flessibile, basato su HTTP, e usa formati di dati leggeri come JSON e XML. È più adatto per applicazioni web moderne.
* **SOAP** è più complesso, basato su XML e richiede l'uso di specifiche più dettagliate (come WSDL per la descrizione dei servizi), ed è più adatto per applicazioni aziendali con requisiti di sicurezza e transazionalità elevati.

**Vantaggi di REST**

* **Scalabilità**: REST è progettato per sistemi scalabili, poiché non mantiene stato tra le richieste.
* **Semplicità**: La comunicazione tramite HTTP è semplice da implementare e ben supportata.
* **Flessibilità**: REST è indipendente dalla piattaforma, dai linguaggi e dai formati di rappresentazione.
* **Efficienza**: Utilizza metodi HTTP semplici e una rappresentazione dei dati leggera come JSON.

**Quando usare REST**

REST è particolarmente utile per applicazioni web, applicazioni mobili, servizi di microservizi e in generale quando c'è bisogno di una comunicazione semplice ed efficiente tra client e server. È un'ottima scelta quando si ha a che fare con risorse ben definite, che devono essere gestite tramite operazioni CRUD standard.

**Concetti di base**

1. **Cos'è REST e perché è importante?**
   * Quali problemi risolve rispetto ad altre architetture?
   * Quali sono le caratteristiche principali di un'architettura REST?
2. **Cos'è un web service RESTful?**
3. **Cosa significa che REST è stateless?**
   * Quali sono i vantaggi e gli svantaggi di un'architettura stateless?
   * Come si gestiscono le sessioni o l'autenticazione in REST?

**Risorse**

1. **Cos'è una risorsa in REST?**
   * Come vengono rappresentate le risorse?
   * Qual è il significato di un URI univoco per ogni risorsa?

**HTTP e CRUD**

1. **Quali sono i principali metodi HTTP utilizzati in REST?**
   * Qual è la differenza tra GET, POST, PUT, DELETE e PATCH?
   * Quando usare PUT invece di PATCH?
2. **Come si collegano i metodi HTTP alle operazioni CRUD?**
   * GET → Read
   * POST → Create
   * PUT → Update
   * DELETE → Delete
3. **Quali sono i principali codici di stato HTTP utilizzati in REST?**
   * Cosa rappresentano i codici di stato 200, 201, 204, 400, 404 e 500?
   * Qual è la differenza tra 200 (OK) e 204 (No Content)?

**Formati di dati e rappresentazione**

1. **Quali formati di dati possono essere utilizzati in REST?**
   * Qual è il formato più comune e perché?
   * Quali sono i vantaggi di utilizzare JSON rispetto a XML?
2. **Come si gestiscono le CORS (Cross-Origin Resource Sharing) in REST?**
   * Qual è il ruolo dell'header Access-Control-Allow-Origin?