ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Scuola di Scienze

Corso di Laurea in Informatica

L’introduzione della metodologia REST nella riprogettazione di Web services pre-esistenti

Tesi di laurea in Tecnologie Web

Relatore
Prof. Fabio Vitali

Presentata da
Gianni Cavicchi

II Sessione

Anno Accademico 2019/2020

Indice

Capitolo 1

Introduzione

Questa tesi nasce con lo scopo di spiegare come, attraverso il REpresentational State Transfer (REST), si riesca a rivoluzionare la filosofia con cui si progettano e sviluppano le API.

Il Web, a partire dalla sua creazione, ha vissuto un'espansione e una evoluzione costante. Agli albori questo sistema distribuito, che collegava relativamente pochi elaboratori, ha iniziato la propria espansione, permessa soprattutto dalla diffusione dei sistemi informativi.

Dal punto di vista dell'utente, il Web non è più solamente una collezione di pagine statiche come semplici volantini o cataloghi e nemmeno un insieme di servizi interattivi e modificabili dall'utente. Ad oggi il Web conta 6,4 miliardi di dispositivi connessi e non più solamente computer o smartphone: è l'avvento dell'Internet Of Things.

Tutto può essere connesso alla rete e scambiare dati, per di più in tempo reale e in maniera costante.

Ovviamente, durante questo processo di evoluzione, si sono susseguiti numerosi bisogni, tecnologie e tecniche che sono state sviluppate di conseguenza.

Nonostante inizialmente non vi fossero esigenze, in quanto le interazioni erano tutte utente-servizio, non possiamo certamente dire lo stesso per le interazioni macchina-macchina.

Queste connessioni erano necessarie per lo scambio di servizi, si richiedevano procedure e funzioni al fine di elaborare dati che possedeva il richiedente.

Nacque così la *Service Oriented Architecture* (SOA). Le applicazioni e i Web services dovettero quindi essere costruiti in maniera tale da poter permettere un relativamente semplice scambio di messaggi contenenti le funzioni o i dati elaborati attraverso di esse.

In questo determinato periodo storico i Web services sono molto complessi e, così come essi, anche i messaggi scambiati. La flessibilità è poca e la manutenzione è difficile, inoltre la riusabilità di questi servizi è praticamente nulla.

Questa epoca, descritta quasi come preistorica, in realtà non è ancora finita completamente.

La tecnologia avanza e con essa il numero e le tipologie di dispositivi interconnessi come, ad esempio, tutto ciò che include un sistema *embedded* (integrato), nonostante non possa eseguire calcoli molto complessi, può ricavare attraverso sensori i dati, per poi esporli attraverso la rete.

Le dimensioni di gran parte dei sistemi sono andate via via riducendosi ma, allo stesso tempo, sono entrati in gioco possessori di risorse e dati sempre più grandi, i cosiddetti *Big Data.*

Questo nuovo modo di vedere il Web, come nel periodo precedente, ha portato ad un nuovo focus e di conseguenza ad un nuovo approccio architetturale chiamato ROA (Resource Oriented Architecture).

Qui entra in gioco REST. Questo stile architetturale, nato nel 2000, permette di cambiare profondamente struttura dei Web service e delle connessioni rendendoli più versatili, interoperabili e debolmente connessi con altri servizi come era prima del suo avvento, permettendo così la diffusione delle risorse a tutti.

Non tutti però si possono definire conformi con lo stato dell'arte. Alcuni Web services presentano ancora una impostazione SOA, molto spesso anche con risorse rilevanti a livello nazionale.

Un esempio è la Scheda Unica Annuale del Corso di Studio (SUA-CdS), uno strumento di monitoraggio e autovalutazione degli Atenei nazionali sviluppato presso il CINECA Consorzio Interuniversitario per conto del Ministero dell'Istruzione della Ricerca e dell'Università.

In questa tesi, che segue il lavoro svolto dal sottoscritto durante il tirocinio curriculare presso l'azienda appena citata, verrà sviluppata un'API REST, chiamata "progetto REST-AURATION", che tenterà di dimostrare il vantaggio ottenuto con questo stile architetturale, oltre che rielaborare i contenuti di uno strumento così interessante.

Questa dissertazione è organizzata come segue: nel primo capitolo andremo ad introdurre il problema e le tecnologie utilizzate nella storia; nel secondo approfondiremo REST e come si è evoluto nel corso degli anni; nel terzo capitolo parleremo del progetto REST-AURATION e le motivazioni del suo sviluppo; il quarto capitolo illustra le funzionalità del progetto REST-AURATION e nel capitolo successivo seguiranno le valutazioni; nel sesto capitolo sarà presente una breve conclusione.

1.1 REST nella letteratura

Il progetto per il quale è stata redatta questa tesi non è altro che l'ennesimo caso di una reingegnerizzazione di un sistema legacy (obsoleto) trasformato in una API REST. Per la maggioranza dei casi, ci troviamo a dover interagire con sistemi sviluppati secondo il protocollo SOAP, in cui non è possibile riutilizzare il codice, i messaggi presentano payload rilevanti e richiedono sistemi molto complessi, sia per chi richiede i dati o i servizi, sia per chi li possiede. REST d'altro canto è visto come un approccio rivoluzionario, capace di rendere il codice riutilizzabile e i sistemi molto più semplici.
Il caso di studio di riferimento è quello descritto in "*Reengineering Legacy Systems with RESTful Web Service*". [LW08]
Il lavoro spiega dettagliatamente il perché e come reingegnerizzare un sistema legacy quasi come se fosse un manuale.
I motivi chiave sono:

- la riusabilità delle funzionalità. Riutilizzare codice è importante e con SOAP non è possibile.
- L'estrazione delle capacità dei servizi. Questo può evitare la ridondanza dei servizi di costruzione, estraendo servizi dai sistemi legacy anche senza richieste dirette da parte degli utenti finali.
- Possibili "mash-up". Sotto la rivoluzione del Web 2.0, i mash-up rappresentano un nuovo promettente approccio allo sviluppo per la creazione di applicazioni composite e consentono l'integrazione rapida di applicazioni web esterne (esistenti e future).
- La rapidità e semplicità nell'effettuare aggiornamenti e modifiche è una caratteristica fondamentale dei servizi REST.

Esistono però alcune difficoltà come, ad esempio, quella di riuscire a identificare le risorse e uscire dalla concezione che siano dati, sono "cose RESTful". Bisogna, quindi, riuscire a identificare e classificare le risorse. Le risorse sono identificabili tramite URI che devono essere, per la filosofia REST e per i benefici che ne conseguono, disegnati per avere un'alta interoperabilità e scalabilità.
L'applicazione di tutto ciò che è stato detto in precedenza viene applicato su un legacy system basato su SOAP per la gestione di candidati (come ad esempio studenti) all'interno di una facoltà facente parte di un Ateneo (caso molto simile a quello di questa tesi).

Grande spazio viene dato alla identificazione delle risorse, alle collezioni di risorse e alla loro gerarchia.
Uno studente sarà una risorse in quanto insieme di informazioni e potrà poi essere inserito in una classe nella quale svolgerà determinati corsi. La classe potrà essere all'interno di una facoltà.
Tutte queste relazioni e altre sono determinanti per la creazione degli URI in quanto uno studente non può esistere all'infuori di una classe e la classe per definizione non è altro che una collezione di risorse che sono gli studenti.
Come è di facile intuizione, le risorse possono essere in molte collezioni e sono in stretta correlazione, rendendo "l'esplorazione" dei contenuti molto rapida e molto intuitiva.
Successivamente le operazioni effettuabili sulle risorse vengono definite in relazione agli URI e alle interconnessioni in maniera molto semplice. [LW08]
Dopo un periodo relativamente lungo, nel 2017, viene proposto un altro lavoro dal nome "*Developing a Prototype of REST-Based Database Application for Shipbuilding Industry*" [JC17], in cui viene proposta la rielaborazione di sistema legacy riguardante un servizio di gestione dei materiali (BOM) nell'industria navale.
Il motivo per cui è stato scelto BOM è il fatto che abbia numerose utenze, sia all'interno che all'esterno del servizio.
Come nel lavoro precedente, dopo un'attenta analisi e valutazione dei benefici, si è scelto di rielaborare il sistema utilizzando l'architettura REST in quanto permette di creare una comunicazione molto semplice tra sistemi debolmente connessi.
Successivamente viene illustrato come, attraverso nuove tecnologie, come possono essere Javascript e lo sfruttamento completo di HTTP attraverso lo stile architetturale REST.
Per terminare l'elaborato, vengono elogiate la scalabilità del Web service e la riduzione dei problemi legati alle comunicazioni attraverso la rete.
Questi due elaborati hanno ispirato prepotentemente il progetto REST-AURATION, oltre che a fungere da linea guida per lo sviluppo insieme ad altri lavori che verranno illustrati nel capitolo 2.2.[JC17]

1.2 Il progetto REST-AURATION

Durante il tirocinio curriculare svolto dal sottoscritto presso il CINECA Consorzio Interuniversitario, più nello specifico presso l'ufficio operante per il MIUR, inizia la rielaborazione dell'applicazione Scheda Unica Annuale del Corso di Studio (SUA-CdS).

La SUA-Cds è uno strumento di monitoraggio composto da testi per l'autovalutazione dei Corsi di studio nazionali, progettata su richiesta del Ministero dell'Istruzione dell'Università e della Ricerca (MIUR).

I motivi per cui l'applicazione SUA-CdS necessita di una rielaborazione sono prevalentemente due: il fatto che sia sviluppato attraverso il protocollo SOAP e la perdita di appetibilità da parte di questo applicativo.

Attraverso i vincoli e i principi REST andremo ad elaborare una API, che fornisce un'astrazione, evitando al programmatore di sapere come funziona "ciò che sta sotto", semplificandone l'utilizzo.

Le risorse devono essere identificate e classificate. Questi oggetti devono essere identificabili tramite URI che a loro volta devono essere, per la filosofia REST e per i benefici che ne conseguono, disegnati per avere un'alta interoperabilità e scalabilità. Questi oggetti possono essere inoltre raggruppati in collezioni.

Il punto di partenza di questa rielaborazione sono quindi le risorse esposte. Andremo ad ampliare il pool di queste e come queste vengono esposte attraverso gli identificatori.

Gli URI RESTful seguono una logica gerarchica ben precisa e così anche quelli della nostra API dovranno. Definiremo una logica partendo dalle risorse e collezioni più ampie e generali addentrandoci sempre di più nelle risorse specifiche di un corso di studi.

Come punto di partenza avremo gli Atenei, composti da Corsi di studio, che a loro volta possiedono risorse come, ad esempio: i testi della SUA-CdS, gli insegnamenti erogati dal Corso e lo storico. Agli URI sono ovviamente associate le risorse, estratte da un database Oracle di sviluppo e rielaborate.

L'aggiornamento dei testi della SUA-CdS è un'altra funzione chiave del progetto REST-AURATION. Questo è possibile attraverso il codice numerico identificativo del testo e il nuovo testo che si ha intenzione di inserire.

Poiché la SUA-CdS è un documento standardizzato non è possibile eliminare i testi e nemmeno lasciare il testo vuoto.

L'aggiornamento dei testi è possibile solo attraverso autenticazione. Necessita di credenziali quali *username* e *password* che, per il momento, devono essere concordate con il sottoscritto.

1.3 Valutazione di REST-AURATION

Il funzionamento del progetto REST-AURATION è stato valutato attraverso un client costruito “su misura” per poter soprattutto verificare l’accessibilità alle risorse.
Il client è molto semplice e attraverso altrettanto semplici chiamate AJAX effettuiamo richieste HTTP per ottenere tutte le risorse.
In aggiunta alla richiesta delle risorse abbiamo la possibilità anche di aggiornare i testi della SUA-CdS.
Il progetto REST-AURATION, paragonato all’applicazione SUA-CdS sviluppata con protocollo SOAP, presenta tanti punti di forza. Una su tutte la quantità e la semplicità di accesso alle risorse. Inoltre, queste sono molto frammentate grazie al concetto stesso di risorsa e di URI.
La riusabilità del codice e delle risorse è molto elevata e permette anche la costruzione di API “mashup” in cui vengono esposte quelle provenienti da altre API.
A livello di ottimizzazione, i messaggi scambiati tra client e server risultano più leggeri.
Essendo in formato JSON e non XM il messaggio, libero dall’ingombro dei tag di quest’ultimo formato, risulta facilmente leggibile. Avendo ridotto il payload, viene ridotto il carico di rete.
Inoltre, sia il Web server che il client costruito per verificarne il funzionamento si dimostrano molto semplici da implementare e non vincolati da complessi tool per lo sviluppo SOAP.

1.4 Sviluppi futuri e conclusioni

Data la modestia del progetto sicuramente necessita di numerose revisioni dedite alla correzione di errori presenti nel codice.

Altre correzioni possono certamente essere fatte ad alcuni vincoli implementativi come, ad esempio, la sicurezza, implementata attraverso *Digest HTTP*, non conforme allo stato dell’arte.
In una visione complessiva, anche il linguaggio utilizzato è ritenuto obsoleto.
PHP 5.2.14 al giorno d’oggi non è più conforme con lo stato dell’arte, sia per l’effettiva possibilità di sfruttamento del linguaggio che per le funzionalità messe a disposizione.
Altri accorgimenti potrebbero essere legati al mascheramento dell’URL visualizzato attraverso il browser utilizzato per accedere al client e alla possibilità di aggiornare altre risorse come gli insegnamenti di un determinato Corso di studio.

Per concludere, la speranza è che il progetto REST-AURATION possa fungere da "pioniere" per l'utilizzo dello stile architetturale REST all'interno di una realtà nazionale importante come è CINECA e che possa addirittura, nel corso degli anni, andare a sostituire l'applicazione SUA-CdS attuale.

Capitolo 2

REST e altre reingegnerizzazioni nella storia

2.1 Sistemi distribuiti e Web service

Prima dell'avvento dei Web services, il Web presentava ancora notevoli problemi nell'interazione all'interno dei sistemi distribuiti in comunicazione attraverso la rete, nonostante avesse riscontrato un enorme successo nella semplicità di interazione tra utenti e servizi.

Per precisare, un sistema distribuito è una collezione di sistemi indipendenti in comunicazione fra loro attraverso una rete, nel nostro caso il Web.

In informatica un Web Service, in italiano "servizio Web", viene definito dal World Wide Web Consortium (W3C) come: *"Un sistema software progettato per supportare l'interoperabilità tra diversi elaboratori su di una medesima rete ovvero in un contesto distribuito.*" [BH04]

Alla fine degli anni '90 nacque così il protocollo SOAP (Simple Object Access Protocol) che segnò un profondo cambiamento nello sviluppo dei Web services.

SOAP è un protocollo per lo scambio di messaggi strutturati tra Web services basato sul protocollo di trasporto HTTP (HyperText Transfer Protocol) e sul linguaggio di markup XML (acronimo di eXtensible Markup Language).

Un altro concetto fondamentale è il concetto di API (Application Programming Interface), simile al concetto di Web Service ma, mentre questi permettono una semplice interazione tra due macchine all'interno di una rete, una API funge da interfaccia tra due differenti applicazioni permettendogli di comunicare.

Una API fornisce una astrazione che evita al programmatore di sapere come funziona "ciò che sta sotto", semplificandone l'utilizzo.

Il problema che ci poniamo è come costruire queste API, soprattutto in ambito Web.

Di nostro interesse particolare sono le Web API Server-Side.

Sono API formate da uno o più endpoint esposti a messaggi di richiesta e risposta.

Gli Endpoint sono una componente fondamentale delle applicazioni Web. Sono entità necessariamente statiche che identificano dove una determinata risorsa può essere trovata e quindi richiesta. Solitamente sono accessibili tramite URI a cui vengono inviate richieste HTTP.

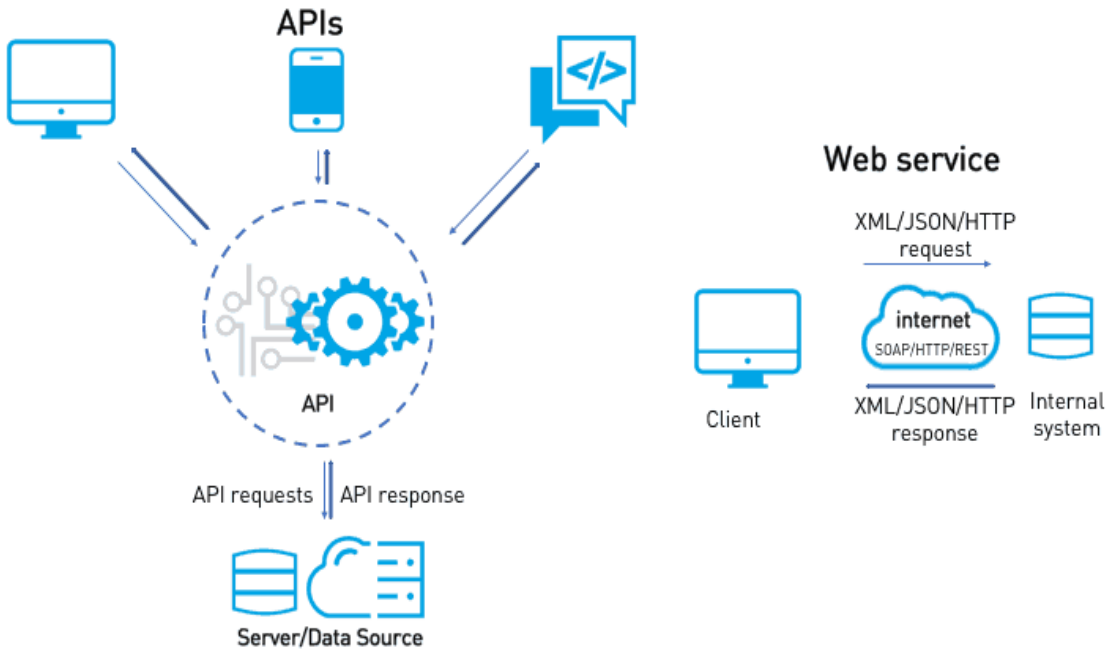


Figura 2.1: Differenza tra API e Web service

Non molto dopo la nascita di SOAP, nel 2000, iniziò ad emergere anche REST (REpresentational State Transfer), uno stile architetturale che, a differenza di SOAP, delineava certi vincoli per lo sviluppo dei Web services e delle API, in modo da renderle più flessibili, snelle, e accessibili, nonché più semplici a livello implementativo ed efficienti.
SOAP e, soprattutto, REST verranno approfonditi nei capitoli seguenti, più precisamente nei capitoli 2.2 e 2.3.

2.2 Simple Object Access Protocol (SOAP)

SOAP (Simple Object Access Protocol) è un protocollo indipendente da HTTP per lo scambio e la negoziazione di informazioni strutturate in formato XML (Extensible Markup Language), linguaggio di markup per la codifica di messaggi e documenti per far sì che risultino leggibili sia dagli umani che automaticamente.

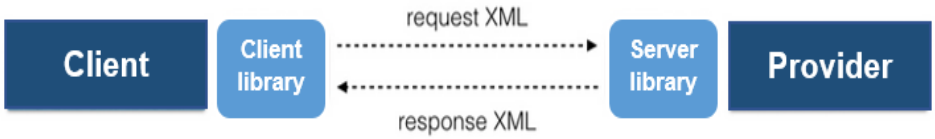


Figura 2.2: Scambio di messaggi attraverso protocollo SOAP

Così è come viene spiegato in un breve stralcio introduttivo del lavoro "Simple Object Access Protocol (SOAP) 1.1" del 2000. [BE00]

"SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP does not itself define any application semantics such as a programming model or implementation specific semantics; rather it defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. This allows SOAP to be used in a large variety of systems ranging from messaging systems to RPC." [BE00]

SOAP usualmente si basa sul protocollo di trasporto HTTP, anche se sarebbe possibile utilizzare anche altri protocolli.

Nonostante si basi su questo protocollo, SOAP permette unicamente l'utilizzo dei metodi HTTP, POST e GET mentre, se sfruttato a pieno, sarebbe possibile utilizzarne altri.

Un messaggio SOAP si compone di quattro elementi:

- *Envelope*: parte obbligatoria, all'interno della quale viene dichiarato il namespace di SOAP, necessario per evitare i conflitti dati dai nomi uguali anche se definiti diversamente.

  Funziona da "involucro" per tutti gli altri.
- *Header*: elemento opzionale che contenente metadati non appartenenti al messaggio, come possono essere la sicurezza e il routing.
- *Body*: che contiene il messaggio vero e proprio.
- *Fault*: contenente gli errori e le informazioni sullo stato della richiesta.

Come è facilmente intuibile, in rapporto all'utilizzo del formato JSON, le dimensioni del messaggio risultano di molto maggiori a causa degli elementi *Envelope* e *Header*.

Di seguito verrà riportato un esempio di un messaggio in formato XML completo di tutti gli elementi.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

Listing 2.1: Messaggio in formato XML

Shiporder sarà l'elemento *Envelope*, contenente tre diversi elementi figlio: *orderperson*, *shipto* ed *element*. L'elemento *item* appare due volte e contiene un *title*, un elemento *note* opzionale, una *quantità* e un elemento *prezzo*.
La riga sopra: xmlns: xsi = "http://www.w3.org/2001/XMLSchema-instance" dice al parser XML che questo documento dovrebbe essere validato su uno schema.
La riga: xsi: noNamespaceSchemaLocation = "shiporder.xsd" specifica DOVE risiede lo schema, permettendo così di definire variabili.
Andando ad effettuare un'analisi che va oltre alla descrizione del contenuto del messaggio, potremo notare quanto effettivamente sia limitante il formato del messaggio. Oltre a ciò che è già stato accennato riguardo all'Overhead del messaggio, i tag presenti vincolano il sistema, che utilizza tale formato di dati ad una manipolazione *ad hoc*,

imponendo che esista un server che rielabori i dati, strutturati da un altro sistema, per poi interfacciarli al pubblico.
Per questo, l'esposizione delle risorse non è quasi mai possibile ad altri Web services ma strettamente vincolata ad una interfaccia per "per umani".

In conclusione, a questa analisi possiamo affermare che le principali controindicazioni nell'utilizzo di SOAP sono:

- L'eccessiva verbosità. I messaggi delle request e dei response sono codificati in un file XML che ha tag lunghi, ciascuno con diversi attributi e, di conseguenza, i messaggi occupano più spazio, e quindi più banda, rispetto a formati come JSON.
- La creazione e l'analisi dei messaggi SOAP-XML sono complessi. I programmatori di Web services devono quindi affidarsi a *tool* e librerie per poter effettuare questo lavoro. Di conseguenza anche i sistemi che elaborano e quelli che richiedono le risorse risulteranno molto complessi.

2.3 REpresentational State Transfer (REST)

Il termine REST (acronimo di "REpresentational State Transfer") fu introdotto nel 2000 nella tesi di dottorato di Roy Fielding, che lo descrive come uno stile architetturale, ovvero un'astrazione degli elementi di un'architettura all'interno di un sistema Ipermediale distribuito.
"*REST The Representational State Transfer (REST) style is an abstraction of the architectural elements within a distributed hypermedia system. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements. It encompasses the fundamental constraints upon components, connectors, and data that define the basis of the Web architecture, and thus the essence of its behavior as a network-based application.*" [FIE00]
L'Ipermedia è una collezione non eterogenea di informazioni che possono essere testuali, grafiche, audio, video, etc. È un'estensione dell'Ipertesto, ossia un documento che lega tra loro altri documenti in maniera non sequenziale tramite connessioni logiche (come ad esempio link ipertestuali).

REST è quindi una convenzione per le comunicazioni client-server "stateless", ossia senza stato, che viene implementata sfruttando al massimo il protocollo di trasporto HTTP anche se tecnicamente sarebbe possibile utilizzare altri protocolli.

REST non è un protocollo, è semplicemente un insieme di assunzioni che si sforzano di creare semplicità e coerenza nella denominazione e determinazione delle risorse, le quali sono univocamente identificate da un "Uniform Resource Identifier" (URI).

"*The Web is an implementation of RESTful principles - It uses URLs to identify resources and HTTP as their service interface.*" [GI11]

Questo piccolo trafiletto, estratto dal lavoro *"In search of an internet of things service architecture REST or WS- A developers' perspective"* [GI11], spiega perfettamente su che cosa si basa lo stile architetturale REST: il concetto di risorsa e lo sfruttamento del protocollo HTTP.

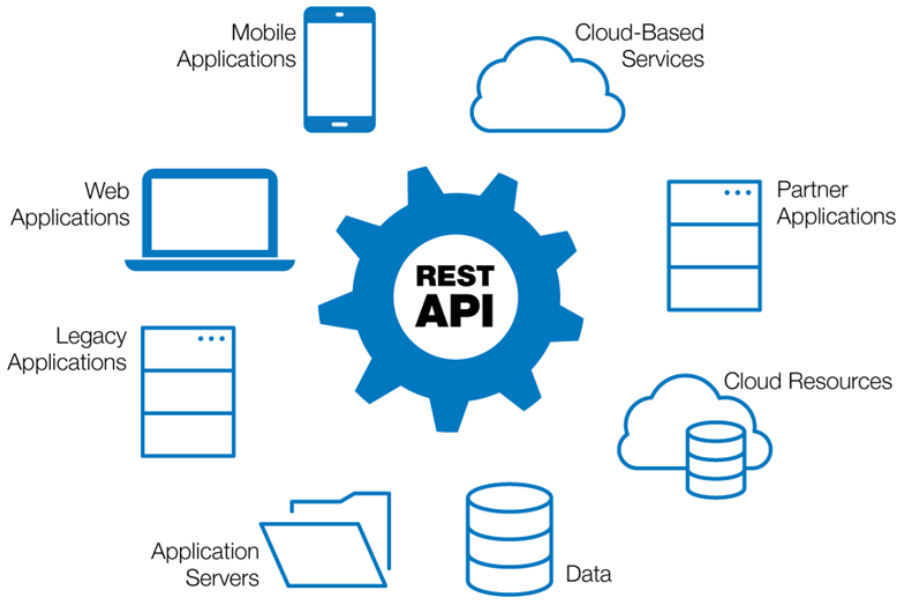


Figura 2.3: Esemplificazione delle potenzialità di REST

Insomma, REST non vuole "porsi al di sopra" del Web ma vuole sfruttare tutto ciò che è il Web.

Il nostro interesse nei confronti dell'architettura REST si manifesta quando lo andiamo a confrontare con il modello architetturale SOAP.

Come già detto, REST è uno stile architetturale che definisce pochi, ma potenti, vincoli e concetti fondamentali per lo sviluppo di API e Web services.

Iniziamo dal concetto di comunicazione Client-Server, dove vengono delineati due ruoli ben definiti nei sistemi:

- Il *Client* che è il sistema che richiede le risorse e le rielabora.
- Il *Server* che invece è il possessore delle risorse e che ha il compito di esporre per fare sì che gli vengano richieste.

A differenza di SOAP, dove i sistemi in comunicazione sono molto complessi, secondo lo stile architetturale REST, il Client risulterà molto più snello rispetto al Server, assumendosi il compito di elaborare le risorse al posto del server, riducendo il traffico di rete ed evitando di sovraccaricare il Server in caso di numerose richieste.

Il sistema deve essere “stateless”, tradotto “privo di stato”, perciò,

Il server non è tenuto a salvare informazioni riguardanti la sessione della richiesta.

Ogni nuova richiesta è quindi indipendente da quelle precedenti.

In questo caso, ogni richiesta deve contenere tutte le informazioni necessarie per poter effettuare una richiesta al Server.

Questa scelta architetturale ha come fine ultimo di aumentare la scalabilità e l’affidabilità del Server.

Il sistema deve essere in grado di supportare caching da parte dei Client.

Il Server può essere stratificato. Un Client normalmente non è in grado di determinare se è connesso direttamente con il server o se è connesso attraverso un agente intermedio come per esempio potrebbe essere proxy o un sistema di sicurezza.

I server possono temporaneamente estendere o personalizzare le funzionalità del client trasferendo del codice eseguibile.

Infine, i sistemi devono essere accessibili in modo uniforme: ogni risorsa deve avere un indirizzo univoco globale e un endpoint valido.

Questo permette un elevato grado di disaccoppiamento tra client e server, facendo sì che, avendo un’interfaccia generica, si possa avere una grande semplicità nel richiedere la risorsa e nella propagazione del servizio.

L’ultimo vincolo introduce un concetto fondamentale, il concetto di risorsa.

Una risorsa è un oggetto con un tipo, dati associati e relazioni con altre risorse e un insieme di metodi con cui interagire. È simile a un'istanza di oggetto in un linguaggio di programmazione orientato agli oggetti, con la differenza importante che per la risorsa sono definiti solo pochi metodi standard, corrispondenti ai metodi HTTP GET, POST, PUT e DELETE, secondo il modello CRUD (CREATE READ UPDATE e DELETE).

Le risorse possono essere raggruppate in collezioni. Ogni collezione è omogenea in modo che contenga solo un tipo di risorsa. Le risorse possono anche esistere al di fuori di qualsiasi raccolta. In questo caso, ci riferiamo a queste risorse come risorse singleton. Le raccolte sono anch'esse risorse. Ogni risorsa o collezione, è quindi associata ad un URI che la identifica.
Nel caso una risorsa appartenga ad un Web Service, l'URI associato dovrà perciò rappresentare un endpoint per accedervi.
Per questo l'utilizzo degli URI permette una grande frammentabilità delle risorse in caso venga sfruttato il loro grande potenziale.
Per convenzione, negli URI, le collezioni terminano con "/", mentre il nome della risorsa di interesse fa da chiusura.
Attraverso queste assunzioni possiamo creare endpoint in maniera gerarchica e logicamente costruibili, permettendo così ad un utente di potervi accedere in maniera veramente molto intuitiva.

2.4 Lo stato dell'arte di REST e delle API REST

L'espressione idiomatica "stato dell'arte" si riferisce al più alto livello di sviluppo di un campo tecnico o scientifico, realizzato in un dato momento. Va ad indicare ciò che è all'avanguardia.
Al giorno d'oggi gli sviluppatori devono creare Web services e applicazioni per dispositivi mobili sempre più leggeri e interconnessi. L'architettura REST, caratterizzata da un'ampia flessibilità, ha rapidamente guadagnato popolarità in questo ambito. Nel 2018, la maggior parte dei servizi Web pubblici hanno fornito API REST e scambiato i dati in formato JSON, non più XLM, compatto e facile da usare. Lo stesso andamento è previsto nel 2019.
Le performance di REST, più efficienti di protocolli che lo hanno preceduto, gli hanno permesso di guadagnare un incredibile popolarità nell'attuale realtà che ci circonda nella quale, in particolar modo dall'avvento degli smartphone, anche solo pochi secondi contano.

"According to Nordic APIs, REST is almost always better for web-based APIs, as it makes data available as resources (e.g. user) as opposed to services (e.g. getUser) which is how SOAP operates. Besides, REST inherits HTTP operations, meaning you can make simple

API calls using the well-known HTTP verbs like GET, POST, PUT, and DELETE." [MON18]

Parlando di tecnologie utilizzate, le API vengono sviluppate per mezzo del linguaggio Javascript. Questo linguaggio di scripting, ad oggi, è molto diffuso in quanto si integra facilmente all'interno di documenti HTML e perché permette la creazione di applicazioni Web dinamiche. In alternativa è possibile anche creare documenti Javascript, con estensione js, e collegarli successivamente.

Questo linguaggio propone una sintassi molto simile a C e C++, ciò ne semplifica l'utilizzo in quanto non necessita di compilazione ma solo di interpretazione, che verrà effettuata dal Web browser che si sta utilizzando.

Javascript, insieme ai suoi framework come Node, Express o JQuery.

Per sviluppare la nostra API REST, abbiamo utilizzato, inerentemente alle tecnologie utilizzate presso il CINECA, la API REST è stata sviluppata con il linguaggio di programmazione PHP, più precisamente utilizzando la versione 5.2.14 (vedi capitolo 4.3.1).

I messaggi sono in formato JSON, in modo da avere una chiara identificazione delle risorse e per i motivi di efficienza sopra elencati.

2.5 Alcuni esempi di reingegnerizzazione di API con REST

Dalla sua, nel 2000, ai giorni nostri REST ha cominciato ad espandersi nel mondo dello sviluppo di Web services e API a una macchia d'olio.

Molti sistemi legacy e molte Web Service SOAP presentavano, e alcuni presentano tuttora, i limiti del protocollo e dell'architettura con cui sono stati sviluppati.

Per questo motivo in questi anni è stato avviato un processo di reingegnerizzazione "globale" che potesse portare benefici a tutto il mondo del Web.

A contorno di questo processo, troviamo in letteratura anche numerosi casi di studio in cui vengono comparati Web services intenti a fornire lo stesso servizio, ma costruiti utilizzando REST e SOAP.

Il primo caso interessante in letteratura è "*Reengineering Legacy Systems with RESTful Web Service*" [LW08], nel quale gli autori hanno deciso di reingegnerizzare un Sistema legacy attraverso REST.

Dopo un breve confronto con SOAP, in cui vengono evidenziati i punti di forza del REpresentational State Transfer, il punto di interesse è ricaduto sull'argomento su cui poi si sviluppa tutto l'elaborato, ossia gli URI.

Come spiegato nel capitolo 2.3, gli URI sono probabilmente lo strumento più potente utilizzato dallo stile architetturale REST. Gli autori, infatti, ne sottolineano l'importanza in quanto i servizi Web RESTful sono determinati da quattro elementi: le risorse esposte, l'ambito, il vincolo sulle risorse e le operazioni standard. Gli URI in tutto ciò hanno il compito di portare la rappresentazione delle risorse insieme all'ambito e di esporre anche alcune regole aziendali. [LW08]

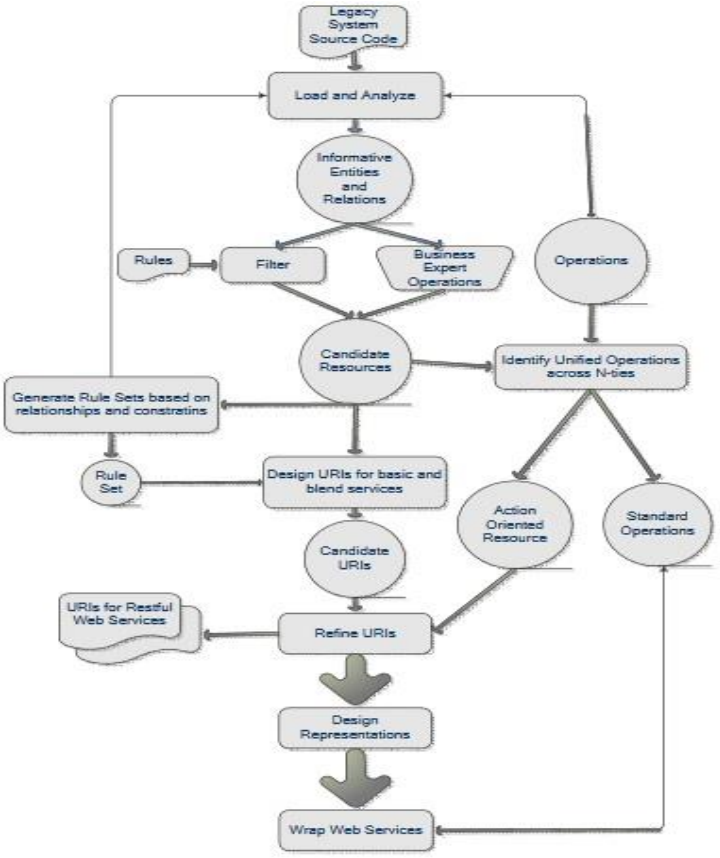


Figura 2.4: Processo di rimodellazione degli URI

Questa immagine mostra il processo di analisi del codice del sistema legacy dove il fulcro sono appunto gli URI e tutto ciò che interagisce con essi, come possono essere le risorse, elaborate attraverso un attento studio e le operazioni associate agli identificatori.

Il risultato è un giudizio sul sistema applicato che, nonostante sia risultato efficace, presenta ancora certi punti su cui approfondire, come le dipendenze tra le risorse. [LW08]

Analogamente al lavoro sopracitato troviamo un altro lavoro analogo del 2011, intitolato "*Migration of SOAP-based Services to RESTful Services"* di Upadhyaya e altri. Gli autori partono dagli stessi presupposti di quelli precedentemente osservati, ossia che i servizi RESTful non includono standard complessi e operazioni eterogenee e quindi sono più semplici da consumare e comporre comparati ai servizi Web basati su SOAP. [UZ11]

La metodologia con cui vengono rielaborati questi Web services SOAP e la sua descrizione tramite WSDL (Web Service Description Language), analogamente agli altri lavori analizzati, passa attraverso gli URI, le risorse che il servizio ha intenzione di esporre e le operazioni associate ad esse.

"Each resource in a RESTful service is associated with four operations (i.e., GET, PUT, POST and DELETE). "We need to identify the operations that manipulate the same resource. We build a dependency graph which connects input and output parameters of each operation along with the semantics (i.e., nouns found in an operation-name) of an operation. A circle denotes an operation. A square represents an input or an output parameter. A triangle means a noun in the name of an operation. For the operations and parameters, the edges pointing to the operations denote message input and the edges pointing to the parameters represent the message output." [UZ11]

Il trafiletto spiega la metodologia con cui vengono identificate le risorse, standardizzando il processo della strutturazione delle operazioni, il loro nome e le risorse che vanno a identificare.

Il grafo delle dipendenze è costruito attraverso una serie di equazioni utili a identificare le risorse con elementi in comune come, appunto, le operazioni o i campi.

Questa metodologia viene applicata ad un caso di studio che prende in esame 713 Web services, di cui, come si può ben notare, la maggior parte è sviluppata con lo stile architetturale REST. Un dato importante, reso noto da questa tabella, è che negli ambiti più importanti la maggioranza dei Web service utilizzi REST già nel 2011.

Category	RESTful Services	SOAP-Services
Internet	160	55
Shopping	97	17
Search	84	11
Financial	64	57
Enterprise	74	29
Government	61	4
Total	540	173

Tabella 2.1: Tipologia dei Web services analizzati.

I risultati nella seguente tabella mostrano la precisione di questo sistema di rielaborazione applicato ai WSDL dei servizi SOAP.

Table 7: Results of Identifying Resources from WSDL

Category	# Methods	#Predicted Resources	#misidentified Resources	#total Resources	Precision	Recall
Finance	59	35	8	40	0.77	0.67
Government	39	25	2	27	0.92	0.85
Travel/Tourism	132	97	16	110	0.83	0.73
Ecommerce	102	80	14	90	0.82	0.73
Others	78	47	6	53	0.87	0.77

Tabella 2.2: Risultati della rielaborazione su sistemi SOAP.

La precisione nell'identificazione delle risorse vanta una precisione dal 77% al 92%, considerando un successo il sistema di rielaborazione.
I risultati Sui WSDL dei Web service REST, invece, si basano su una semplice analisi di identificazione delle risorse già presenti, verificando se combaciano.
Il risultato è una precisione media del 74% e, negli anni futuri, si punterà ad aumentare l'efficacia della metodologia. [UZ11c]

Sempre nella letteratura pubblicata nel corso del 2011, troviamo un caso in cui viene somministrato un progetto ad una classe di sessantanove studenti di *computer Science.*
Gli studenti vengono incaricati di sviluppare due progetti con lo scopo di implementare un Web service collegato a due sensori, uno con lo stile architetturale REST e uno con lo stile SOAP. Vengono successivamente raccolti, attraverso un questionario, i pareri degli studenti in cui esprimono per cosa, secondo loro, una tecnologia funziona meglio e per quali scopi.
Il lavoro in questione è: "*In Search of an Internet of Things Service Architecture: REST or WS-*? A Developers' Perspective"* [GI11] è stato svolto presso l'Istituto per l'elaborazione Pervasiva ETH di Zurigo, Svizzera.
I ricercatori partono da un tema che interessa tutto il Web of Things: l'interoperabilità tra oggetti intelligenti e standard e applicazioni esistenti. Da qui esplicitano i due approcci orientati ai servizi nella ricerca e sviluppo, REST e WS (WSDL+SOAP). Il loro scopo è integrare gli studi esistenti sulle metriche delle prestazioni con una valutazione delle preferenze degli sviluppatori e delle esperienze di programmazione riguardo questi due servizi, in modo da rendere più consapevole la scelta. [GI11]

I risultati che ottengono nel contesto dello studio condotto mostrano che REST si distingue come l'architettura di servizio preferita.
Per rendere più chiaro come sono arrivati a questa conclusione riassumiamo i criteri utilizzati dagli sviluppatori nella tabella seguente.

Requirement	REST	WS-*	Justification
Mobile & Embedded	+	-	Lightweight, IP/HTTP support
Ease of use	++	-	Easy to learn
Foster third-party adoption	++	-	Easy to prototype
Scalability	++	+	Web mechanisms
Web integration	+++	+	Web is RESTful
Business	+	++	QoS & security
Service contracts	+	++	WSDL
Adv. security	-	+++	WS-Security

Table 4. Guidelines for choosing a service architecture for IoT platforms.

Tabella 2.3: Preferenze e per quale motivo.

Come nel lavoro precedente, le preferenze nel business e nell'ambito della sicurezza ricadono sul protocollo SOAP, in quanto percepito più sicuro, anche per la minore apertura nei confronti di altri Web services.

Avvicinandosi sempre di più ai giorni nostri, nel 2013, nel lavoro intitolato "*Model-Driven Generation of a REST API from a Legacy Web Application*" [RM13], viene proposta un'altra soluzione per reingegnerizzare automaticamente sistemi legacy.
Il progetto sviluppato dai ricercatori è stato denominato MIGRARIA e definisce un processo sistematico e semiautomatico per la modernizzazione delle applicazioni Web legacy basate su dati. Questo processo inizia con la generazione di una rappresentazione concettuale del sistema legacy attraverso l'utilizzo di tecniche di reverse engineering. La rappresentazione concettuale si basa sul meta-modello MIGRARIA MVC, che consente di specificare le applicazioni Web legacy sviluppate mediante framework Web basati su MVC e successivamente va a specificare l'insieme di componenti o sottosistemi da modernizzare.
Lo scopo principale degli autori è quello di integrare il sistema con la generazione di nuovi modi di accesso e interazione come un client RIA e il relativo livello di connessione al server, un'API REST in questo caso. [RM13]

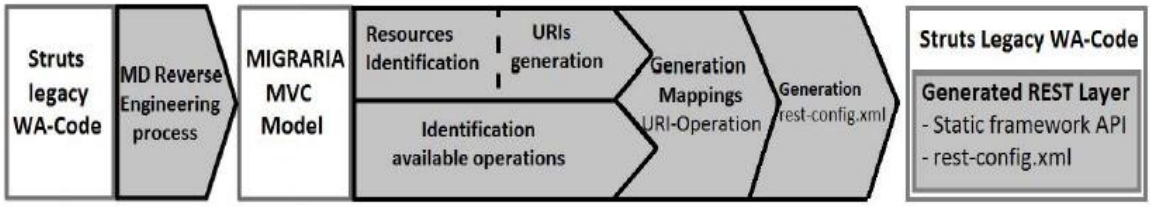


Figura 2.4: Rappresentazione del progetto MIGRARIA.

Questo lavoro, come illustrato nella figura precedente, prende in esame una parte del processo di modernizzazione definito all'interno della MIGRARIA, progetto che affronta l'evoluzione della presentazione di un Web legacy system verso un nuovo client RIA Web 2.0.

Allo scopo di rendere più chiara questa asserzione rendiamo noto che RIA (Rich Internet Application) è un'applicazione Web aventi le funzionalità delle applicazioni desktop, senza però necessitare dell'installazione sul disco fisso, mentre il Web 2.0 è caratterizzato da una maggiore interazione rispetto al Web 1.0.

Per questo motivo la grande diffusione di REST nella riprogettazione si rende necessaria, in quanto aumenta l'interoperabilità tra i servizi, la frammentabilità e la ridotta dimensione dei sistemi e dei messaggi.

Avendo descritto il lungo e in largo la popolarità di REST nel mondo del Web, andiamo ora a focalizzarci sul perché di questa popolarità.

Nel lavoro *"Performance Analysis of SOAP and RESTful Mobile Web Services in Cloud Environment"* [D, del 2014, viene messa a confronto l'efficienza di un sistema SOAP a quella di un sistema REST.

"*Authors have developed the frameworks for provisioning web services on Mobile phone which is suitable for both SOAP and REST. They have compared both frameworks and did various experiments in which they found that REST based framework is more suitable for handheld, resource constrained mobile device and wireless network. They also found that the level of resource consumption is less in REST as compared to SOAP. But SOAP framework is more secure as compared to REST*." [DS14]

Questi autori per ottenere dei dati rilevanti hanno eseguito e raccolto i risultati del servizio web REST in Apache Tomcat 7.0.

SN	SOAP		REST	
	Size of requested data(in Bytes)	Response time (in ms)	Size of requested data(in Bytes)	Response time (in ms)
1	422	500	54	121
2	424	521	50	118
3	423	520	49	115
4	425	528	49	115

Tabella 2.5: Confronto tra Web services SOAP e REST in relazione al tempo di risposta sull'host locale Apache Tomcat (richiesta HTTP GET).

In seguito, hanno anche raccolto i risultati di REST e dei Web services SOAP su Google App Engine.

SN	SOAP			REST		
	Size of requested data (in Bytes)	Response time (in ms)	CPM_MS	Size of requested data(in Bytes)	Response time (in ms)	CPM_MS
1	330	678	421	39	68	23
2	321	639	398	42	53	47
3	323	643	410	54	69	23
4	325	651	430	58	73	21

Tabella 2.6: Confronto tra i dati delle risposte del Web services SOAP e di REST.

Entrambi i servizi sono stati testati sul server Apache Tomcat e sul server cloud Google App Engine. Le applicazioni mobili (client di servizio) vengono implementate utilizzando Android Developer Tool.

Dai dati raccolti è chiaro che per una dimensione inferiore dei messaggi la risposta e il tempo di trasmissione sono inferiori, il che comporta un consumo di energia inferiore e un Web service più veloce. Questo soddisferà i vincoli fisici dei dispositivi mobili e

raggiungerà la qualità nell'obiettivo del servizio. Ciò dimostra che il servizio Web RESTful è migliore per i dispositivi mobili. [DS14]

Per finire questa analisi della letteratura nel corso degli anni, presentiamo una reingegnerizzazione descritta in un articolo del 2017 nel quale gli autori dimostrano l'efficienza della tecnologia REST applicata ad un sistema con numerosi utenti, nel loro caso un gestore dell'industria cantieristica chiamato BOM (Bill of materials). La motivazione dietro al loro lavoro, oltre verificare le performance di REST, è stata quella di trovare una soluzione alla realtà di fronte alla quale si trovavano. Una realtà in cui la maggior parte delle applicazioni di database per le imprese tradizionali si basano su due livelli con client-server semplice o su tre livelli con middleware RPC aggiuntivo.
Questa configurazione obsoleta ha la caratteristica di non essere estensibile e rende perciò necessaria un'alternativa per collegare il database al Web o alla piattaforma mobile. [JC17]
Dopo la reingegnerizzazione, conforme allo stato dell'arte, le varie componenti dei nuovi Web services REST sono stati i seguenti:

Before	After
Database	DB + Web Application Server
SQL code	Grails ORM code
OLEDB (ADO.Net)	JDBC
Windows Clients	REST Clients (Web browsers)
Deployment Server	Web Application Server
FTP protocol (for client updates)	HTTP or HTTPS protocol

Tabella 2.8: Tecnologie utilizzate nel two-tier system

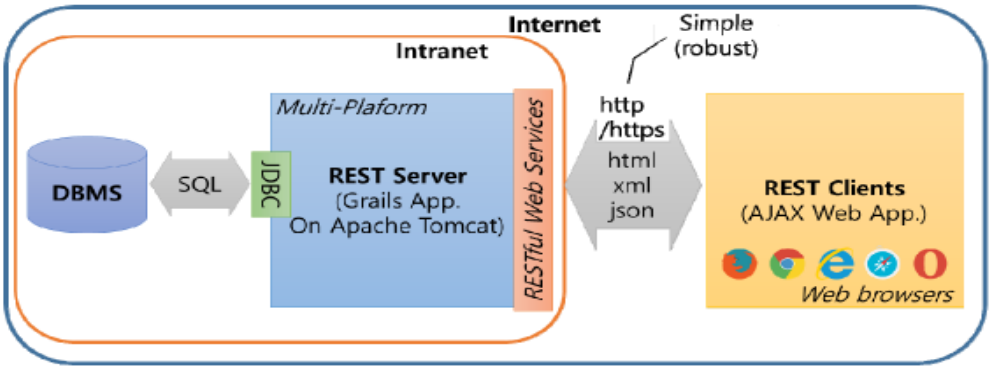
Nell'immagine precedente vengono comparate le tecnologie utilizzate prima della rielaborazione attraverso REST e dopo nei *sistemi two-tier*.

Nell'immagine successiva viene invece illustrato il cambiamento di tecnologie nei sistemi *three-tier*.

Before	After
Database	Database
Native DB Client library	JDBC
SQL code	Grails ORM code
Middleware Server	Web Application Server
C language (middleware code)	Groovy or Java Code
RPC protocol (DCE)	RESTful web service (http)
Binary Message	JSON Message
Windows Clients	REST Clients (Web browsers)
Deployment Server	Web Application Server
FTP protocol (for client updates)	HTTP or HTTPS protocol

Tabella 2.9: Tecnologie utilizzate nel three-tier system

Nonostante la riprogettazione dei *two-tier system* e *three-tier system* li rendano molto più efficienti, un Web service progettato "da zero" risulta molto più semplice a livello di struttura,
come verrà illustrato nell'immagine successiva.



Tab 2.10: Sistema progettato conforme allo stato dell'arte.

I risultati di questo caso sono molto simili agli obiettivi che stanno alla base del progetto di questa tesi, ossia: un miglioramento dell'efficienza della manutenzione dell'applicazione dell'approccio a tre livelli legacy con l'adozione del servizio web RESTful e del moderno framework di applicazioni web; un aggiornamento delle colonne di una tabella in un'applicazione Web Grails con servizi Web RESTful. Come nel nostro progetto, questo caso di studio ha esaminato i problemi strutturali di una realtà governata da applicazioni legacy, come possono essere quelle utilizzate nel settore navale o dell'università e della ricerca, e ha proposto un prototipo basato su REST in alternativa.

Capitolo 3

Il progetto REST-AURATION

3.1 CINECA

Il CINECA è un consorzio interuniversitario italiano senza scopo di lucro istituito nel 1967 dalla collaborazione del MIUR (Ministero dell'Istruzione, dell'Università e della Ricerca) e quattro Rettori, precisamente dell'Ateneo di Bologna, Venezia, Padova e Firenze presso il rettorato dell'Alma Mater Studiorum.

La necessità di strumenti per il supercalcolo scientifico spinse, verso la fine degli anni 60', alla collaborazione tra gli enti sopracitati in quanto un singolo Ateneo non sarebbe stato in grado di sostenere le spese a causa dei fondi non sufficienti e soprattutto per la mancanza di strutture adeguate.

Viene quindi siglata una convenzione che nel 1969 portò all'inaugurazione del *Consorzio Interuniversitario per il Calcolo Automatico dell'Italia Nord Orientale* a Casalecchio di Reno in provincia di Bologna.

Alla struttura in questione nel corso degli anni si sono aggiunte le sedi di Milano, Roma e Napoli e conta attualmente circa 700 dipendenti.

Oggi il CINECA è il maggiore centro di calcolo in Italia e uno dei più importanti a livello mondiale nonché il punto di riferimento per il sistema accademico nazionale e, grazie all'esperienza acquisita in oltre trent'anni di attività, ha assunto un ruolo di assoluta importanza nell'ambito della Pubblica Amministrazione.

CINECA offre supporto alle attività di ricerca della comunità scientifica tramite il supercalcolo e le sue applicazioni, grazie a un ambiente di calcolo al massimo livello delle architetture e delle tecnologie disponibili. Infatti, fin dalla sua costituzione vanta di sistemi all'avanguardia per il supercalcolo: nel 1969 viene installato nella sua sede il primo supercomputer in Italia, un *CDC 6600*, allora il più potente computer al mondo disponibile sul mercato; negli anni successivi il CINECA continua ad acquisire elaboratori di altissima potenza, come il *Cray X-MP12*, il primo supercomputer vettoriale installato in Italia nel 1985 e il supercomputer *Fermi* nel 2012, basato su architettura Blue Gene/Q,

grazie al quale raggiunge la settima posizione nella lista dei 500 supercomputer più potenti al mondo; l'ultima acquisizione è il supercomputer *Marconi*, il diciannovesimo computer più potente al mondo. [BAS19]

Il suo personale specializzato è altamente qualificato e affianca i ricercatori nell'utilizzo dell'infrastruttura tecnologica, sia in ambito accademico che industriale.

Il Consorzio si occupa di progettare e sviluppare sistemi informativi per la pubblica amministrazione, la sanità e le imprese. Tra le sue mansioni principali troviamo la realizzazione di sistemi gestionali e servizi a sostegno delle università e del MIUR, tanto da diventare il suo "braccio tecnico operativo". In questo ruolo è impegnato nella costante ricerca di soluzioni in grado di accompagnare gli Atenei nel loro cammino di ammodernamento dei metodi di governo di realtà così complesse e articolate.

Tra i tanti servizi gestionali svolti su indicazione del MIUR, il CINECA, insieme ad ANVUR (Agenzia Nazionale Di Valutazione Del Sistema Universitario e Della Ricerca), si impegna nell'azione di monitoraggio dei flussi informativi volti a far pervenire proprio ad ANVUR i dati necessari per la Valutazione della Qualità della Ricerca.

Il principale strumento utilizzato per questo scopo è la Scheda Unica Annuale del Corso di Studio, che andremo ad approfondire nel capitolo seguente. [CIN19]

3.2 La Scheda Unica Annuale del Corso di Studio (SUA-CdS)

Questo strumento è di fondamentale importanza in quanto fornisce, a chiunque ne sia interessato, informazioni utili come: gli obiettivi della formazione; la descrizione del piano formativo e dell'ambiente di apprendimento; i possibili sviluppi nella carriera successiva al completamento del Corso di Studio.

La SUA-CdS è considerato lo strumento di riferimento nel processo di autovalutazione, valutazione dei Corsi di Studio.

La sua adozione costituisce uno dei requisiti di sistema per l'*Assicurazione della Qualità della formazione*.

Secondo ANVUR: "*Il sistema AVA (Autovalutazione – Valutazione periodica – Accreditamento) ha l'obiettivo di migliorare la qualità della didattica e della ricerca svolte negli Atenei, attraverso l'applicazione di un modello di Assicurazione della Qualità (AQ) fondato su procedure interne di progettazione, gestione, autovalutazione e*

miglioramento delle attività formative e scientifiche e su una verifica esterna effettuata in modo chiaro e trasparente.
La verifica si traduce in un giudizio di Accreditamento, esito di un processo attraverso il quale vengono riconosciuti a un Ateneo (e ai suoi Corsi di Studio) il possesso (Accreditamento iniziale) o la permanenza (Accreditamento periodico) dei Requisiti di Qualità che lo rendono idoneo allo svolgimento delle proprie funzioni istituzionali." [ANV19]

Il modello SUA-CdS deve essere compilata e/o aggiornata entro il 30 maggio di ogni anno da parte del personale incaricato dall'Ateneo e si compone di due sezioni:

1. "Qualità", rappresenta uno degli strumenti principali di "Autovalutazione, Valutazione periodica e Accreditamento" (AVA) introdotto dalla L. 240/2010, dal D.lgs. 19/2012 e recepito dal nuovo D.M. 6/2019.
   Raccoglie ogni informazione utile per far conoscere: la domanda di formazione; il percorso formativo; i risultati di apprendimento attesi; i ruoli e le responsabilità che riguardano la gestione del sistema di assicurazione della qualità del Corso; i presupposti per il riesame periodico dell'impianto del Corso di Studio e le eventuali correzioni individuate.
2. "Amministrazione" che raccoglie i dati di istituzione (RAD) e di attivazione pregressa (OFF.F) del Corso.
   RAD è l'acronimo di "Regolamento Didattico di Ateneo" e viene utilizzato per indicare la sezione della banca dati ministeriale inerente all'offerta formativa.
   OFF.F invece è un sito di consultazione, ancora consultabile, che raccoglie le informazioni sui Corsi di Studio di laurea e laurea magistrale fino al 2012.
   Questa sezione consente una migrazione verso il nuovo sistema di gestione delle informazioni che viene a costituirsi come una piattaforma di comunicazione "integrata" che consente di veicolare verso tutti i destinatari del processo di comunicazione la medesima informazione. [MIU19]

Come appena illustrato, la SUA-CdS è composta da numerosi testi. Propongo un esempio fittizio ma verosimile, il testo del Corso di Studi di una laurea triennale in "Scienze Proibite" presso l'Ateneo di "Mordor".
Verrà mostrato il testo che come titolo identificativo ha "Il Corso di studi in breve".

```
Obiettivi del Corso di Studio
Le tecnologie proibite e gli orchi fanno ormai parte di tutto ciò che
tocca la vita delle persone. Capire le diverse dimensioni a cui si può
applicare la magia, risolvere problemi che richiedono una preparazione
tecnico-scientifica, stare al passo con l'innovazione e contribuire
all'evoluzione della tecnologia sono i motivi per studiare Scienze
Proibite a Mordor. Il Corso sviluppa le conoscenze necessarie alla
risoluzione di problemi complessi: l'utilizzo di tecniche e linguaggi
orcheschi, la progettazione di armi, l'architettura e le prestazioni
dei sistemi di alambicchi, la progettazione di sistemi proibiti, di
applicazioni alchemiche e mistiche, la realizzazione di sistemi di
comunicazione e riti di morte. La solida preparazione scientifico-
metodologica permette di sperimentare la propria creatività e di
acquisire competenze rapidamente spendibili nel mondo del lavoro.
Entrare nel mondo del lavoro l'orco è una figura professionale
specializzata che trova collocazione in aziende ed enti pubblici per
mansioni ad alto contenuto di guerra, legate soprattutto al trattamento
e alla trasmissione degli incantesimi (Magic and Alchemical
Technologies):
-     Progettista di software e sistemi magici integrati
-     Analista e programmatore di sistemi orcheschi
-     Consulente per l'innovazione mistica
-     Amministratore di sistemi, di reti di fabbricazione armi
-     Manager dell'innovazione
Proseguire gli studi
Il Corso consente l'accesso al corso di Laurea Magistrale in Maestria
Jedi, ad altri corsi di laurea magistrale o a master di primo
livello.
Descrizione link: Presentazione del Corso di Studio
Link inserito:
https://corsi.unimord.it/laurea/scienzeproibite/esplora-il-corso
```

Listing 3.1: Esempio di possibili testi della SUA-CdS

Un problema di non poca rilevanza è il fatto che questi testi siano visibili al pubblico solamente attraverso il portale Universitaly (https://www.universitaly.it), limitando l'accessibilità e la possibilità di trovare queste informazioni su altre piattaforme.

Universitaly è il portale online del MIUR, creato con lo scopo di guidare gli studenti nello scoprire l'offerta formativa di ciascuna università italiana e i requisiti di accesso ad esse; ottenere linee guida utili all'orientamento post diploma e iscriversi ai test d'ingresso dei corsi di laurea ad accesso programmato nazionale (come ad esempio Medicina).

Andiamo a spiegare ora l'architettura che sta dietro al portale Universitaly: un Client espone le risorse, che vengono prima elaborate e spedite da un Server, dopo averle ricevute da un altro Server, che le ha estratte da un database ed elaborato il messaggio.
Il problema sopracitato ha come conseguenza primaria la non possibilità di seguire i principi dell'"Italian Open Data License" (IODL). Questo è un contratto di licenza che ha lo scopo di consentire agli utenti di condividere, modificare, usare e riusare liberamente la banca di dati, i dati e le informazioni con essa rilasciati, garantendo la stessa libertà ad altri. Lo scopo di IODL è facilitare il riutilizzo delle informazioni pubbliche nel contesto dello sviluppo della società dell'informazione. L'esclusività del mezzo di esposizione delle risorse va a limitare questa libertà ai possibili utenti.
Recentemente è stata aggiornata versione 1.0 di questa licenza.
"La pubblicazione della IODL 2.0 arriva dopo una discussione pubblica e aperta a tutti, svoltasi on line nel gruppo dedicato su InnovatoriPA dove dipendenti pubblici, avvocati, rappresentanti della comunità Open Data si sono confrontati; nel corso della stesura della nuova versione (alla quale ho avuto il piacere e l'onore di contribuire) abbiamo concordato sulla necessità di elaborare una nuova licenza che avesse le seguenti caratteristiche:

1. *maggiore semplicità (sia per le Amministrazioni che per gli utenti);*
2. *minori oneri per gli utenti (ai quali viene solo chiesto di citare la fonte del dato);*
3. *stimolo al riutilizzo da parte di soggetti privati."* [BEL12]

La SUA-CdS e la sua esposizione si sviluppano, a livello implementativo, nella seguente modalità.
I testi, racchiusi in un database Oracle avente la seguente struttura:
Esiste una tabella madre chiamata Z_SUA contenente i Corsi di studio e ha come *chiave primaria* l'*id_rad* (l'identificativo univoco con cui viene individuato il Corso di studi all'interno dei database).
A questa tabella sono collegate altre tabelle, precisamente una per ogni testo della SUA-CdS.
Ognuna di queste a sua volta ha connesse tabelle contenenti gli allegati dei testi.
L'immagine seguente rappresenta lo schema ER (Entità-Relazione) del database contenente la SUA-CdS.

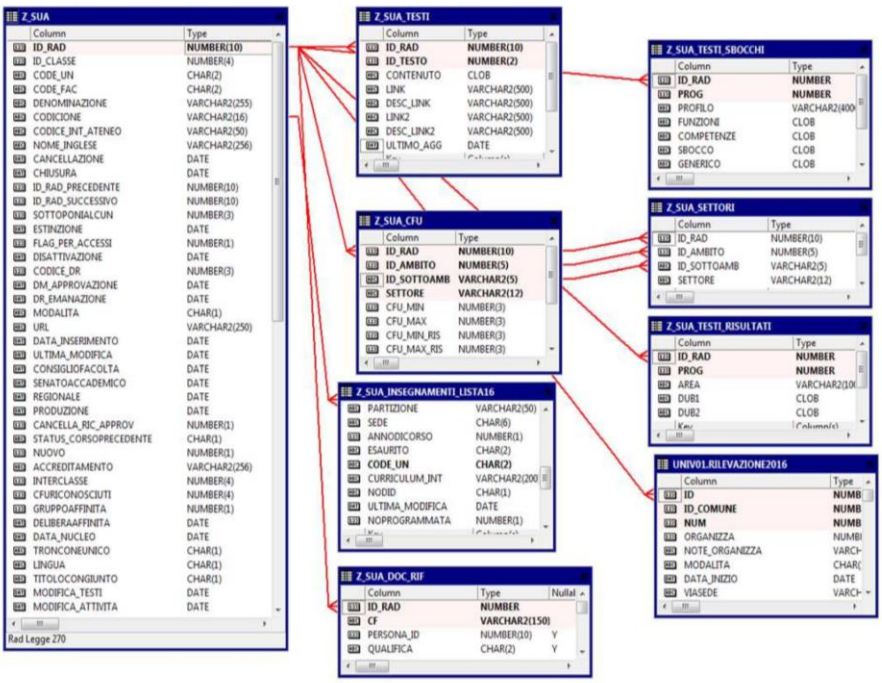


Figura 3.1: Schema ER del database contenente i testi della SUA-CdS.

I testi, per il momento, sono dati "grezzi" che vengono poi elaborati attraverso un server sviluppato con un'architettura SOAP (vedi capitolo 2.2), e poi inviati al richiedente, in questo caso il portale Universitaly, che li espone a suo piacimento ai browser che lo interrogano.

Il protocollo di trasporto SOAP, come già detto, è il problema più grande da affrontare.

I messaggi scambiati tra client e server sono in formato XML. Come potrete osservare nelle righe successive, il codice e i tag propri del formato rendono ad uno sviluppatore difficile manipolare il messaggio senza una formattazione data dal client e, soprattutto, contribuiscono a creare messaggi con un *overhead* rilevante. In altre parole, i messaggi sono molto pesanti.

Di seguito verrà mostrato un esempio di messaggio in formato XML contenente i primi due testi della SUA-CdS del Corso di Studio fittizio di "Scienze dei Beni Oscuri" dell'università di "Mordor".

```
<?xml version="1.0" encoding="UTF-8"?>
<ELENCO><SUA><ID_SUA>1550868</ID_SUA><ANNO_VALIDITA>2019</ANNO_VALIDI
TA><CODE_UN>02</CODE_UN><CLASSE>2001</CLASSE><NOME_CLASSE>L-1 -
Scienze
Proibite</NOME_CLASSE><CODICIONE>0720106200100003</CODICIONE><ID_RAD>
1389843</ID_RAD><ID_RAD_PREC>1323992</ID_RAD_PREC><ID_RAD_SUCC/><ATEN
EI_CONVENZIONE/><TITOLO_CONGIUNTO/><ATTIVAZIONE/><NOME_CORSO>SCIENZE
DEI BENI OSCURI</NOME_CORSO><NOME_CORSO_IN>Sciences for Dark
```

```
Heritage</NOME_CORSO_IN><NOME_TO_VIS/><CODICE_INTERNO>7312^2019^PDS0-
2019^1006</CODICE_INTERNO><LINGUA_CORSO>I</LINGUA_CORSO><NUOVO_CORSO>
1</NUOVO_CORSO><CORSI_509/><FAC_RIF><NOME_FAC/></FAC_RIF><STRUTTURA_R
IF ID_STRU="19660"><NOME_STRU>Studi
Proibiti(DISUM)</NOME_STRU></STRUTTURA_RIF><ALTRE_STRU/><ALTRE_FACOLT
A/><DATA_CONSIGLIO>08/04/2019</DATA_CONSIGLIO><DATA_SENATO>11/04/2019
</DATA_SENATO><DATA_NUCLEO>25/02/2013</DATA_NUCLEO><DATA_CONSULTA>18/
01/2008</DATA_CONSULTA><DATA_CONSULTA2>19/02/2015</DATA_CONSULTA2><MO
DALITA_CORSO>C</MODALITA_CORSO><SITO_CORSO>https://manageweb.ict.unim
ord.it/ricerca/dipartimenti/disum/offerta-formativa/didattica/scheda-
corso-scienze-beni-
oscuri</SITO_CORSO><MAX_CFU>12</MAX_CFU><GRUPPI_AFF>1</GRUPPI_AFF><UL
TIMO_RAD_CUN>1389843</ULTIMO_RAD_CUN><TESTI><CONTENUTO
ID_TESTO="REL_NUC">
<TESTO>Scienze dei beni oscuri (cod off=1323992)
L'Ateneo presenta nella stessa classe il corso di Scienze dei beni
oscuri per il turismo. [...]
Sempre nell'ambito delle attività affini è stato inserito il SSD
GEO/09 - Georisorse minerarie e applicazioni mineralogiche-
petrografiche per l'ambiente e i beni oscuri. Tali modifiche non
risultano motivate. [...]
Si segnala che il valore massimo di CFU previsto nell'ordinamento
precedente per le attività affini risulta aumentato di 6 CFU con il
conseguente aumento dei CFU totali. Alla luce di quanto sopra, il NVA
esprime parere favorevole sulla proposta.
</TESTO><LINK/><TITOLO>Sintesi della relazione tecnica del nucleo di
valutazione</TITOLO><NOTA/><UPLOAD>0</UPLOAD></CONTENUTO><CONTENUTO
ID_TESTO="PAR_PARTI_SOC">
<TESTO>Negli ultimi anni (2015-2018) il Corso di Studio ha tenuto
rapporti con Istituzioni ed Enti pubblici e privati attivi nel campo
dei beni oscuri (Soprintendenze, Archivi e Biblioteche, Musei,
Confindustria, imprese attive nel campo dell'editoria specializzata e
dell'alchimia applicata ai beni oscuri). [...]
Nel 2016 sono stati organizzati incontri presso le strutture che
hanno ospitato negli ultimi anni gli stagisti, per monitorare
l'esperienza formativa in loco e di esaminare con i responsabili la
preparazione degli studenti e il profilo professionale previsto dai
tre corsi, in relazione alle esigenze del mondo del lavoro. [...]
Infine, il 30 gennaio 2018 è stato organizzato un incontro con le
parti sociali (v. verbale allegato) che ha evidenziato la necessità
di potenziare alcune competenze e alcuni ambiti disciplinari, per una
più efficace applicazione delle conoscenze teoriche all'esperienza
della magia oscura. </TESTO><LINK/> [...].
```

Listing 3.2: Esempi di possibili testi della SUA-CdS in formato XML

3.3 REST-AURATION

Il Progetto REST-AURATION nasce dal tirocinio curriculare svolto dal sottoscritto presso il CINECA, più nello specifico presso l'ufficio operante per il MIUR. Come già accennato, l'applicazione SUA-CdS presenta numerosi problemi. Tra quelli che andremo ad analizzare ci sono l'architettura SOAP con cui è stata progettata; la riusabilità del servizio e, negli ultimi anni, la perdita di accessi attraverso quello che per ora è l'unico portale di accesso alle informazioni, ossia Universitaly.

L'obiettivo del tirocinio e la motivazione per cui è stata redatta questa tesi è appunto quello di andare a "ri-valorizzare" questo strumento così importante, da qui la scelta del nome REST-AURATION

In una fase iniziale si è presa la decisione di utilizzare SPLUNK per capire a quale realtà ci trovavamo di fronte. Questo è un software nato quindici anni fa per l'analisi dei Big Data aziendali ottenuti da qualsiasi sorgente, ad esempio dai log dei server, dai click, dai sensori o, semplicemente, dal traffico di rete. Nonostante abbia un compito altamente specializzato e circoscritto alla raccolta dei dati è diventato, con il tempo, il leader di mercato in questo campo, tanto da contare al giorno d'oggi oltre quattromila clienti.

"Splunk's search language, lookups, macros, and subsearches reduce hours of tedium to seconds of simplicity, and its tags, saved searches, and dashboards offer both operational insights and collaborative vehicles. [...] Splunk provides both compelling features for today, and a platform for exploring new methods for tomorrow." [SC10]

Nel nostro caso, SPLUNK è stato utilizzato per visualizzare il numero di accessi alla sezione inerente alla SUA-CdS del Corso di Studio di Informatica presso l'Ateneo di Bologna all'interno del portale Universitaly, per verificare l'ipotesi della perdita di appetibilità del servizio.

Per effettuare una stima complessiva e comparare dati provenienti da anni diversi, si è deciso di prendere in esame i dati dal 2015 al 2017. Infatti, avendo effettuato questo studio nei primi mesi del 2019, la SUA-CdS valida per l'anno accademico 2018/2019 si trovava ancora ad avere qualche mese di validità, perciò gli accessi sarebbero stati sicuramente minori e quindi non un caso valido di esame.

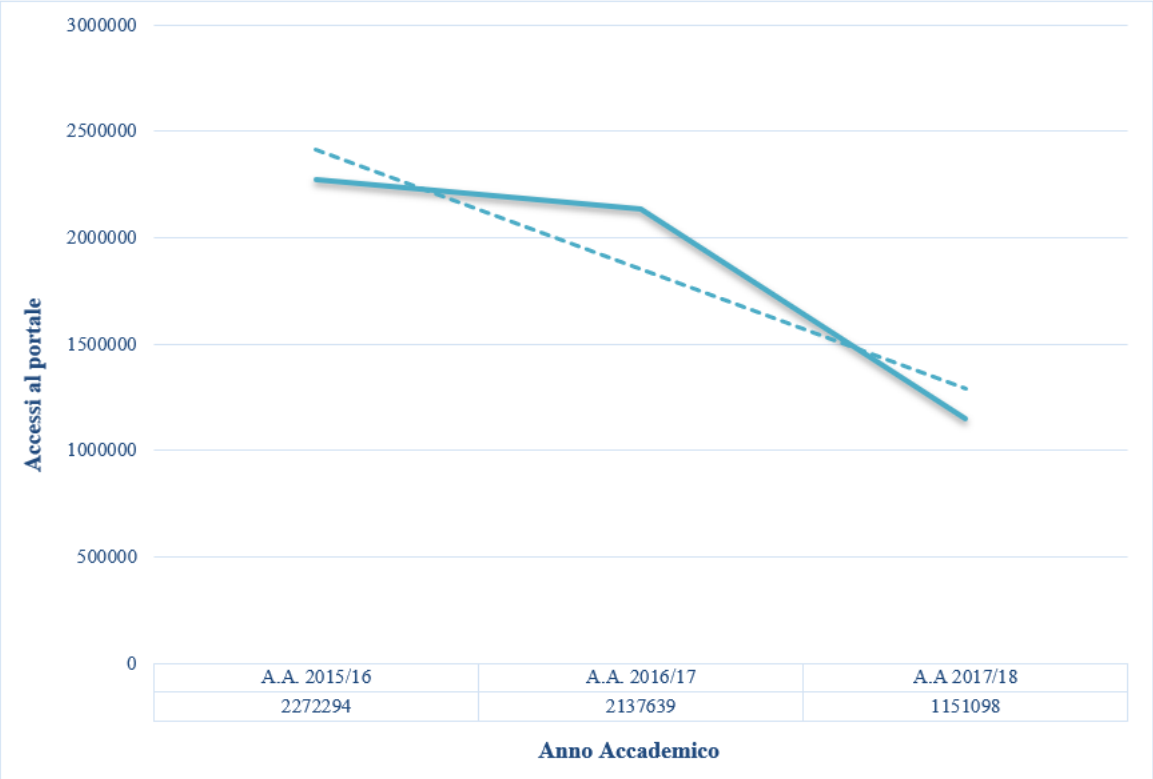


Grafico 3.1: Illustrazione del calo degli accessi

I risultati hanno rivelato che il numero di accessi al portale presa in esame sono calati drasticamente e stanno continuando a farlo. Ciò va a confermare la nostra ipotesi di partenza.

Tornando a parlare del progetto REST-AURATION, nel periodo svolto presso il CINECA nelle vesti di tirocinante e, per necessità di tempo nei mesi successivi, è stato molto semplice decidere di sviluppare questa API sfruttando le caratteristiche REST e tutte le sue potenzialità.

Questo ci permette di ampliare le funzionalità della Scheda Unica Annuale del Corso di Studio in termini di esposizione, sfruttando al massimo il concetto di risorsa e l'espressività degli URI.

Gli URI di REST-AURATION, come illustrato nelle specifiche di REST, seguono una logica gerarchica che va dall'ambito più generale a quello più specifico.

Questa logica segue la struttura degli Atenei italiani, partendo da questi e per arrivare fino alle informazioni vere e proprie dei Corsi di Studio.

Ovviamente ciò porta altri benefici come, ad esempio, una grande frammentabilità delle risorse, tutte accessibili attraverso il sistema degli URI che permette la loro

identificazione, garantendo messaggi di piccole dimensioni che non risultano ingombranti quando vengono scambiati attraverso la rete.

La gerarchia prevede che all'apice dell'URI vi sia ciò che "racchiude tutto il resto", la collezione di tutti gli Atenei nazionali.

Successivamente, con l'aggiunta di campi, avremo la possibilità di addentrarci fino a scegliere un determinato Corso di Studio e di muoverci all'interno di risorse e informazioni molto differenti tra loro.

Seguendo questi principi si avrebbe anche la possibilità di riutilizzare le risorse esposte per creare API Web esterne con finalità molto diverse oppure di sviluppare "nuovi mashup" di tanti servizi offerti da altre API e Web services.

REST-AURATION espone una quantità importante di nuove informazioni utili che non saranno più solamente date dai testi della SUA-CdS dell'anno accademico corrente, in quanto si amplieranno le possibilità e verrà data l'opportunità di avere una visione di come è cambiato il Corso di studi nel tempo, fino all'anno accademico attuale.

Di tutte le tipologie di informazioni sfruttabili, che andremo in seguito ad approfondire, con la nuova SUA-CdS si ha la possibilità di selezionare un anno di specifico interesse nell'intervallo dal 2013 a oggi.

Ad esempio, ora si può prendere visione di tutti i Corsi di Studio attivi presso qualunque Ateneo italiano nell'Anno Accademico 2015/16.

Rispetto all'applicazione SUA-CdS ancora vigente, sono stati aggiunti numerosi endpoint (uno per ogni URI identificativo di una risorsa), grazie ai quali riusciamo ad ottenere risorse nuove e con una specificità maggiore rispetto ai canoni dei Web Service SOAP.

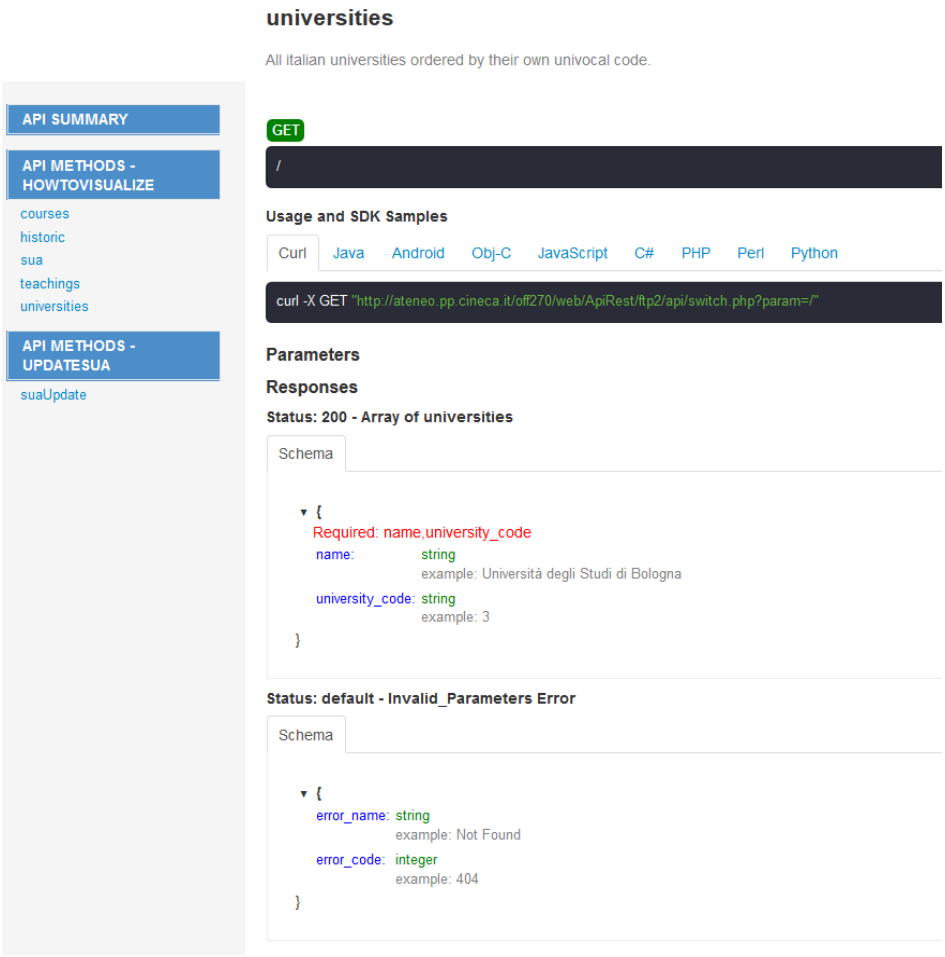


Figura 3.2: Esposizione delle funzionalità di REST-AURATION

Per spiegare il suo funzionamento utilizzeremo la documentazione dell'API.

L'esempio nella figura sovrastante, e quelli seguenti, provengono dalla documentazione generata automaticamente attraverso SWAGGER (vedi capitolo 4).

In questo esempio possiamo notare, in alto a sinistra, le tipologie di risorse e collezioni che il progetto REST-AURATION espone.

Centralmente abbiamo la descrizione di come la nostra API espone le risorse, dove specifica l'URI per ottenere la determinata risorsa e che forma ha quest'ultima. Ad esempio, nella foto possiamo osservare come l'URI "/" ritorni la collezione di tutti gli Atenei nazionali e che campi abbiano queste risorse.

Un'altra conseguenza molto importante, data dall'utilizzo degli URI e dalla frammentabilità delle risorse, è che le risorse sono fortemente connesse. Ad esempio, sempre gerarchicamente parlando di URI, il campo *university_code* della risorsa verrà usato per identificarla all'interno degli URI più "specifici".

Listing 3.2: Esempi di possibili testi della SUA-CdS in formato XML

Esponendo i dati dal 2013 ad oggi la quantità dei Corsi di Studio è molto più elevata e ci si trova quindi a dover specificare l'anno di interesse.

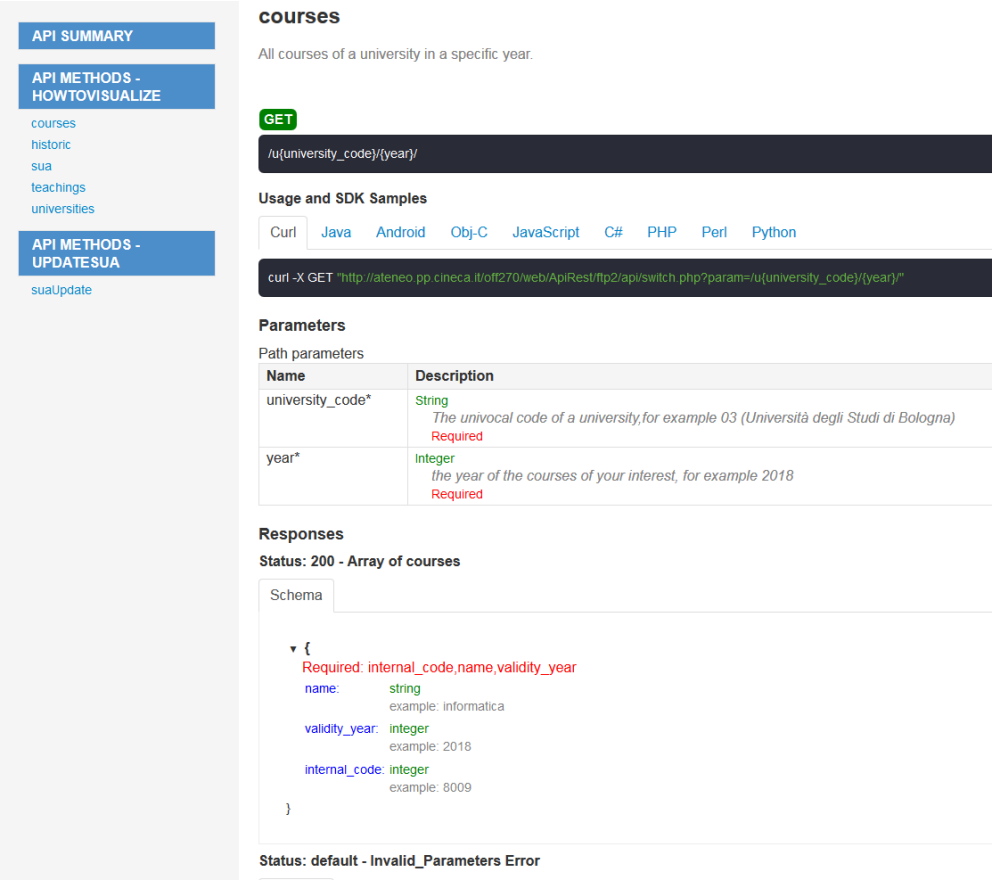


Figure 3.3: Formato dei Corsi di Studio.

In figura possiamo notare come è possibile ottenere i Corsi di Studio inserendo nell'URL *u + university_code*, come ad esempio può essere *u03* nel caso dell'Ateneo di Bologna e, successivamente, un anno di interesse.

Otterremo così la collezione dei Corsi di Studio che contengono un *internal_code*, ossia il codice univoco che ogni corso possiede all'interno del proprio Ateneo. Il medesimo corso può però avere lo stesso codice in anni differenti.

Dopo aver scelto il Corso di Studio attraverso il suo codice, possiamo ora concentrarci sulle risorse che possiede quest'ultimo.

Come si può notare nelle figure precedenti è possibile richiedere al progetto REST-AURATION informazioni di fondamentale importanza, come i testi della SUA-CdS di un determinato Corso oppure gli insegnamenti erogati dal Corso di Studio.

La risorsa di maggior interesse esposta dal progetto REST-AURATION sono sicuramente i testi della SUA-CdS. Nella documentazione sottostante possiamo osservare come

l'utilizzo dell'architettura REST possa rendere più semplice l'interazione Client-Server, nonché la scorrevole lettura da parte di umani e macchine.

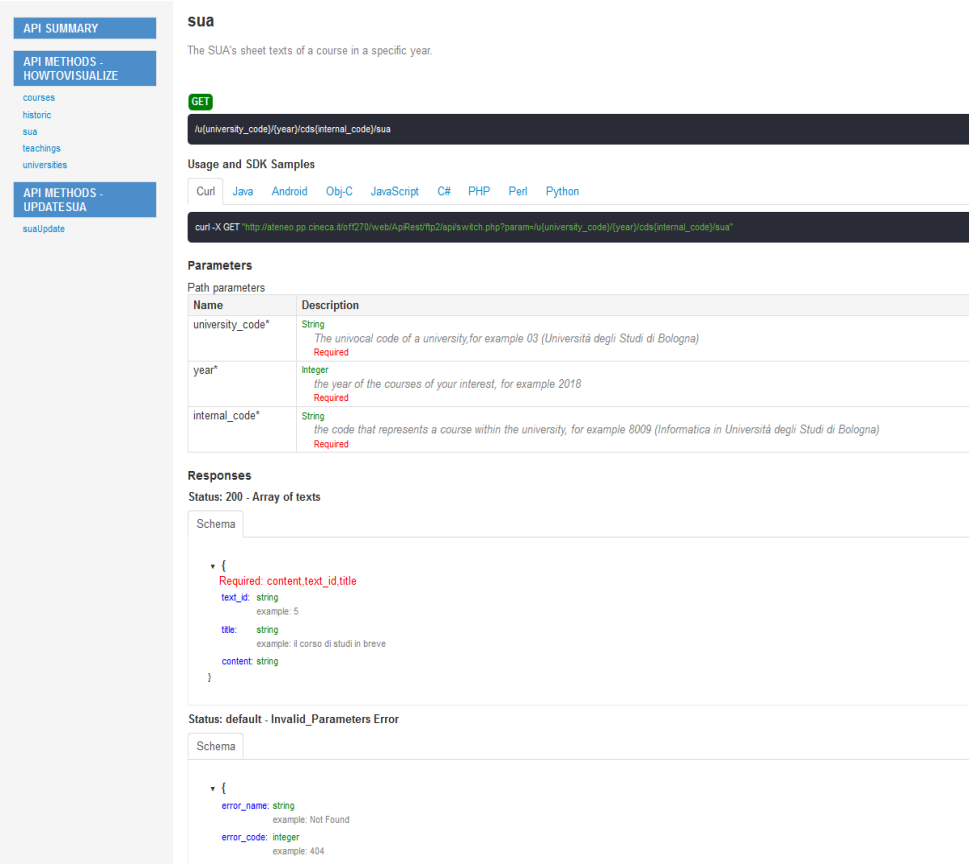


Figura 3.4: Formato della SUA-CdS.

Per gli insegnamenti l'URI viene costruito in modo leggermente diverso, per poter ottenere più specificità. È possibile, infatti, aggiungere campi all'URL come rende visibile la figura nella pagina successiva.

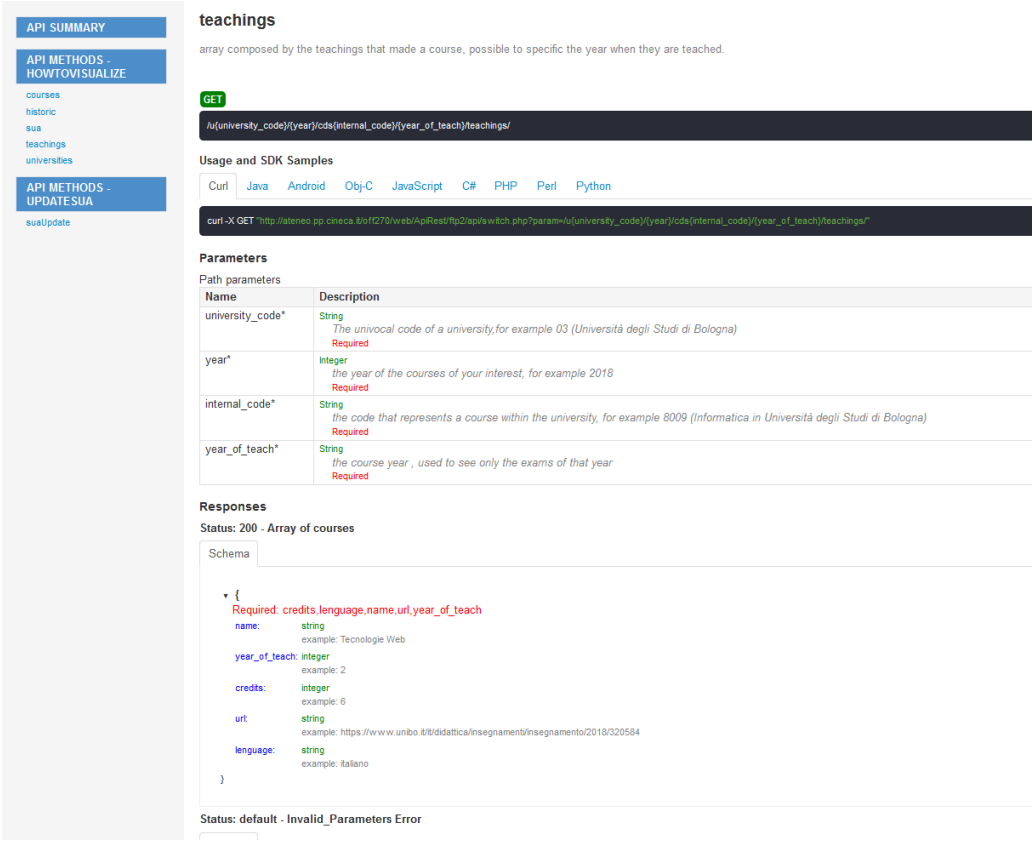


Figura 3.5: Formato degli insegnamenti del Corso di Studio.

Per ultimo, possiamo visionare l'evoluzione di un Corso nel tempo dal 2002 ad oggi, delineato nello storico. Queste possono essere visionate attraverso un URL che possiede la seguente forma, nel caso di Informatica nel 2018: "/u03/2018/cds8009/historic".

La denominazione di un Corso di Studio viene data con lo scopo di rendere chiari gli ambiti che andrà poi a trattare. Questa, tuttavia, può subire cambiamenti con il passare degli anni e delle tecnologie, con conseguente mutamento delle basi sulle quali poggia. Per questo motivo è stato ritenuto importante introdurre lo storico.

Oltre a poter visionare informazioni su Atenei, Corsi di Studio e informazioni inerenti ad essi, l'API Web del progetto REST-AURATION permette di aggiornare i testi della SUA-CdS, solo si è autorizzati a farlo.

Come visibile dalla documentazione il Corso di Studio non è una collezione di risorse, in quanto tutte le informazioni appartengono a risorse sostanzialmente diverse. Non avremo quindi nessuna collezione all'URL: "/u03/2018/cds8009/", semplicemente è un URI incompleto.

Attraverso il metodo POST di HTTP all'URI della Scheda Unica Annuale del Corso di Studio di un determinato Corso, dopo aver specificato il nuovo contenuto del testo e il

text_id, con lo scopo di identificare le informazioni da modificare, è possibile sostituire direttamente i testi interessati.

Non è possibile, però, cambiare i titoli dei testi, in quanto la SUA-CdS deve avere gli stessi parametri di valutazione per tutti, così come non è possibile inserire o modificare i testi con *text_id* avente valore superiore a 65, in quanto non esistenti; questo non vale per i testi mancanti perché risultano semplicemente invisibili per mancanza di testo.

Eliminare testi dalla Scheda Unica Annuale non è stato previsto in quanto questi, seppur vuoti, devono essere presenti.

Non tutti possono modificare i testi della SUA dato che è richiesta un'autenticazione attraverso Username e Password. Questi parametri devono essere concordati, per il momento solo in via teorica, con qualcuno che abbia i diritti per modificare i file che compongono l'API.

Per concludere, REST-AURATION offre al pubblico informazioni che attraverso l'applicazione SUA-CdS attuale non sono modificabili e rende il processo di scambio di messaggi molto più agevole sia nello stabilire una connessione che nello sviluppo dei sistemi

Client e Server.

Capitolo 4

Architettura del progetto REST-AURATION

In questo capitolo, con l'intento di andare a descrivere l'architettura del progetto REST-AURATION, andremo ad approfondire la Specifica OpenAPI e SWAGGER, un software che unisce un insieme di strumenti open-source utili alla creazione di API RESTful che seguano le specifiche OpenAPI.

4.1 OpenAPI Initiative (OAI)

OpenAPI è un'iniziativa creata da un consorzio di esperti dedita a valorizzare l'enorme potere della standardizzazione di come le API REST sono descritte.
Come struttura di *governance* aperta sotto l'amministrazione della Linux Foundation, l'OAI si focalizza sull'importanza e la potenza che un sistema descrittivo indipendente dal linguaggio dell'API REST può avere, promuovendone la creazione e lo sviluppo.
Ovviamente, per portare avanti questa idea, sono necessarie delle specifiche di cui andremo ad elencare i punti cardine:

- Ogni endpoint deve essere raggiungibile e avere delle operazioni associate;
- Ogni operazione deve avere dei parametri di input e di output;
- Sono necessari metodi di autenticazione;
- Ogni API necessita di informazioni sull'autore, licenze e condizioni d'uso.

Le specifiche della propria API possono, e devono, essere scritte in JSON (JavaScript Object Notation), oppure in YAML.
La scelta di questi due linguaggi deriva dal fatto che sono di facile comprensione sia dagli umani che dalle macchine, oltre ad essere conosciuti e utilizzati in lungo e in largo all'interno del mondo informatico.
La specifica OpenAPI completa è disponibile all'interno di un repository di GitHub, un servizio di hosting per lo sviluppo software basato su Git, un sistema di controllo e tracciamento dei cambiamenti al codice sorgente durante lo sviluppo software.

La specifica è trovabile più precisamente al seguente URL:
“https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md”.

4.2 SWAGGER

Il progetto SWAGGER nasce nel 2011 da Tony Tam dalla necessità di generare automaticamente la documentazione delle API REST, limitando la scrittura manuale del programmatore che, senza questo progetto, impiegherebbe ore per descrivere in maniera chiara il proprio progetto.
Nel corso degli anni questo bisogno ha portato ad un aumento esponenziale dell’utilizzo da parte degli sviluppatori di API REST e a supportare di conseguenza il progetto.
Nel 2015 l’azienda SmartBear Software, che tuttora possiede SWAGGER, ha iniziato a contribuire, grazie all’aiuto della sponsorizzazione della Linux Foundation, alla creazione di una nuova iniziativa: la OpenAPI Initiative, già citata numerose volte, a cui oggi collaborano grandi aziende del settore come Google e Microsoft.
SWAGGER, come già accennato, è un insieme di tecnologie e strumenti costruiti attorno alla specifica OpenAPI. Questo permette la descrizione delle API REST in modo che le macchine siano in grado di leggerle e rende possibile la creazione di una documentazione interattiva e facilmente leggibile agli umani.
SWAGGER fornisce anche la possibilità di generare automaticamente librerie client per la propria API in diversi linguaggi.
Tutto ciò è reso possibile dal momento in cui l’API fornisce una descrizione di sé stessa a livello di risorse esposte attraverso gli endpoint.
Questa descrizione contiene informazioni, quali possono essere le autorizzazioni richieste per accedere all’API oppure le operazioni ammesse da quest’ultima.
In pratica, se la API REST segue le specifiche OpenAPI, sarà possibile generare automaticamente questa descrizione, che avrà formato YAML o JSON.

4.3 L’architettura del progetto REST-AURATION

Per iniziare a fare chiarezza, il progetto REST-AURATION, può essere “stratificato” in tre sezioni: la connessione e l’estrazione delle informazioni da DataBase; l’elaborazione delle informazioni e l’esposizione attraverso gli endpoint delle risorse.
Come “contorno” di questa struttura di base, possiamo seguire queste sezioni al contrario in fase di inserimento di nuovi dati, nel caso della modifica dei testi della SUA-CdS.

Un’ altra sezione, utile più allo sfruttamento del progetto, è la documentazione generata appunto attraverso gli strumenti SWAGGER.

4.3.1 PHP

Acronimo ricorsivo di “Hypertext Preprocessor”, PHP, è un linguaggio di programmazione di alto livello, nonostante permetta funzioni a basso livello, debolmente tipato di scripting interpretato, l’interprete necessario è un software libero sotto la licenza PHP.

Un linguaggio ad alto livello ha il vantaggio di consentire una migliore stesura del programma, una più agevole leggibilità da parte di altri programmatori e di rendere più semplice la ricerca degli errori. È caratterizzato da un’astrazione dai dettagli del funzionamento di un calcolatore e dalle caratteristiche del linguaggio macchina.

I linguaggi a basso livello, viceversa, sono quei linguaggi di programmazione che hanno un basso livello di astrazione e sono orientati alla macchina dipendendo in modo molto accentuato dalle caratteristiche hardware di quest'ultima.

PHP nasce nel 1994 ad opera del danese Rasmus Lerdorf e, ad oggi, è utilizzato principalmente per implementare i Server nelle applicazioni Web. Ovviamente non è l’unico utilizzo, concepito inizialmente per facilitare la gestione di pagine Web personali, viene utilizzato largamente per lo sviluppo di Pagine Web dinamiche e supportato da HTML per gli script dinamici.

Alla base dell’enorme successo di questo linguaggio abbiamo la sua enorme flessibilità e i numerosi framework disponibi.

Attualmente, l’ultima versione di PHP rilasciata è la 7.3.9 e, nel corso degli anni ha visto una notevole espansione a livello delle funzionalità.

Ad esempio, fornisce le funzionalità per connettersi e interagire con una grande quantità di DBMS (DataBase Management System) come ad esempio MySQL e ORACLE.

Questa funzionalità, oltretutto risulta molto utile al nostro progetto, in quanto, alla base delle risorse esposte, ci sono le “informazioni grezze” estratte da un database Oracle di sviluppo del CINECA.

Sempre parlando di funzionalità utili al progetto REST-AURATION, dalla versione PHP 5.0 viene migliorato il supporto al paradigma di programmazione orientato agli oggetti. Di seguito verrà spiegato il motivo dello sfruttamento di questa funzionalità.

4.3.2 Il database e la connessione

La connessione al database è il primo passo verso l'estrazione delle risorse.
All'interno del file che si occupa della connessione al database, *database.php*, viene istanziato un modello utile alla creazione di oggetti, chiamata classe, che avrà il compito di creare l'oggetto *Database*. Una volta istanziato, questo avrà, nei suoi metodi la capacità di creare una connessione temporanea al database, in modo da estrarre quello che ci serve. La classe possiede due campi contenenti rispettivamente l'username e la password. questi sono necessari in quanto il database non permette a chiunque di estrarre i dati e richiede perciò un'identificazione. In questo caso sarà ristretto ai dipendenti del CINECA.
La classe è l'unica sezione di codice contenuta nel file. L'unico scopo effettivo di questa istanza è creare la connessione al database Oracle, attraverso una funzione chiamata *getConnection* che non possiede parametri di input e sarà il solo metodo appartenente all'oggetto utilizzato.
Quella che seguirà è la sezione di codice chiave del file *database.php*.

```
    public function getConnection(){

        $this->conn = null;

        $this->conn = oci_connect($this->username,$this->password,
                        "MIUR01_DEV", "AL32UTF8");
        if (!$this->conn) {
            $m = oci_error();
            echo $m['message'], "\n";
            exit;
         }
         else {
                return $this->conn;

        }
    }
```

Listing 4.1: Connessione al database

In questa breve sezione di codice la funzione più importante è sicuramente *oci_connect*, messa a disposizione da PHP 5.1.2 che permette di connettersi automaticamente ad un database Oracle.

I parametri di questa funzione, in sequenza, sono: le credenziali (username e password); il nome del database, ossia *MIUR01_DEV*, e infine il set di caratteri supportato, *AL32UTF8*, una codifica in UTF8.
La funzione restituisce un identificatore di connessione necessario ad estrarre poi i dati nel caso in cui la connessione sia stata stabilita, in caso contrario restituirà la stringa "FALSE" e la funzione *getConnection* restituirà di conseguenza un messaggio di errore.

4.3.3 L'estrazione dei dati

Ora che sappiamo dove andare a prendere le risorse, rimane da esplorare solamente la questione della loro estrazione.
Come per il marmo grezzo serve il piccone, per i dati all'interno di un database serve un linguaggio di interrogazione. Il nostro piccone sarà SQL, acronimo di "Structured Query Language", cioè un linguaggio di interrogazione per database che si basano sul modello relazionale.
Nel file *product.php* viene istanziata una nuova classe, chiamata *Product*. Dopo la connessione al database da parte della classe *Database*, il nuovo oggetto avrà il compito di estrarre i record dalla banca di dati del CINECA.
Come in ogni classe, *Product* avrà campi e metodi che, in questo caso, saranno tutti funzionali ad ottenere nuove informazioni. I campi serviranno a "immagazzinare" i parametri, ossia i filtri per le richieste SQL dei metodi.
Tra i tanti campi di particolare interesse, troveremo *conn*, che tramite il costruttore della classe verrà posto uguale all'identificativo di connessione precedente ottenuto dalla funzione *getConnection*. Questo campo sarà necessario alle query, ossia le richieste SQL.
I metodi otterranno i dati, sfruttando i campi che spesso saranno passati da un file esterno, e li restituiranno leggermente rimaneggiati.
Per spiegare in maniera più chiara, seguirà un esempio.

```
function testi(){

        $query ="SELECT
                    s.id_testo as text_id , s.contenuto as content,
                        z.titolo as title
                FROM
                    z_sua_testi s, z_sua_testi_titoli z
                    WHERE s.id_rad=:id_rad and
```

```
                                s.id_testo=z.id_testo";

        $stmt2=oci_parse($this->conn,$query);
        oci_bind_by_name($stmt2,":id_rad",$this->id_rad);
        oci_execute($stmt2);

        return $stmt2;
}
```

Listing 4.2: La funzione *testi*.

Il metodo *testi* ha lo scopo di ottenere i titoli della SUA- CdS.

La query, scritta all'interno della variabile *$query*, possiede a sua volta una variabile, che in SQL è definita con ":" davanti al nome di essa che, in questo caso è *:id_rad*.

Partiamo dallo spiegare cosa sia un Id RAD.

Come dice in parte il nome, è un identificativo univoco con cui viene individuato il Corso di studi all'interno dei database istituzionali.

Per spiegare come associare le variabili a dei valori spieghiamo i metodi in ordine logico.

Partiamo quindi da *oci_parse*. Questo prendendo in input l'identificativo di connessione, ottenuto in precedenza, e la query restituisce uno "statement identifier", una sorta di immagine che verrà utilizzata in seguito per altre funzioni come legare le variabili all'interno di *$query* e per poter estrarre i record del database.

Avremo poi *oci_bind_by_name*, una funzione sempre messa a disposizione da PHP. Questo metodo prende in input lo "statement identifier", la variabile a cui assegnare un valore all'interno dello statement e la variabile *id_rad*, che in questo caso è un campo della classe *Product*, ottenuta sempre da un foglio esterno.

Con questa funzione avremo l'*id_rad* del Corso di cui vogliamo ottenere la SUA-CdS all'interno dello statement chiamato, nell'esempio, *$stmt2*.

Per finire, viene eseguito lo statement tramite la funzione *oci_execute*, che lo prende come parametro ed esegue le istruzioni della query, restituendo "TRUE" in caso di successo e "FALSE" in caso di errore.

I risultati potranno essere ripresi in futuro attraverso la funzione *oci_fetch*, che prende in input sempre lo "statement".

Il file *product.php* contiene tanti altri metodi analoghi all'esempio fatto, ognuno dei quali con la stessa finalità di *testi*: restituire uno statement da cui successivamente verranno ricavati i dati.

Questi dati però sono ancora da estrarre ed esporre. Attraverso una serie di file, sempre con estensione *.php*, andremo ad utilizzare tutte le funzioni sviluppate in questi file per ottenere all'occorrenza quello che viene richiesto ed esporlo poi.

Senza stare a mostrare nuovamente ciò che fanno le funzioni sopra citate richiamate dal file, riprenderemo però l'esempio in cui cominciamo ad estrarre e lavorare la SUA-CdS di un Corso di Studio.

Il file che prenderemo in esame è *testi.php*, dove vengono istanziati gli oggetti dichiarati in *database.php* e *product.php* per creare una connessione al database e poter ottenere lo "statement" di cui abbiamo tanto parlato in precedenza.

Nelle righe successive dichiariamo la funzione *returnTesti* che prende in input *id_rad_request2*, semplicemente un *id_rad* di un Corso di Studio, utile per estrarre i testi correlati a questo CdS.

Questo è ciò che segue:

```
function returnTesti($id_rad_request2){
[...]
[...]
      $array_testi = array();

      $i=0;

      while(oci_fetch($stmt2)){
          $cont=oci_result($stmt2 , "CONTENT");
          $array_testi[$i]=array(
              "text_id"=>oci_result($stmt2 , "TEXT_ID"),
              "content"=>$cont->load(),
              "title"=> oci_result($stmt2, "TITLE")
          );
      $i++;
      }
      return($array_testi);
}
```

Listing 4.3: La funzione *returnTesti*

Dichiariamo un array chiamato *array_testi*, che funzionerà da scatola per le nostre informazioni.

Un array può essere definito come: una collezione organizzata di oggetti. Il concetto di "collezione" implica che tali oggetti siano variabili dello stesso tipo. L'aggettivo

"organizzata" indica la possibilità di identificare univocamente tutti gli oggetti dell'array in modo sistematico.

Nella seconda linea inizializziamo anche un iteratore *i* ponendolo uguale a 0. Questo ci servirà per riporre in "scatole" più piccole tutte le informazioni inerenti a ogni testo, senza perdere l'ordine di queste "scatole".

Per finalmente riempire la scatola utilizzeremo un ciclo While, un costrutto iterativo caratteristico di tutti i linguaggi di programmazione imperativi, che itererà finché non avrà visionato tutti i record dello statement *stmt2*, ottenuto dal metodo della classe *Product*.

Il ciclo itererà fino a quando la funzione *oci_fetch* non ritornerà "FALSE" e quindi, fino a quando avrà finito di salvare in un buffer di memoria interno e mascherato le informazioni estratte dai record.

All'interno di questo ciclo la funzione *oci_result* restituisce il valore contenuto nel campo specificato insieme allo "statement" di riferimento *$stmt2,* nei parametri della funzione.

I campi dello statement restituiti sono:

- *CONTENT,* il contenuto del testo della SUA-CdS. Il suo formato non è leggibile finchè non utilizziamo la funzione *load()* di PHP per renderlo una stringa;
- *TITLE,* il titolo nonché l'argomento che verrà trattato all'interno del *CONTENT*;
- *TEXT_ID,* il codice associato al testo e al titolo della SUA-CdS.

Una volta ottenuti questi tre campi, andremo a creare un array all'interno di *array_testi*, avendo così una scatola composta da tante "scatoline" contenenti questi tre elementi, tutti etichettati tramite un'associazione chiave-valore.

Una volta finita di riempire, la scatola è pronta per essere impacchettata e spedita anche se, per il momento, non è un prodotto fatto e finito. Il formato non ci piace, ma questo non è nelle mansioni del file *testi.php*.

Iniziamo così la spedizione della scatola, dove *array_testi* sarà il valore restituito dalla funzione *returnTesti*.

Il prodotto ottenuto da questo processo non coincide con quello desiderato. Potremmo dire che la nostra scatola non è ancora stata impacchettata come realmente vorremmo, ma iniziamo ad avvicinarci.

4.3.4 L'esposizione delle risorse

Ci troviamo ora ad avere delle risorse quasi lavorate completamente e pronte a essere messere in vetrina.

Il compito spetta al file *switch.php*, che ha il ruolo di preparare gli endpoint per poter fornire le risorse a chi le richiede.

La struttura di base del file è molto semplice. Prima di tutto crea delle dipendenze con tutti gli altri file sopra citati e successivamente si mette "ad aspettare" una richiesta delle risorse.

Ad ogni richiesta il codice eseguito riprende tutto ciò che è stato fatto in precedenza, istanziando le classi *Database* e *Product*, e infine trasforma in JSON l'array ottenuto dai file intermedi, come *testi.php* oppure *atenei.php*.

Ricordiamo che una richiesta, nel nostro caso, supporterà solo il metodo GET di HTTP fatta eccezione nell'inserimento, in cui sarà possibile utilizzare il metodo POST,

Il file, una volta ricevuta una richiesta, che dovrà avere la forma di un URI secondo la filosofia REST, il codice analizza il parametro "param" di quest'ultima dopo averlo "spaccato" attraverso la funzione "explode" di PHP e aver inserito ogni campo dell'ormai non più percorso all'interno di un array chiamato "$path". Attraverso un immenso "switch-case", altro costrutto caratteristico della programmazione imperativa, il file controllerà se i campi della richiesta corrispondono a qualche endpoint.

Un esempio di questo controllo è quello che segue.

```
switch($count){
    case "0":
    header("HTTP/1.1 404 Not Found");
$arrr=array("url"=>"http://ateneo.pp.cineca.it/off270/web/ApiRest/ftp2
/api/errors/404.html");
    echo json_encode($arrr);
    break;

    case "2":
        if($path["0"]=="" && $path["1"]==""){
        $var=returnAtenei();
        echo json_encode($var);
        }
        else{
            header("HTTP/1.1 404 Not Found");
$arrr=array("url"=>"http://ateneo.pp.cineca.it/off270/web/ApiRest/ftp2
/api/errors/404.html");
            echo json_encode($arrr);
            }
        break;
```

Listing 4.4: Sezione del codice di controllo dell'URI in *switch.php*.

$count è la variabile che come valore ha il numero di campi che aveva la richiesta.
A seconda di che valore ha, andremo a valutare un caso specifico per verificare che esista un endpoint corrispondente alla richiesta.
Nel primo caso, in cui *$count* corrisponde a zero, verrà restituito un messaggio con un JSON contenente l'URL di una pagina di errore, più precisamente quella dell'errore HTTP 404 "Resource not found".
Negli altri casi, a seconda di corrispondenza tra le risorse esistenti, verrebbero ripercorsi tutti i passaggi per ottenere una risorsa.
Se per caso la richiesta avesse la forma "/", *switch.php* restituirebbe al client il JSON contenente una collezione di tutte gli Atenei italiani.
Nel caso i campi della richiesta fossero giusti, ma secondo la logica del progetto poco specifica, come ad esempio "/u03/", viene costruito da parte del file un messaggio di errore con Header HTTP 400, ossia "formato della richiesta sbagliato", e un oggetto JSON in cui si propongono tutte le vie possibili per raggiungere un endpoint.

4.3.5 L'inserimento delle risorse

Come già detto, è possibile aggiornare la SUA-CdS. Andiamo di seguito a spiegare come questo sia possibile.
Il codice è lo stesso per i casi della richiesta delle risorse, fatta eccezione per il caso in cui la variabile *$_REQUEST['data']* abbia un valore, in quanto ciò vorrebbe dire che sono stati mandati dati al Server, che, in caso siano considerati corretti, andranno ad aggiornare il database.
Nel caso sia presente la variabile *$_REQUEST['data']* nell'header della richiesta, questo verrà concatenato all'URL di *$param* e poi verrà trattato esattamente come se ne avesse sempre fatto parte, con i controlli opportuni.
Per essere più precisi, nel caso non sia presente viene concatenata una stringa vuota.
Vengono poi riportate qui le sezioni di codice chiave.

```
[...]
 if(isset($_POST["data"]) && array_key_exists("text", $_POST["data"])
&& array_key_exists("text_id", $_POST["data"])){
                    $cds=str_replace("cds","",$path["3"]);
                    if($_POST["data"]["text"]===NULL||
$_POST["data"]["text_id"]===NULL||
```

```
!is_numeric($_POST["data"]["text_id"])
||$_POST["data"]["text_id"]>65){
                        header("HTTP/1.1 400 Bad Request");
                        exit();
                    }
                    update_sua($_POST['data'],$path["2"],$cds);

$arrr=array("url"=>"http://ateneo.pp.cineca.it/off270/web/ApiRest/ftp
2/api/200.html");
                    echo json_encode($arrr);
                    exit();
                    }
[...]
```

Listing 4.5: Codice di controllo in inserimento

Nell'esempio viene mostrato il controllo effettuato sui dati spediti tramite il metodo HTTP POST, dove viene controllato se effettivamente è stato effettuato un metodo POST correttamente, visionando se all'interno di esso vi è una variabile *data*, e se questa è stata popolata correttamente in quanto array.

Una volta superato questo controllo, il testimone passa al file *insert.php*, nel quale viene prima di tutto impostato un Header a 401 "Unauthorized", che rimarrà tale finché non saranno state inserite le credenziali attraverso il metodo di autenticazione Challenge-Response a chiave pubblica Digest Authentication.

Queste credenziali vengono combinate e cifrate, appunto "digerite" in modo che non vengano mai palesate durante la verifica di quest'ultime ma si abbia una stringa senza un significato che è una combinazione dell'username, della password e altre variabili sconosciute a chi utilizza il servizio.

Il luogo in cui, per il momento, sono salvate le credenziali, è all'interno di un array dentro il file stesso, per questo le credenziali devono essere concordate.

L'autenticazione rimane valida fino alla chiusura della sessione.

Se l'operazione di autenticazione è corretta, come anche i parametri, verrà aggiornato il database effettuando il processo inverso rispetto a quando i dati vengono esposti e, se l'operazione viene eseguita correttamente, verrà impostato l'header HTTP 200 "ok". Nel messaggio spedito dal Server vi sarà un URL che punta ad una pagina HTML sempre facente parte del server con scritto "200-OK The request was fulfilled.".

nel caso in cui solo i parametri non siano corretti, il database non verrà aggiornato e l'header della richiesta verrà posto a 400 "bad request".
Il messaggio ritornato dal Server in questo caso conterrà un URL ad un'altra pagina HTML però contenente una scritta analoga all'header HTTP appena impostato.

4.3.6 Documentazione

La documentazione è stata generata automaticamente attraverso SWAGGER, un insieme di tool di cui abbiamo già discusso a inizio capitolo.
La documentazione non è altro che una formalizzazione di ciò che è stato illustrato in questo capitolo ed esiste in due formati: YAML e HTML.
All'interno del progetto REST-AURATION sono visionabili entrambi, rispettivamente ai seguenti URL:

- Formato YAML:
  "http://ateneo.pp.cineca.it/off270/web/ApiRest/ftp2/api/documentation/Cavicchi-CinecaRestApiDocumentation.yaml".
- Formato HTML:
  "http://ateneo.pp.cineca.it/off270/web/ApiRest/ftp2/api/documentation/index.html".

La prima serve soprattutto a generare quella in formato HTML con l'ausilio di SWAGGER, nonostante abbia il difetto di rimanere poco leggibile dagli umani.
La seconda è il risultato dell'operazione appena citata e viene quindi generata automaticamente ma, allo stesso tempo, risulta molto intuitiva e interattiva.

Capitolo 5

Valutazione del progetto

REST-AURATION

5.1 Valutazione del funzionamento

Per verificare il funzionamento del PROGETTO REST-AURATION è stato deciso di sviluppare un client *ad hoc* che verifichi funzionalità del sistema.

Il client è locato sempre all'interno di uno spazio dedicato presso CINECA ed è liberamente accessibile tramite l'indirizzo:

"http://ateneo.pp.cineca.it/off270/web/ApiRest/ftp2/api/client.html".

Il client è un documento HTML dinamico grazie ad un breve script Javascript, insieme al framework dello stesso linguaggio JQuery.

Javascript è un linguaggio di scripting, orientato agli oggetti e alla gestione degli eventi, che nasce nel 1995 ad opera di Brendan Eich. Per la sua concezione è sempre stato utilizzato nella programmazione client-side.

Nonostante ciò, negli ultimi tempi, viene utilizzato anche per lo sviluppo di Web API server-side, soprattutto grazie al rilascio di framework che hanno permesso la progettazione di queste applicazioni più semplice, quale ad esempio Express.

JQuery è stato utilizzato per semplificare la scrittura degli script Javascript, la manipolazione del Document Object Model (DOM) di HTML e la gestione degli eventi.

Fondamentale è l'utilizzo di JQuery poiché implementa le funzionalità AJAX (acronimo di Asynchronous Javascript And XML), fondamentali per l'implementazione del nostro client.

AJAX è un insieme di tecniche di programmazione Web che utilizzano tecnologie per creare applicazioni Web client-side asincrone.

L'asincronicità è un elemento fondamentale per aumentare le prestazioni poiché permette di continuare l'analisi della pagina HTML mentre, in background, vengono caricate ed eseguite operazioni sulle risorse richieste e ottenute attraverso la connessione ad un server. Dal punto di vista dell'usabilità, invece, AJAX permette di caricare risorse senza dover per forza ricaricare il documento.

Come già detto, questo insieme di tecnologie può essere considerato come fondamentale per verificare il funzionamento del progetto REST-AURATION. Grazie ad una piccola sezione di codice all'interno di uno script che andremo poi ad analizzare, riusciremo a connetterci a tutti gli endpoint forniti dalla nostra API.

```
$.ajax({
            type: "GET",
            url:  "http://ateneo.pp.cineca.it/off270/web/ApiRest/ftp2/
                  api/switch.php?param=" + val + "&Access-Control-
                  Allow-Origins=*",
            contentType: "application/json; charset=utf-8",
            dataType: "json",
            success: function(data){
                console.log(JSON.stringify(data,null,2));
                $("#append").append('<table id="tab" class="table">');
                $("#tab").append('<thead id="ok"><tr id="cont">');
                var cho=0;
                for(x in data[0]){
                    if(x=="university_code"){
                        cho=1;
                    }
                    else if(x=="internal_code"){
                        cho=2;
                    }
                }
            [...]
 error: function(errorMessage) {
                console.log(errorMessage.status);
                var resp=jQuery.parseJSON(errorMessage.responseText);
                if (errorMessage.status==400){
              [...]
                    }
                else window.location.href =
                     errorMessage.responseText.url;
            }
  });
```

Listing 5.1: Chiamata AJAX all'API REST-AURATION

Questo esempio direttamente tratto dal sorgente del nostro client, permette di osservare la struttura di una cosiddetta “chiamata AJAX”.

I parametri *type*, *url*, *data-type* e *content-type* specificano i parametri della richiesta. Il primo specifica il metodo HTTP utilizzato nella richiesta, in questo caso, GET, il metodo utilizzato quando richiediamo delle risorse; il valore di *url* è l’endpoint a cui ci stiamo collegando; *data-type* e *content-type*, rispettivamente, specificano il formato dei dati che ci fornirà il server, in questo caso JSON, e il formato dei dati che andremo ad inviare al server, ma che nel nostro caso è inutile,

“SUCCESS” e “ERROR” sono due funzioni che vengono chiamate a seconda che la chiamata al server sia andata a buon fine o meno e di conseguenza andranno a operare su risorse sostanzialmente diverse.

La funzione “SUCCESS” avrà il compito di operare per elaborare le risorse restituite dalla chiamata AJAX. Nel nostro caso avremo un JSON contenente le informazioni richieste attraverso l’endpoint, già spiegato approfonditamente nei Capitoli 3 e 4. Queste informazioni saranno poi processate, rimaneggiate e inserite dinamicamente all’interno del documento HTML e, quindi, visualizzate.

In caso di “ERROR” la nostra chiamata AJAX riceverà come risposta un messaggio di errore che, oltre al codice di status della richiesta, come ad esempio “404”, conterrà un oggetto JSON a sua volta contenente degli url a pagine di errore o a endpoint, che verranno elaborati e inseriti all’interno dell’HTML. Questo accade, ad esempio, quando l’url della chiamata AJAX è logicamente incompleto o profondamente diverso.

A questo esempio ne seguiranno altri, utili a mostrare cosa succede quando il client interagisce con la nostra API, per mostrare come sia possibile testarla.

Nell’immagine successiva, attraverso un’istantanea dello schermo, illustreremo effettivamente cosa succede nel momento in cui effettuiamo una chiamata AJAX e come la chiamiamo.

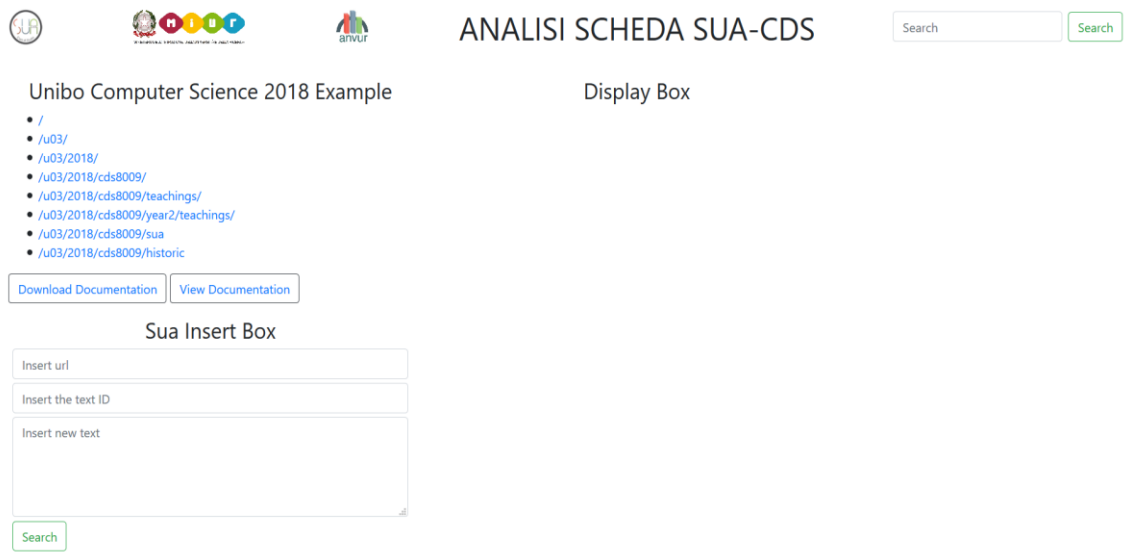


Figura 5.1: Schermata del Client

Questo è il primo impatto che abbiamo con il nostro client.

A sinistra abbiamo un esempio di possibili endpoint per sfruttare al massimo ciò che REST-AURATION espone in merito al Corso di Informatica nel 2018 presso l'Ateneo di Bologna.

Questi URL sono cliccabili e andranno ad inizializzare la chiamata AJAX che avrà come parametro *url* la stringa cliccata.

Si può notare che non tutti gli URL sono logicamente completi, alcuni sono stati inseriti per mostrare come viene elaborato un messaggio di errore.

Affianco al titolo, in alto a destra, possiamo effettuare l'inserimento di un URL a nostro piacimento.

Come dice anche la voce in basso a sinistra, in questa sezione avremo il form compilabile per poter aggiornare la SUA-CdS. Nel caso tentassimo una prova di inserimento comparirebbe un secondo form come nella figura seguente.

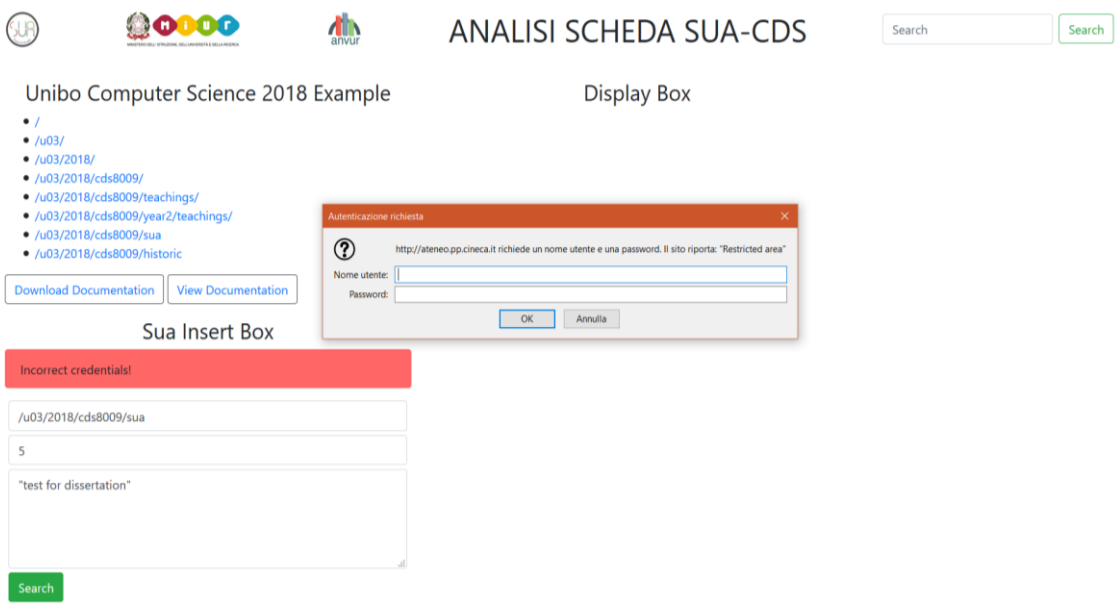


Figura 5.2: Form di autenticazione

Il form in primo piano appartiene alla prima risposta tramite HTTP del server al client, in cui vengono richiesti username e password per poter modificare la SUA-CdS.

In caso si sia in possesso di queste credenziali, insieme all'header HTTP della risposta del client al server verranno inviate permettendo all'utente di proseguire e ricevere, infine, un messaggio con lo status code 200 e l'oggetto JSON, contenente un url ad una pagina HTML di successo, a cui si arriva attraverso la funzione di successo di una chiamata AJAX apposita per l'inserimento.

La parte centrale e di destra della schermata è dedicata alla visualizzazione delle risorse offerte da REST-AURATION.

Di seguito verrà mostrata una cattura dello schermo in cui possiamo osservare i primi dati restituiti cliccando sull'url "/u03/2018/cds8009/teachings/", come si può leggere al di sopra della tabella.

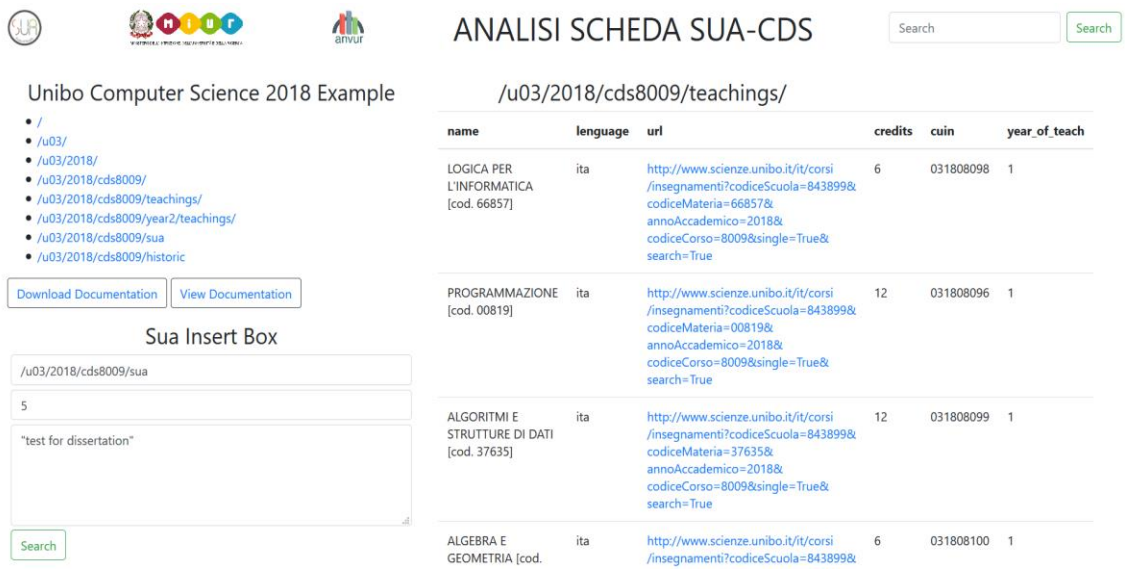


Figura 5.3: Tabella con l'elenco degli insegnamenti

5.2 Valutazione delle funzionalità

Nel capitolo precedente è stato valutato il funzionamento del progetto, in questo capitolo faremo un piccolo confronto tra le funzionalità del progetto REST-AURATION e la Scheda Unica Annuale del Corso di Studio attuale.

A livello di quantità di dati esposti, il progetto REST-AURATION fornisce una quantità di dati di gran lunga superiore e tutte inerenti al Corso di Studio di cui viene esposta la SUA-CdS, che espone esclusivamente i testi.

Questo apre numerose porte per lo sviluppo di nuove API e Web services che prendono come punto di partenza il progetto REST-AURATION oppure che lo sfruttano in combinazione con altri.

Avere una varietà dei dati più ampia può andare a coprire zone di più ampio interesse, nonché contribuire a sviluppare una conoscenza più profonda di questo strumento che già all'interno dei testi racchiude tante informazioni caratterizzanti di un Corso di Studio, non solo riguardo agli insegnamenti erogati ma anche inerenti alla struttura e i servizi.

5.3 L'esposizione dei dati

Come già detto, il progetto REST-AURATION espone una quantità e una varietà di dati maggiore. Ma come vengono esposti e quanto sono visibili?

Sicuramente l'esposizione dei dati è molto intuitiva grazie agli URI "parlanti".

Inoltre, la possibilità che offre il progetto REST-AURATION in merito all'esposizione non è tanto in merito alla semplicità o alla quantità di dati, bensì al fatto che sia "aperto a tutti", ognuno può elaborare un proprio "client" con le risorse ottenute.
D'altro canto, la SUA-CdS ed il portale Universitaly rivestono un ruolo di riferimento per l'orientamento.
Nonostante ciò, il calo degli accessi al portale è evidente e una reingegnerizzazione, come nel caso del progetto REST-AURATION può portare a cambiamenti, anche graduali, molto veloci nello sviluppo e specifici ad una certa tipologia di risorse restituita.

5.4 La frammentazione delle risorse

La frammentazione delle risorse è un grosso punto a favore di REST-AURATION. Il così grande numero di endpoint permette di ottenere (quasi) solo le risorse che ci interessano.
Ad esempio, come visto nell'architettura del nostro client, la possibilità di richiederle tutte attraverso una singola chiamata AJAX rende tutto molto facile.
Avere risorse di così poche informazioni occupa la banda di
Parlando dell'applicazione SUA-CdS che utilizza SOAP la frammentabilità è nulla e, inoltre, anche il suo utilizzo richiede vari passaggi per trasformarla, eliminando tutti i tag di contorno necessari per il protocollo SOAP.
Per sostenere REST-AURATION, l'utilizzo di JSON fa sì che le risorse possano essere "studiate" senza troppa fatica e lavorate con semplicità, richiedendo anche solo pezzi della collezione di risorse.
Una visione alternativa che avrebbe potuto aumentare la frammentazione dei testi della SUA-CdS è la seguente.
Poterla trattare come una collezione di risorse e non una unica, avrebbe potuto permettere di richiedere solamente un testo specifico, anche se per identificarlo sarebbe stata necessario utilizzare il codice identificativo (poco intuitivo) oppure il titolo del testo, che comunque presenta una lunghezza non poco rilevante e quindi facile da sbagliare.
Una possibile soluzione sarebbe osservare con attenzione la SUA-CdS esposta dal progetto REST-AURATION e poi, all'interno della chiamata AJAX, trattare l'array dei testi e vincolarlo nell'esposizione ai soli testi desiderati.

Capitolo 6

Conclusioni e sviluppi futuri

Nonostante sia frutto di un lavoro chiaramente modesto, possiamo affermare che REST-AURATION può essere utilizzato come base di partenza per diffondere l'utilizzo di REST all'interno di una realtà importante come CINECA o per la costruzione di Web services che possono trarre vantaggio e risorse da questa API.

Quelli che sono i punti forti di REST rispetto a SOAP sono stati verificati. L'API presenta buona scalabilità, efficienza e semplicità nell'accesso alle risorse.

Inoltre, se utilizzato, soddisferebbe i requisiti IODL 2.0 e potrebbe portare alla creazione di un nuovo portale o alla rielaborazione di Universitaly.

Il progetto REST-AURATION, come ci si potrebbe aspettare, non è esente da errori e imperfezioni. Il progetto è incompleto e il percorso da fare per sostituire la SUA-CdS strutturata con SOAP è lungo e tortuoso.

Il linguaggio di programmazione è di gran lunga lo scoglio più grande.

Questo progetto è stato scritto con la versione 5.2.14 di PHP in quanto era la versione più aggiornata sul Server Web concessomi durante il tirocinio.

Un'API, in conformità con lo stato dell'arte, viene scritta in linguaggio Javascript insieme ai framework, come Express e Node.

In alternativa a Javascript, ad oggi, è stata rilasciata la versione 7.3.9. di PHP.

Riguardo alla sicurezza in inserimento, la scelta del metodo di autenticazione *Digest HTTP* è stata vincolata dal linguaggio di programmazione utilizzato. Le librerie per questa versione di PHP non vengono più pubblicate o, per lo meno, non riguardano le tecniche più moderne, anche perché molto probabilmente si basano su tecnologie introdotte nelle versioni del linguaggio di programmazione successive.

Una possibile soluzione potrebbe essere l'utilizzo del sistema di autenticazione *Oauth2*.

Un ultimo proposito, in favore della sicurezza e della chiarezza, sarebbe stato mascherare il file *switch.php*, mantenendo solamente esposto attraverso il browser l'URL della risorsa richiesta.

Bibliografia

[BH04] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris e D. Orchard. *"Web Services Architecture"*, https://www.w3.org/TR/ws-arch/ , W3C Working Group Note 11 febbraio 2004.

[BE00] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte e D. Winer, *"Simple Object Access Protocol (SOAP) 1.1"*, https://www.w3.org/TR/2000/NOTE-SOAP-20000508/ ,W3C Note 8 maggio 2000.

[FIE00] R. T. Fielding, *"Architectural Styles and the Design of Network-based Software Architectures"*, University of California, 2000.

[GI11] D. Guinard, I. Ion e S. Mayer *"In Search of an Internet of Things Service Architecture: REST or WS-*? A Developers' Perspective"*, 2011.

[MON18] A. Monus, *"SOAP vs REST vs JSON comparison [2019]"*, 2018.

[BAS19] S. Bassini, *"E' tempo di storia, scopriamo l'informatica"*, https://www.aicanet.it/storia-informatica/calcolo-scientifico-in-italia/cineca , ultima visita: 12 settembre 2019.

[CIN19] *"Chi siamo | Cineca"*, https://www.cineca.it/it/content/chi-siamo , ultima visita: 11 settembre 2019.

[ANV19] *"AVA - Autovalutazione, Valutazione Periodica e Accreditamento"*, https://www.anvur.it/attivita/ava/ , ultima visita: 17 settembre 2019.

[MIU19] *"Il portale per la qualità delle sedi e dei corsi di studio"*, http://ava.miur.it/ , ultima visita: 20 settembre 2019.

[BEL12] E. Belisario, *"IODL 2.0: la nuova licenza italiana per gli Open Data è ancora più semplice e aperta"*, https://blog.ernestobelisario.eu/2012/03/08/iodl-2-0-la-nuova-licenza-italiana-per-gli-open-data-e-ancora-piu-semplice-e-aperta/ , 2012.

[SC10] J. Stearley, S. Corwell e K. Lord, *"Bridging the Gaps: Joining Information Sources with Splunk"*, 2010.

[JC17] J. Jeon e J. Chung, *"Developing a Prototype of REST-based Database Application for Shipbuilding Industry: A Case Study"*, 2017.

[UZ11] B. Upadhyaya, Y. Zou, H. Xiao, Joanna Ng e A. Lau, *"Migration of SOAP-based Services to RESTful Services"*, 2011.

[DS14] A. Dudhe e S.S. Sherekar, *"Performance Analysis of SOAP and RESTful Mobile Web Services in Cloud Environment"*, 2014.

[LW08] Y. Liu, Q. Wang, M. Zhuang e Y. Zhu, *"Reengineering Legacy Systems with RESTful Web Service"*, 2008.

[GI11] D. Guinard, I. Ion e S. Mayer, *"In Search of an Internet of Things Service Architecture: REST or WS-*? A Developers' Perspective"*, 2011.

[RM13] R. Rodriguez-Echeverria, F. Macias, V. M. Pavon, J. M. Conejero e F. Sanchez-Figueroa, *"Model-Driven Generation of a REST API from a Legacy Web Application"*, 2013.

Ringraziamenti

Vorrei ringraziare di cuore, innanzitutto, la mia famiglia per avermi sostenuto in ogni singola scelta presa fino a questo momento.

Un altro grazie va ai miei amici. Siete sempre stati al mio fianco durante questo percorso lungo e tortuoso, ascoltandomi nei momenti di difficoltà e di gioia, condividendo una miriade di eventi che non sarà possibile dimenticare.

Un ringraziamento speciale va a Vivian e mi piacerebbe farlo attraverso questa frase di Fabrizio De André:
"Alla compagna di viaggio
i suoi occhi il più bel paesaggio
fan sembrare più corto il cammino."
Grazie per aver sopportato le mie ansie, le mie isterie e, soprattutto, per avermi aiutato ogni singolo istante, oltre a regalarmi momenti di gioia indescrivibili.

Grazie veramente a tutti, senza di voi non sarebbe stato possibile.