**Realizzazione di servizi RESTfull in ambito del progetto ASL**



**Studente** : Trionfetti Nico

**Classe** : 5A ITIS Marconi JESI

**Indirizzo** : Informatica

**Materia** : TPSIT/INFORMATICA/SISTEMI E RETI

**INDICE**

* **Progetto ASL 3**
* **Interfaccia Web Service 4**
* **Protocollo HTTP 5-6**
* **Servizio REST 7-12**
* **Web Server Embedded Grizzly 13**
* **JAVA e framework JERSEY 14-15**
* **JSON 16**
* **Schema del DB ASL 17-19**
* **Esempi di codifica dei servizi REST ASL 20-22**

**Progetto ASL**

Durante quest'anno scolastico, abbiamo svolto la realizzazione di un progetto "**Progetto ASL**", che consiste nella creazione di un sito WEB che permetta la gestione dell'alternanza scuola lavoro nella nostra scuola.

Questo sito permetterà di visualizzare un portfolio dell’alunno che riassume tutta la sua esperienza ASL nel periodo scolastico, le fasi di formazione e le fasi di stage.

Il progetto è stato svolto dalle classi del 5° anno del corso di informatica sotto la guida dei rispettivi insegnanti.

Sono state anche individuate delle figure responsabili alla formazione degli studenti riguardo le conoscenze indispensabili alla realizzazione del progetto, in questo caso la creazione e gestione di un database e la creazione e gestione di una pagina HTML con codice PHP.

Dopo che il progetto è stato esposto alle classi, ogni studente ha avuto il compito di creare delle domande da porre al responsabile del progetto con lo scopo di individuare su che punti bisognava soffermarsi principalmente.



**INTERFACCIA WEB SERVICE**

Successivamente alla conclusione del **progetto ASL** ho deciso di crearci dei **Web Service .**

I **Web Service**, non sono altro che dei servizi che vengono richiamati tramite il **protocollo HTTP**.

Questi Web Servici, consentono alle applicazioni che si collegano di usufruire delle funzioni che mette a disposizione.



Per farne un esempio, quando stiamo collegati al sito del ASL, e vogliamo vedere tutte le aziende dove è possibile svolgere l'alternanza scuola lavoro, il sito invece di effettuare la connessione al database ed eseguire la QUERY, utilizzeremo una uri che ci permetterà di accedere al servizio che ci restituirà l'elenco delle aziende.

Un Web service è in grado di offrire un’interfaccia software assieme alla descrizione delle sue caratteristiche, cioè è in grado di farci sapere che funzioni mette a disposizione (senza bisogno di conoscerle a priori) e ci permette inoltre di capire come vanno utilizzate.

Ciò significa che (sempre per rimanere al nostro esempio) con una semplice connessione al servizio per avere una lista delle aziende, anche senza conoscerlo, possiamo stabilire le operazioni che fornisce e possiamo subito iniziare ad usarle perché ogni operazione ha una sua descrizione comprendente i parametri che si aspetta di ricevere, quelli che restituirà ed il tipo di entrambi .

**PROTOCOLLO HTTP**

I servizi REST sfruttano il protocollo HTTP che è un insieme di regole che permettono di trovare uno standard di comunicazione tra diversi computer attraverso la rete, dove per rete si intende un insieme di due o più computer connessi tra di loro ed in grado di condividere informazioni.

Quando due o più computer comunicano tra di loro si scambiano una serie di informazioni. Per potersi scambiare informazioni, i vari computer devono avere dei protocolli che permettano di attribuire ad un determinato comando un significato univoco per tutte le macchine.



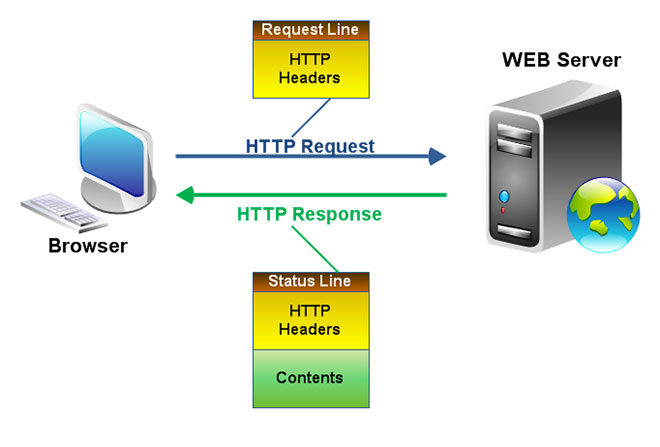
**L'HTTP** è un protocollo "stateless" (senza memoria) che permette sia la ricerca che il recupero dell'informazione in maniera veloce.

La scelta di un protocollo "stateless", cioè di un protocollo che non "conserva memoria" della connessione fatta, è stata necessaria affinché fosse possibile saltare velocemente da un server ad un altro attraverso i "links" ipertestuali.

Ad ogni richiesta, si effettua una nuova connessione al server che viene chiusa al termine del trasferimento dell'oggetto richiesto (pagina HTML, immagine, testo, ecc.).

È gestito da un software (server HTTP) residente sugli host che intendono essere fornitori di informazioni. Chi vuole accedere alle informazioni fornite dal server **HTTP** deve utilizzare un software client (browser) in grado di interpretare le informazioni inviate dal server.

IL **SERVER**, informalmente, è un programma che "gira" in attesa di una richiesta di connessione sul suo socket (la porta assegnatagli, tipicamente la 80).



Questo protocollo ogni qualvolta **l'URL** (che è una stringa che specifica la risorsa a cui riferirci) istanziata contiene nel primo campo la parola http viene utilizzato questo protocollo.

I comandi utilizzati per comunicare con esso sono detti Metodi.

I metodi indicano l’operazione che deve essere eseguita sulla risorsa identificata dal Request-URI.

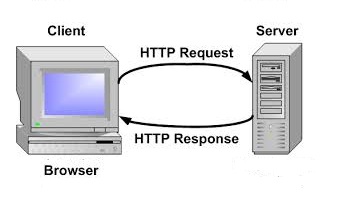
Nel campo *Metdo* della*Request-Line* appare uno solo dei seguenti metodi :

**"OPTIONS", "GET" ,"HEAD", "POST", "PUT" ,"DELETE" ,"TRACE", "CONNECT".**

IL metodo utilizzato viene riconosciuto dal server se è implementato su di esso altrimenti il server restituisce un intero che indica la non implementazioe del metodo specificato.

Il metodo "GET" è sempre supportato da qualsiasi server mentre gli altri sono opzionali.

**Fasi di comunicazione**



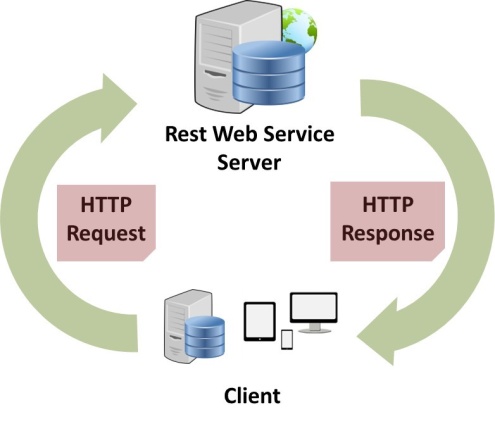
L'acquisizione del documento da parte del client può essere schematizzata in quattro fasi :

* **CONNESSIONE** : Il client crea una connessione TCP-IP con il server usando il suo nome di dominio (o il numero IP) ed il numero della porta di trasmissione; se non viene fornito il numero di porta, il protocollo assume per default che il numero sia 80.
* **RICHIESTA** **DOCUMENTO** : Il client invia la richiesta di un documento mediante una riga di caratteri ASCII terminata da una coppia di caratteri CR-LF (Carriage Return, Line Feed).
* **RISPOSTA** : La risposta inviata dal server è un messaggio in linguaggio HTML nel quale è contenuto il documento richiesto (o un messaggio d'errore).
* **SCONNESSIONE** : Il server subito dopo aver spedito il documento si sconnette. Comunque anche il client può interrompere la connessione in ogni momento, in questo caso il server non registrerà nessuna condizione d'errore.

**SERVIZIO REST**

Che cosa sono i **servizi REST** che sono andato a creare ?

La definizione "Servizi REST" è apparsa per la prima volta nel 2000 nella tesi di **Roy Fielding**, discussa presso l’Università della California. In questa tesi si analizzavano alcuni principi alla base di diverse architetture software, tra cui appunto i principi di un’architettura software che consentisse di vedere il Web come una piattaforma per l’elaborazione distribuita.



È bene precisare che i principi **REST** non sono necessariamente legati al Web, nel senso che si tratta di principi astratti di cui il **World Wide Web** ne risulta essere un esempio concreto.

All’epoca questa visione del Web passò un po' in sordina, ma negli ultimi anni l’approccio **REST** è venuto alla ribalta come metodo per la realizzazione di **Web Service** altamente efficienti e scalabili ed ha al suo attivo un significativo numero di applicazioni.

Il motivo è semplice: dal momento che il Web ha tutto quello che serve per essere considerata una piattaforma di elaborazione distribuita secondo i principi **REST**, non sono necessarie altre sovrastrutture per realizzare quello che è il Web programmabile. Il che è una dichiarazione di aperto antagonismo ai **Web Service** basati su **SOAP**.

Che cos’è REST?

Abbiamo detto che **REST** non è un’architettura ne uno standard, ma un insieme di linee guida per la realizzazione di una “architettura di sistema”. Ma quali sono questi principi che rendono il Web adatto a realizzare **Web Service** secondo l’approccio **REST**? Il tutto può essere riassunto nei seguenti cinque principi:

* Identificazione delle risorse
* Mappare le azioni ‘CRUD’ sui metodi HTTP
* Risorse autodescrittive
* Collegamenti tra risorse
* Comunicazione senza stato

## Identificazione delle risorse

Le risorse sono gli elementi fondamentali su cui si basano i **Web Service** RESTful.

Per risorsa si intende un qualsiasi elemento oggetto di elaborazione. Per fare qualche esempio concreto, una risorsa può essere un cliente, un libro, un articolo, un qualsiasi oggetto su cui è possibile effettuare operazioni. Per fare un parallelo con la programmazione ad oggetti possiamo dire che una risorsa può essere assimilata ad una istanza di una classe.

Il principio che stiamo analizzando stabilisce che ciascuna risorsa deve essere identificata univocamente. Essendo in ambito Web, il meccanismo più naturale per individuare una risorsa è dato dal concetto di URI.

Il principale beneficio nell’adottare lo schema URI per identificare le risorse consiste nel fatto che esiste già, è ben definito e collaudato e non occorre pertanto inventarsene uno nuovo.

I seguenti sono esempi di possibili identificatori di risorse:

* http://localhost/ASL/Stage/getAll

Gli URI sono abbastanza auto esplicativi: La URI identifica tutti gli stage.

L’interpretazione che abbiamo dato a queste URI è però desunta dalla semantica delle parole contenute nelle sue parti. Dal punto di vista di un **Web Service** un URI è soltanto una stringa che identifica una risorsa, per cui anche http://www.myapp.com/tgw34/2099ww può essere un URI valido per identificare un cliente.

# Mappare le azioni ‘CRUD’ sui metodi HTTP

Una volta spiegato come individuare una risorsa abbiamo bisogno di un meccanismo per indicare quali operazioni effettuare su di esse. Il principio dell’**uso esplicito dei metodi HTTP** ci indica di sfruttare i metodi predefiniti di questo protocollo, e cioè **GET**, **POST**, **PUT** e **DELETE**.

Facciamo un semplice esempio per provare a chiarire questo concetto. Quando inseriamo un URI nella barra degli indirizzi di un browser stiamo in realtà chiedendo al browser di eseguire un metodo HTTP sulla risorsa individuata dall’URI. Il metodo che implicitamente stiamo eseguendo è GET, il cui effetto è l’accesso ad una rappresentazione della risorsa identificata dall’URI.

Dal punto di vista del codice, non avremo bisogno di un metodo del tipo getCliente(1234) per ottenere la rappresentazione del cliente con codice 1234, sarà sufficiente sfruttare il metodo standard GET del protocollo HTTP sull’URI che identifica quel determinato cliente.

Questo rende uniforme l’invocazione di operazioni sulle risorse, cioè il client non ha bisogno di sapere qual è la specifica interfaccia da utilizzare per invocare il metodo che consente di ottenere la rappresentazione di un cliente. In un contesto non RESTful potremmo avere metodi come getCliente() o getCustomer() o altra specifica dipendente dalle scelte di chi ha sviluppato il **Web Service**.

In un contesto RESTful sappiamo già a priori come ottenere la rappresentazione di una risorsa. In altre parole, questo principio **REST** stabilisce una mappatura uno a uno tra le tipiche operazioni **CRUD** (creazione, lettura, aggiornamento, eliminazione di una risorsa) e i metodi HTTP.

|  |  |  |
| --- | --- | --- |
| Metodo HTTP | Operazione CRUD | Descrizione |
| POST | Create | Crea una nuova risorsa |
| GET | Read | Ottiene una risorsa esistente |
| PUT | Update | Aggiorna una risorsa o ne modifica lo stato |
| DELETE | Delete | Elimina una risorsa |

È opportuno notare che questo principio è in contrasto con quella che è la tendenza generale nell’utilizzo dei metodi HTTP. Infatti, molto spesso viene utilizzato il metodo GET per eseguire qualsiasi tipo di interazione con il server. Ad esempio, spesso per l’inserimento di un nuovo cliente nell’ambito di un’applicazione Web viene eseguita una richiesta di tipo GET su un URI del tipo:

http://www.myapp.com/addCustomer?name=Rossi

Questo approccio non è conforme ai principi **REST** perchè il metodo GET serve per accedere alla rappresentazione di una risorsa e non per crearne una nuova. In pratica questo uso del metodo GET introduce un effetto collaterale che, tra l’altro, può avere delle conseguenze indesiderate se, ad esempio, l’URI viene invocato più volte o se la risorsa viene memorizzata in uno dei vari livelli di cache esistenti tra client e server.

Come principio generale, nel progettare un **Web Service** in modalità **REST** è utile evitare l’uso di verbi negli URI ma limitarsi ad utilizzare nomi, ricordandosi che un URI identifica una risorsa.

Allo stesso modo è opportuno sottolineare che il corpo di una richiesta HTTP con metodi PUT e POST è pensato per il trasferimento della rappresentazione di una risorsa e non per eseguire chiamate remote o altre attività simili.

Abbiamo sottolineato come le risorse giocano un ruolo fondamentale nell’approccio **REST** ed abbiamo visto come identificarle e come gestirle. Ma come sono rappresentate?

## Risorse autodescrittive

## Image result for url rest

Le risorse di per sè sono concettualmente separate dalle rappresentazioni restituite al client. Ad esempio, un **Web Service** non invia al client direttamente un record del suo database, ma una sua rappresentazione in una codifica dipendente dalla richiesta del client e/o dall’implementazione del servizio.

I principi REST non pongono nessun vincolo sulle modalità di **rappresentazione di una risorsa**. Virtualmente possiamo utilizzare il formato che preferiamo senza essere obbligati a seguire uno standard. Di fatto, però, è opportuno utilizzare formati il più possibile standard in modo da semplificare l’interazione con i client.

La possibilità di rappresentazioni multiple produce alcuni benefici pratici: ad esempio, se abbiamo un output sia HTML, sia XML, possiamo consumare il servizio sia con un’applicazione sia con un comune browser. In altre parole, seguendo i principi REST nel progettare un’applicazione Web è possibile costruire sia una **Web API** che una **Web UI**.

**Collegamenti tra risorse**

Un altro vincolo dei principi REST consiste nella necessità che le risorse siano tra loro messe in **relazione tramite link** ipertestuali. Questo principio è anche noto come HATEOAS, dall’acronimo di Hypermedia As The Engine Of Application State, e pone l’accento sulle modalità di gestione dello stato dell’applicazione.

In sostanza, tutto quello che un client deve sapere su una risorsa e sulle risorse ad essa correlate deve essere contenuto nella sua rappresentazione o deve essere accessibile tramite collegamenti ipertestuali.

Il fatto di utilizzare un URI come identificatore di una risorsa, quindi un meccanismo standard e consolidato, consente al client di accedere anche a risorse messe a disposizione da altre applicazioni che girano eventualmente su altri server.

Inoltre, la possibilità di avere collegamenti ipertestuali all’interno della rappresentazione di una risorsa rappresenta un meccanismo per poter gestire lo stato dell’applicazione, come vedremo in seguito.

## Image result for web

## Comunicazione senza stato

## Image result for stateless httpImage result for stateless http

Il principio della **comunicazione stateless** è ben noto a chi lavora con il Web. Questa è infatti una delle caratteristiche principali del protocollo HTTP, cioè ciascuna richiesta non ha alcuna relazione con le richieste precedenti e successive. Lo stesso principio si applica ad un Web Service **RESTful**, cioè le interazioni tra client e server devono essere senza stato.

È importante sottolineare che sebbene REST preveda la **comunicazione stateless**, non vuol dire che un’applicazione non deve avere stato. La responsabilità della gestione dello stato dell’applicazione non deve essere conferita al server, ma rientra nei compiti del client.

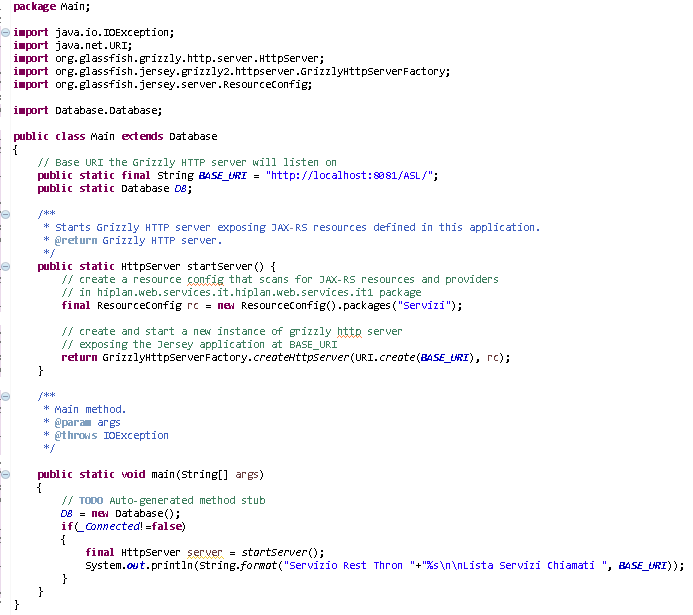
La principale ragione di questa scelta è la scalabilità: mantenere lo stato di una sessione ha un costo in termini di risorse sul server e all’aumentare del numero di client tale costo può diventare insostenibile.

**WEB SERVER EMBEDDED GRIZZLY**

Scrivere applicazioni con server scalabili nel linguaggio di programmazione Java è sempre stato difficile, i problemi di gestione dei thread rendevano impossibile la scalabilità di un server a migliaia di utenti.

**Il framework NIO di Grizzly** è stato progettato per aiutare gli sviluppatori a sfruttare **l'API NIO Java** (API di Input e Output).

 L'obiettivo di **Grizzly** è di aiutare gli sviluppatori a creare server scalabili e robusti utilizzando NIO oltre a offrire componenti Web **Framework** (**HTTP**), **WebSocket** e molto altro!



**JAVA E FRAMEWORK JERSEY**

**JAVA**

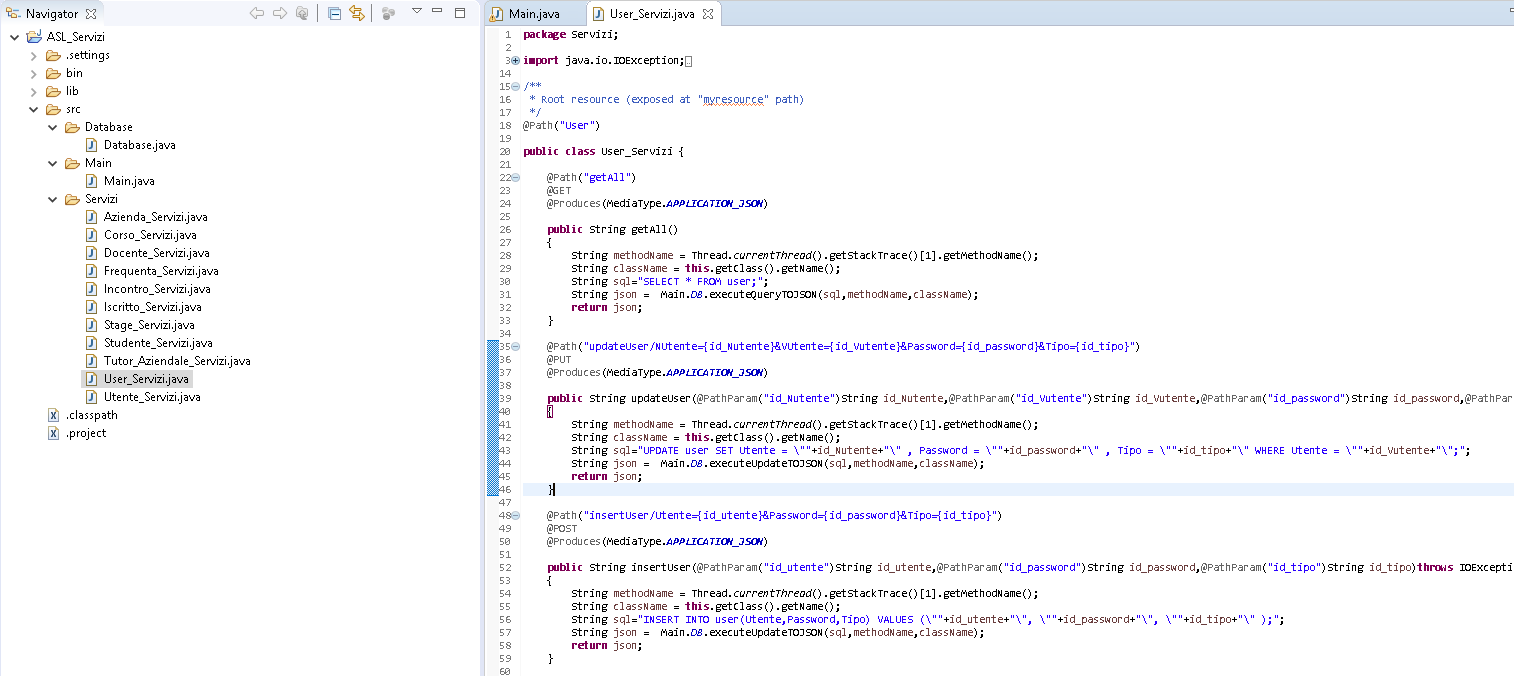
****

Per scrivere il codice dietro al servizio REST ho utilizzato il linguaggio **Java** che è un [linguaggio di programmazione ad alto livello](https://it.wikipedia.org/wiki/Linguaggio_di_programmazione_ad_alto_livello), [orientato agli oggetti](https://it.wikipedia.org/wiki/Programmazione_orientata_agli_oggetti) e a [tipizzazione statica](https://it.wikipedia.org/wiki/Tipizzazione_statica)(alle variabili viene assegnato un valore prestabilito).

in Java, il [programmatore](https://it.wikipedia.org/wiki/Programmatore) può utilizzare librerie di terze parti. Queste librerie, contenute in vari [package](https://it.wikipedia.org/wiki/Package_(Java)), vengono utilizzate dal [programmatore](https://it.wikipedia.org/wiki/Programmatore) per utilizzare determinati [metodi](https://it.wikipedia.org/wiki/Funzione_(informatica)) o [attributi](https://it.wikipedia.org/wiki/Variabile_(informatica)) comuni per semplificare e uniformare i [programmi](https://it.wikipedia.org/wiki/Programmi) e renderli maggiormente leggibili ai programmatori. Esistono numerosissimi package utilizzabili dai programmatori nel [linguaggio Java](https://it.wikipedia.org/wiki/Linguaggio_Java).

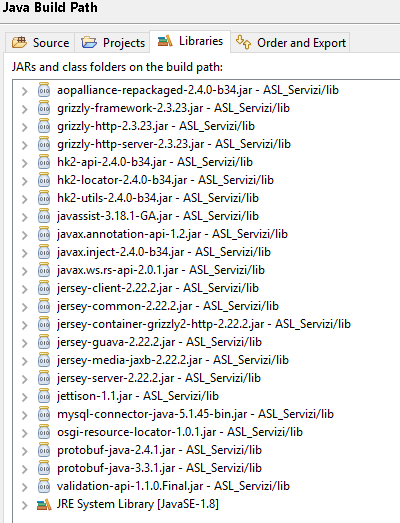
**ECLIPSE**

**Eclipse** è un ambiente di sviluppo multipiattaforma che permette di scrivere e mandare in esecuzione un codice scritto in java .



**FRAMEWORK JERSEY**

**Jersey** è l’implementazione per la realizzazione di **Web Service RESTful** su piattaforma Java. Jersey permette di creare risorse utilizzando delle specifiche annotazioni per i metodi e le classi. In altre parole il **framework** si occupa di gestire le richieste **HTTP** e la negoziazione della rappresentazione e noi possiamo concentrarci alla soluzione del problema. Per utilizzarlo `e sufficiente che siano disponibili nel **classpath**(è una variabile d'ambiente contenente una serie di **path** (essenzialmente di percorsi di cartelle) in cui il programma/compilatore cerca di default i file) le librerie.



I @**Path** indica la **URI** a cui la risorsa sarà raggiungibile, in questo caso sarà, ad esempio http://localhost/ASL/Stege/getAll. L’annotazione `e posta al livello della dichiarazione della classe, quindi 'Stage' sarà il **path** di base per tutti i suoi metodi. I @**Produces** specifica uno o più MIME-Type con cui lo stato della risorsa può essere trasferito al client che ne fa richiesta. Se il client non ne supporta neanche uno il **framework** risponderà con l’opportuno codice di errore HTTP. I @**GET** evidenzia il comando HTTP che il metodo `e incaricato di gestire.

**JSON**

Per il trasferimento dei dati ho deciso di utilizzare il **Json** che è un semplice formato per lo scambio di dati, per le persone è facile da leggere e scrivere mentre per le macchine è facile da generare e analizzare.

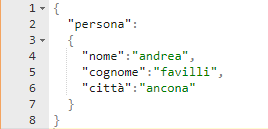
E’ un formato di testo completamente **indipendente** dal linguaggio di programmazione, ma utilizza una struttura facilmente deducibile da programmatori di linguaggi della famiglia del C, come C,C++, C#,Java,JavaScript e molti altri. Questa caratteristica fa di JSON un linguaggio ideale per lo scambio di dati.

https://lh3.googleusercontent.com/FaEO7tQqJrdmzmHGFdh4Ej9A3p_DxmeU3sztBLEicC83Q0wLmPNjBNhze6-8VYcZIkD54-uTUCpILK0Nasjh0_G1FCAYUvxSsqxqhUQWMAY2nDAw1zH8l_Pi1er7Tw6cyT_QceeOOaE

**Stringhe** => devono essere scritte tra virgolette

**https://lh3.googleusercontent.com/9GMU02wUCn8sPhaOKdqt1m8FKQ4UGDDdMhSb9W8WPq-qX5fnD3h9TgGGKc30AbliEr8rBVzbmppBTExx-rgPL09llZcLo807H7ruB0oCFswfvav-CqqwsOw5zIlwuXGb29IlxDyZ4EINumeri** => possono essere interi o

in virgola mobile



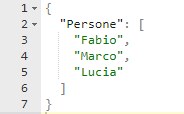
**Oggetti**

**https://lh6.googleusercontent.com/UyJ1oGfq5fe9j9jBJyGUnkxTCgo6oTpCDW0HpolLdZbNDZNXWdTPWbCOOMdIx4sghywPL1sStwIiauHC25Z3q_1OR3nq2OvlO2E2BgH73ZjXR9XIXkeihixHhk8jAZavsy_2-AhntEg**

**Boolean**

https://lh5.googleusercontent.com/i9qfMOltMsmbbQCqBxb6tuPbAUmMw3TqZzJhT4XPO3zOoVNldtZehpz9fnX6O_EKkcJd4Cqc2i0inJyK2Ch-bJipB7WJBsy6XLZE8nVcnFTZ7AgGku79DKx3jzCPC4MyJFI2A3iuyIk

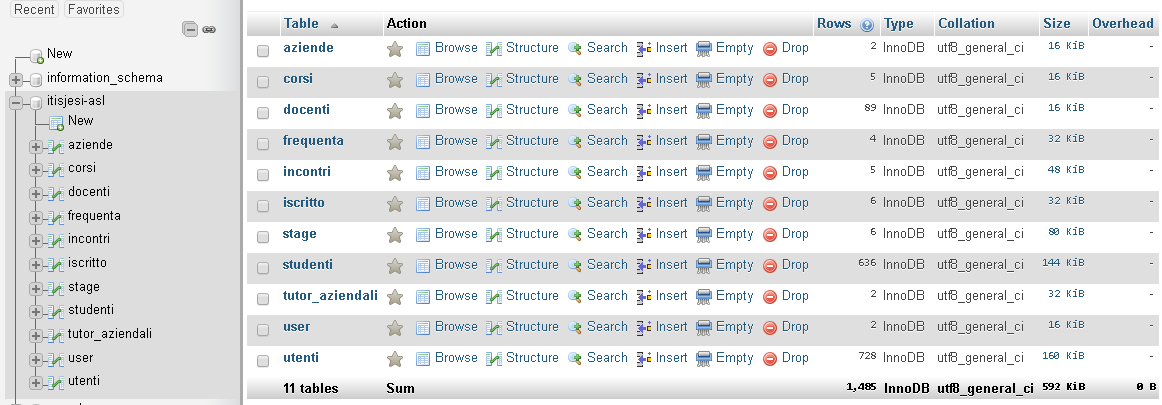
**Null**



**Array**

**SCHEMA DEL DB ASL**

I servizi REST che andremo ad eseguire, lavorano utilizzando un database chiamato **itisjesi-asl**.



Con il termine **database** si indicano, in informatica, gli **archivi di dati**,

organizzati in modo integrato, progettati con tecniche di modellazione dei dati e gestiti sulle memorie di massa dei computer utilizzando appositi software, con l’obiettivo di manipolare e ritrovare i dati memorizzati in modo efficiente, superando anche i limiti presenti nelle organizzazioni tradizionali degli archivi.

Il **database** è una collezione di **archivi** di dati ben organizzati e ben strutturati, che sono gestiti in modo integrato e che costituiscono una base di lavoro per utenti diversi con programmi diversi. I prodotti software per la gestione dei database sono indicati con il termine **DBMS,** acronimo di **DataBase** **Management** **System**.



I database sono costituiti da tabelle che rappresentano le **entità** e le relazioni dello schema concettuale.

Queste **entità** sono composte da record ed uno o più di questi è la chiave primaria che identifica univocamente un **record**.

Quando non è possibile identificare una chiave primaria viene creato un campo ID che è un valore auto incrementante per identificare un record.

**LINGUAGGIO** **SQL**

**SQL** (Structured Query Language) è il linguaggio che permette di effettuare le operazioni per estrarre e manipolare i dati da un database.

E’ lo standard tra i sistemi relazionali : viene usato in tutti i prodotti DBMS come set di comandi per l’utente della base di dati

Come per creare una tabella (**CREATE** )

Prendere dei dati (**SELECT**)

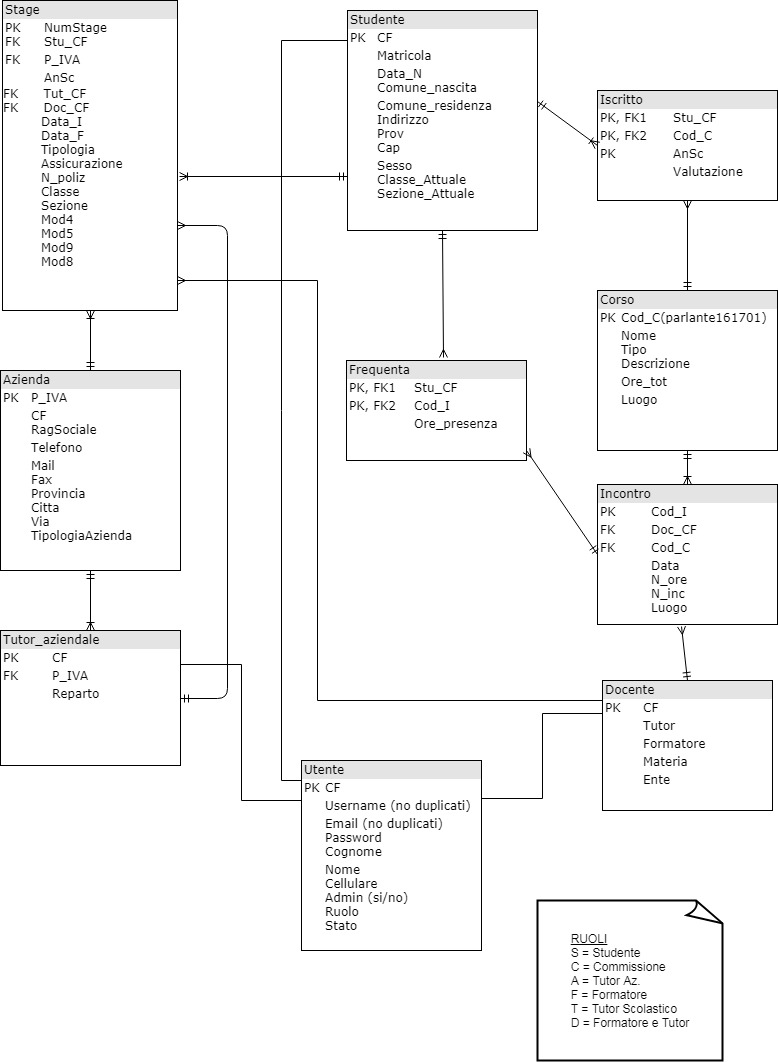
Cancellare dei record (**DELETE**)

Modificare dei record (**UPLOAD**)

Inserire dei record (**INSERT**)

Unire delle tabelle (**JOIN**)

**Schema del DATABASE ASL**

****

User

PK Utente

Password

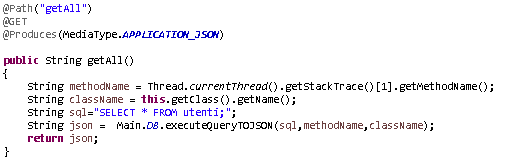
Tipo

**ESEMPIO DI CODIFICA DEI SERVIZI REST ASL**

Questi servizi che sono andato a creare svolgono funzioni di **GET**, **POST**, **PUT** e **DELETE** (select , insert, update e delete) per ogni tabella del database ITISJESI-ASL.

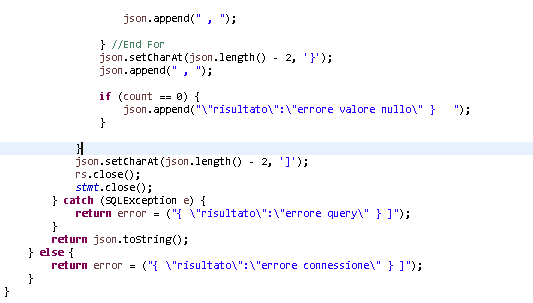
Per i servizi che svolgono funzioni di **GET** (select), viene eseguita la funzione **executQueryTOJSON**() che permette di eseguire la **Query** e di trasformare il risultato della **Query** in una stringa **JSON**

Esempio :



Metodo per trasformare risultato della **Query** in **JSON** :



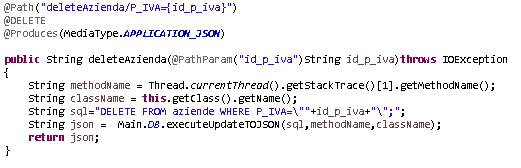


Valore restituito :

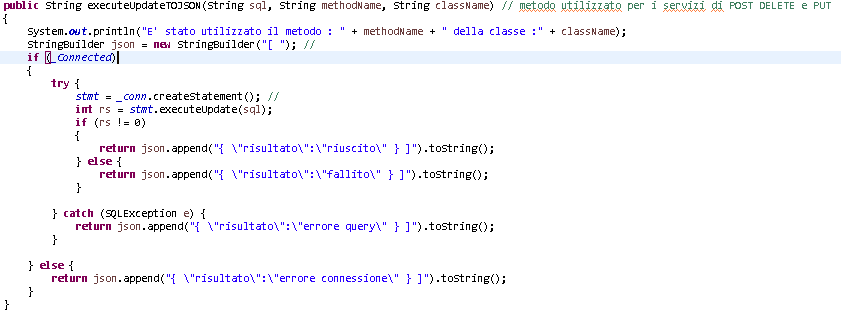


Mentre per eseguire metodi di **POST**, **PUT** e **DELETE** , che hanno bisogno di un'altro comando per eseguire una Query utiliziamo il metodo **executeUpdateTOJSON**() che permette di eseguire la **Query** e di controllare il valore che ritorna è stata eseguita correttamente e ritorno un messaggio **JSON** che conferma o no, l'avvenuta esecuzione.

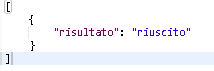
Esempio :



Metodo per l'esecuzione della **Query**(POST, PUT e DELETE)



Valore restituito (nel caso che venga eseguito):



Valore restituito (nel caso che non venga eseguito):

