# Appunti: API REST (04b-REST.txt)

## Introduzione alle API Web

- **Definizione:** Un'interfaccia HTTP che permette ad applicazioni remote di utilizzare i servizi di un'applicazione.
- **Utilizzo:**
  - Applicazioni automatiche che utilizzano i dati.
  - Applicazioni Web che interagiscono con l'utente per eseguire azioni sui dati.
- **Esempio:** Twitter API v1.1.

## REST: REpresentational State Transfer

- **Definizione:** Modello architetturale alla base del World Wide Web e delle applicazioni web "ben fatte".
- **Confronto con applicazioni non REST:**
  - **Non REST:** API che specifica funzioni, interfaccia indipendente dal protocollo di trasporto.
  - **REST:** Utilizzo di protocolli di trasporto (HTTP) e di naming (URI) per interfacce generiche, fortemente connesse all'ambiente d'uso.
- **Obiettivo:** API consistenti, predicibili, facili da capire e usare.

## Il modello CRUD

- **Definizione:** Pattern tipico per il trattamento dei dati.
- **Operazioni:**
  - **Create:** Inserimento di un nuovo record (es. creazione di un cliente).
  - **Read:** Accesso in lettura (individuale o contenitore).
  - **Update:** Modifica di un record esistente.
  - **Delete:** Rimozione di un record.

## Principi fondamentali di REST

1. **Risorse:** Ogni concetto rilevante dell'applicazione Web è una risorsa.
2. **URI:** Identificatore e selettore primario di ogni risorsa.
3. **Verbi HTTP:** Utilizzo dei verbi HTTP per esprimere le operazioni CRUD:
   - `PUT` : Creazione di un nuovo oggetto.
   - `GET` : Visualizzazione dello stato della risorsa.
   - `POST` : Cambio di stato della risorsa.
   - `DELETE` : Cancellazione di una risorsa.
4. **Rappresentazioni:** Rappresentazione parametrica dello stato interno della risorsa, personalizzabile tramite `Content-Type` .

## Esempi REST

- **Crea cliente (PUT):**
  - `PUT clients/1234 HTTP/1.1`
  - Body: Rappresentazione XML dell'oggetto da creare.
- **Aggiorna cliente (PUT):**
  - `PUT clients/1234 HTTP/1.1`
  - Body: Rappresentazione JSON dell'oggetto da sovrascrivere.

- Nota: `PUT` usato sia per creare che per sostituire.

## Individui e Collezioni

- **Individui:** Singole entità (es. un cliente, un esame).
- **Collezioni:** Insiemi di individui (es. tutti i clienti, tutti gli esami superati).
- **URI:** Forniti ad entrambi.
- **Operazioni CRUD:** Eseguibili su entrambi.
- **Rappresentazione vs. Risorsa:** Il corpo di richieste/risposte contiene una *rappresentazione* della risorsa, non la risorsa stessa.

## Gerarchie

- **Collezioni:** Possono contenere individui o altre collezioni.
- **URI gerarchici:** Consigliati per esplicitare le relazioni (es. `/clients/1234/orders/`).
- **Vantaggi:** API più leggibile, routing semplificato.

## Linee guida per gli URI in REST

- **Collezioni:** Plurali e con slash finale (es. `/customers/`).
- **Individui:** Singolari (es. `/customers/abc123`).
- **Distinzione:** Chiaramente distinguibili.

## Filtri e ricerca negli URI REST

- **Filtri:** Generano sottoinsiemi tramite regole.
- **Gerarchie:** Specificano filtri frequenti e rilevanti.
- **Query:** Utilizzata per filtri non gestiti dalla gerarchia (es. `/customers/?tel=0511234567`).

## Uso dei verbi HTTP in REST

| Verbo | Risorsa | Descrizione |
|---|---|---|
| `GET` | `/customers/` | Elencare tutti i clienti |
| `GET` | `/customers/abc123` | Accedere ai dati del cliente id=abc123 |
| `POST` | `/customers/` | Creare un nuovo cliente (il client *non* decide l'identificatore) |
| `PUT` | `/customers/abc123` | Creare un nuovo cliente (il client decide l'identificatore) |
| `PUT` | `/customers/abc123` | Modificare (tutti) i dati del cliente id=abc123 |
| `POST` | `/customers/abc123/` | Modificare *alcuni* dati del cliente id=abc123 |
| `DELETE` | `/customers/abc123` | Cancellare il cliente id=abc123 |

- **Importante:** Differenza tra `POST` e `PUT` per la creazione.

## Semantica del POST

- **RFC2616 (vecchia):** `POST` per creare un nuovo subordinato della risorsa identificata dalla Request-URI.
- **RFC7231 (aggiornata):** `POST` richiede che la risorsa elabori la rappresentazione secondo la sua semantica specifica.

- **In pratica:** `POST` utilizzabile in molte situazioni, con semantica locale, purché non sovrapposta ad altri verbi.

## Altri consigli e linee guida

- **Convenzioni di denominazione:** Coerenti e chiare negli URI.
- **Gerarchie:** Valutare i livelli necessari (chiarezza vs. carico).
- **Evitare:** API che rispecchiano la struttura interna del database.
- **Filtri e paginazione:** Fornire meccanismi tramite parametri nelle query.
- **Richieste asincrone:** Supportarle, restituire codice 202 (Accepted) e informazioni per accedere allo stato.

## Descrivere una RESTful API

- **RESTful:** API che utilizza i principi REST.
- **Documentazione:**
  - **End-point (URI/route):** Separando collezioni e singoli elementi.
  - **Metodi HTTP:** Cosa succede con `GET`, `PUT`, `POST`, `DELETE`, ecc.
  - **Rappresentazioni Input/Output:** Esempi fittizi, non necessariamente schemi formali.
  - **Condizioni di errore:** Messaggi restituiti.

## Conclusioni

- **REST:** Applicazione come ambiente con stato modificabile tramite comandi (metodi HTTP) applicati a risorse (URI) e visualizzati tramite rappresentazioni (Content-Type).
- **Pregi:** Sfrutta appieno le caratteristiche del web (caching, proxying, sicurezza, ecc.).
- **Semantic Web:** Possibilità di sfruttare tecniche del Semantic Web per funzionalità avanzate.