

- Più moderno per la creazione di web api usando principi HTTP
- Significato:
 - Trasferimento dello stato in maniera rappresentativa
 - Permette di creare sistemi distribuiti che possono comunicarsi in maniera diretta
 - Ogni risorsa viene rappresentata tramite URI (/baseUrl/{id}), id=Path parameter, specifichiamo la risorsa
 - Le risorse potrebbero cambiare nel tempo, in particolare con l'iterazione del client
 - Manipolato solamente attraverso GET, modificarla, POST/PUT/DELETE
 - Hanno degli identificatori
 - Risorsa = Qualunque cosa importante da referenziare
 - Espongono una interfaccia uniforme
 - Tutti gli altri servizi implementeranno lo stesso approccio
 - Imparo uno imparo tutti
 - Manipolate con la rappresentazione
 - I messaggi sono
 - Stateless, deve poter funzionare anche senza sapere i messaggi antecedenti
 - No sessioni
 - No cookies
 - No chiavi dell'URL
 - Autodescrivono, avere descrizione con i metadati senza riferimenti precedenti
 - Interfaccia uniforme + Stateless + AutoDescrittivi = Cacheable
 - Cache web, spazio di memoria tra il client ed il server
 - Necessario per un sistema a livelli
 - Riduce latenza e traffico di rete
 - Cache-control → Specifica per quanto il browser lo può tenere in cache
E con questo noi possiamo fare richieste get-if-not-matched
Cioè, faccio una get, e se il pacchetto di prima non è stato modificato, me lo dice così lo potrò richiedere successivamente
 - Una risorsa può essere correlato con link ad altre api → Client scopre possibile manipolazioni da solo
 - Si basa sui componenti URI & HTTP
 - Le iterazioni devono sempre essere client-server
- Rest vs Soap
 - Protocollo
 - Rest: HTTP
 - Soap: HTTP, TCP, SMTP
 - Formato:
 - Rest: XML, JSON
 - SOAP: XML-SOAP
 - Identificatori:
 - Rest: URI
 - SOAP: URI, WS-addressing
 - Documentazione:
 - Rest: Testuale, OpenAPI
OpenAPI rende il tutto automatico e di facile comprensione
 - SOAP: WSDL
 - Service discovery
 - No standard
 - UDDI
- Principi guida (Standard, semplificati):
 - Nomi: Ciò che vale la pena venire comunicato, chiamate **risorse**
Le operazioni: Cambiare, rimuovere, creare
 - Verbi: Operazioni tra le risorse
 - GET: Richiesta
 - Post: Procedure / Creare
 - PUT: Modificare / Creare
 - DELETE: Autoesplicativo
 - Head: Get senza body
 - Patch: Un post che non ci ha creduto abbastanza

Safe = non altera lo stato del server, aka read-only
Idempotente = Non importa quante volte lo eseguiamo, la risposta è sempre la stessa
[Ma io ho un déjà-vu]

 - PUT vs POST
 - Post per creare con chiave primaria generata automaticamente oppure "Fare un processo"
Usare post quando abbiamo un verbo
 - PUT per modificare
Può essere usato per creare quando vogliamo definire noi la chiave primaria di una tabella
 - Content type: la rappresentazione che vogliamo dare alla risorsa, ex. Json, Xml, Html
- Parti importanti:
 - Caching: migliora response-time
 - Statelessness and less communication: Più facile bilanciare il carico tra i server
 - Un software è meno specializzato, imparo uno imparo tutti
 - I naming si basano sui meccanismi del web, quindi sicuri (GET, ecc) ed unici
- Per la creazione:
 - Comprendere le risorse e dare nome
Scegliere un nome è difficile siccome:
 - Impone al cliente una sequenza di azioni per accedere alle risorse
 - La URL deve essere descrittiva
 - Non devono cambiare mai
 - Se cambiano, bisogna creare una nuova versione lasciando quella passata
 - Opachi, aka non dovrebbero far trapalare informazioni non necessarie (es .php)

Per trovarli:

 - Usare path variables (unimib/{matricola})
 - Dobbiamo evitare una gerarchia (unimib/{matricola1},{matricola2})

- Potremmo usare le query
- Definire i formati che accettiamo (Json, Xml, ecc)
 - Dovremmo usare sempre il formato quello ben conosciuto e standard
 - Se il client invia un qualcosa non corretto, si può optare per scartare la richiesta
- Definire le operazioni (Es GET)
- Quali codici di eccezione dobbiamo restituire
- REST può essere implementato come un CRUD
 - Create
 - Read
 - Update
 - Read

(Potrebbero non essere usati)

Ognuno ritorna un codice HTTP

Ed una struttura logica del genere è:

- Controller: gestisce iterazione ed endpoint
- Servizio: gestisce logica
- Repository: legge/ecc database
- Operazioni asincrone
 - Noi facciamo una richiesta client-server
 - Il server ci dice "Ricevuta la tua richiesta, la faremo in futuro, controlla l'avanzamento in questo url"
 - Il client ogni X richiede all'url la situazione della nostra richiesta
 - Richieste minimo 2 iterazioni per completare una iterazione
- Hypermedia control
 - Nella risposta JSON ci vengono ritornati dei link per poter velocizzare delle operazioni
 - Quindi se nella risposta abbiamo dei link, abbiamo hypermedia control
 - Questo dà un'informazione utile e maggiore scopribilità del sistema server
- Esempio:
 - Get `unimib.it/{matricola}` ⇒ **Endpoint**
 Ritorna informazioni dello studente e le varie operazioni che sono:
 - `cambiaPianoStudio`
 - `Administration`
 - `pianoStudio`
 Ed usando i vari metodi:
 - Get: `listaEsami`
 - `Unimib.it/{matricola}/pianoStudio/{code}`
 Informazione sull'esame
 - Post: `registraEsame`
 - Put: `modificaEsame`
 - Delete: `rimuoviEsame`