

# Integrazione Validator con REST Services esposti dalle Views di STU3

***Specifiche tecniche di redazione  
modulistica per l'interazione con la  
struttura e banca dati del backend  
STU3. “BP-OR-AP-02 - Integrazione  
Validator-RestService” del kit di riuso  
del progetto PON-GOV “SPRINT”***

## Per conto di: Sviluppatori interi o esterni di Modulistica Globo STU3

Andrea Cividini <i>Redatto</i>	Marco Deligios <i>Verificato e autorizzato</i>	6/3/2020 <i>Data d'emissione</i>	2 <i>Revisione</i>
I_Prodotti <i>Commessa</i>	Protocollo <i>Protocollo</i>	Pagina 2 di 12	

Le informazioni contenute in questo documento sono di proprietà intellettuale della GLOBO srl, sotto licenza [CC-BY-4.0](#). Prodotti o aziende indicate nel documento possono essere marchi o marchi registrati dei rispettivi proprietari.

La GLOBO srl governa i processi e i dati all'interno di un sistema integrato di gestione della qualità e della privacy conforme alla normativa UNI EN ISO 9001:2015, al [Decreto legislativo 30/06/2003, n. 196](#) "Codice in materia di protezione dei dati personali" e al [Regolamento \(UE\) 27/04/2016, n. 679](#), "General data protection regulation (GDPR)".

Accogliendo la raccomandazione dell'Unione Europea nell'ambito della certificazione ambientale, che suggerisce di "migliorare l'efficienza ambientale di un'impresa consentendole di avere una conoscenza reale degli aspetti ambientali più rilevanti nella propria attività", questo documento è impaginato per essere stampato in modalità fronte e retro.



## SOMMARIO

0	STORIA DELLE REVISIONI	4
1	SCOPO E CAMPO D'APPLICAZIONE	4
2	TERMINI E DEFINIZIONI	4
3	CONTENUTI	5
3.1	Concetti generali e Flusso applicativo	5
3.1.1	Considerazioni varie	6
3.2	Configurazione della funzionalità e tipologie servizi	6
4	ESEMPI	10
4.1	Creazione di una vista REST	10
4.2	Esempi di Codice	11
4.2.1	Impostazione	11
4.2.2	Codice di esempio	11
4.2.2.1	Costruzione <select> REST	11
4.2.2.2	Costruzione <textarea> REST	12
4.2.2.3	Risultato	12

## 0 STORIA DELLE REVISIONI

Rev.	Data	Redatto	Descrizione
1	19/3/2019	Andrea Cividini	Prima versione del documento
2	6/3/2020	Andrea Cividini	Revisione del documento

## 1 SCOPO E CAMPO D'APPLICAZIONE

Questo documento rappresenta la guida tecnica di riferimento di integrazione REST Services per gli sviluppatori di modulistica compatibile con il portale STU3 di Globo s.r.l. Qualora il popolamento di campi o opzioni selezionabili non possa essere fatto tramite semplice redazione HTML ma necessiti di attingere alla banca dati contenutistica di STU3, questo documento fornisce tutte le specifiche tecniche necessarie per creare gli automatismi di compilazione necessari.

## 2 TERMINI E DEFINIZIONI

- **REST Services:** basandoci sulla definizione tecnica di RESTful API, definiamo all'interno di questo documento che con il termine *Rest Services* si intende la funzionalità di Validator che permette la fruizione di dati provenienti dal backend contenutistico ed il loro inserimento come valori di compilazione del modulo
- **Validator:** libreria Javascript di Globo s.r.l. che permette, tramite regole custom scritte in attributi personalizzati del Modulo Globo, la validazione ed interdipendenza da campi per un supporto alla corretta compilazione da parte dell'utente

- **Modulistica Globo:** i documenti HTML redatti secondo le specifiche di Globo s.r.l. utilizzati per la presentazione di istanze all'amministrazione afferente tramite il portale STU3
- **STU3:** Sportello Telematico Unificato, portale web di Globo s.r.l. alla versione 3 che permette la presentazione di pratiche all'amministrazione tramite la Modulistica Globo, supportata dal motore Validator; basato su Drupal 8
- **Views o viste:** ci riferiamo alle interrogazioni strutturate alla banca dati contenutistica implementate dall'omonimo plugin di Drupal 8; contestualmente a questo documento faremo riferimento quasi sempre all'esportazione JSON dei risultati di queste estrazioni

## 3 CONTENUTI

### 3.1 Concetti generali e Flusso applicativo

La funzionalità REST Services espone allo sviluppatore una API, configurabile tramite degli attributi personalizzati degli elementi HTML del Modulo Globo che si intende integrare, che permette sostanzialmente di soddisfare due necessità:

- Popolare valori di campi in base ai dati esposti da una Vista esportata in JSON REST
- Popolare le opzioni di una select in base ai dati esposti da una Vista esportata in JSON REST

Per fare questo si creano quindi concettualmente tante definizioni di servizi remoti che seguiranno il seguente flusso logico:

- L'invocazione del servizio si collega ad un determinato evento del DOM al caricamento del modulo
- Quando tale evento è invocato, il servizio legge le configurazioni di interazione col backend ed esegue una chiamata verso il portale
- In base allo stato della HTTP si deduce l'esito positivo o meno della richiesta
- Si distribuiscono i valori della risposta JSON nei campi configurati
- Il servizio rimane in ascolto per la prossima occorrenza dell'evento

Da qui si deduce che gli attori in gioco sono i seguenti (Figura 1: Flusso applicativo REST Services)

- Validator con il suo servizio di REST Services, che esegue la chiamata e gestisce i valori di ritorno
- Views, con il plugin di esportazione JSON, che esegue l'interrogazione alla banca dati e struttura la risposta nel formato richiesto

- Drupal, che con la sua definizione delle entità e relative interfacce di amministrazione permette di popolare la sorgente dati interessata dalla modulistica

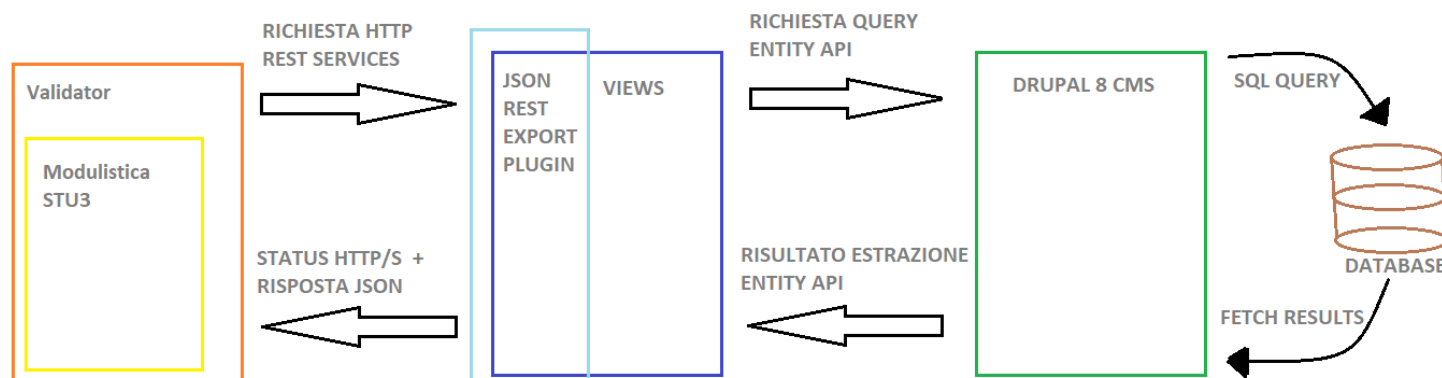


Figura 1: Flusso applicativo REST Services

### 3.1.1 Considerazioni varie

Non è obbligatorio che la risorsa REST consumata dal servizio sia esposta da Views: può essere un generico endpoint che rispetti le regole strutturali dell'oggetto della risposta dopo descritte.

Tuttavia NON è possibile utilizzare un endpoint appartenente ad un dominio differente, in quanto violerebbe la cross domain origin policy.

## 3.2 Configurazione della funzionalità e tipologie servizi

Per ogni risorsa JSON che si intende consumare per la compilazione del modulo è necessario dichiarare un'istanza di REST Service.

È possibile dichiarare potenzialmente infiniti REST Services per ogni modulo; è inoltre possibile che diverse istanze di servizio puntino alla medesima risorsa JSON e/o scrivano gli stessi campi sfruttando, ad esempio, eventi diversi: sta alla accurata analisi del redattore far sì che non vi siano risultati imprevisti – data la natura asincrona dell'invocazione dei diversi servizi.

Ci possono essere due tipi di servizi:

- Servizio di popolamento valori campo (mod. A)
- Servizio di popolamento opzioni di una select (mod. B)

Non è possibile avere un web service ibrido tra le due modalità.

La dichiarazione di un servizio avviene valorizzando gli attributi di un elemento DIV, posizionato a piacimento – all'interno della FORM - nel DOM del modulo; tali attributi sono descritti nella Tabella 1: attributi di configurazione REST Service.

Ad ogni tipologia di servizio corrisponde una regola di composizione della risposta JSON. La struttura richiesta per le due tipologie di servizio è descritta in Tabella 2: Struttura dei payload di risposta.

In caso di assenza di risultati da parte della sorgente dati è opportuno rispondere con esito HTTP positivo (200 OK) e come payload di risposta un oggetto JSON vuoto. In caso di stati HTTP di errore (403, 404, 500 etc etc) Validator mostrerà un popup di errore invitando l'utente a contattare l'amministrazione per verifiche.

Validator NON garantisce la gestione di errori di redazione e/o struttura dei REST Services (es. riferimenti a campi / attributi inesistenti, utilizzo di URL esterni nel rest-name etc etc).

Le chiamate per la consumazione degli endpoint avvengono correlate dei COOKIE di sessione ed autenticazione Drupal: se è interesse del redattore proteggere dall'accesso anonimo la risorsa JSON è possibile farlo.

**Tabella 1: attributi di configurazione REST Service**

Attributo	Possibili valori	Utilizzo mod. A	Utilizzo mod. B
rest-name	Stringa formattata path con placeholders tra quadre  /path/di/esempio/[param]	Rappresenta il path, dalla radice, dell'endpoint REST, a cui mandare la richiesta GET per ottenere i dati; la stringa viene analizzata e tutte le porzioni di testo racchiuse tra parentesi quadre verranno sostituite, prima della chiamata, con il valore del campo il cui name è la stringa estratta dalle parentesi. Es. rest-name="/rest/service/[id_field]" + <input name="id_field" value="5"> = GET /rest/service/5	
rest-type	"select" "text"	"text" dichiara che il payload della risposta JSON sarà utilizzato per valorizzare dei campi	"select" dichiara che il payload della risposta JSON sarà utilizzato per valorizzare le opzioni di una select
rest-param-out	Vedere utilizzo specifico	Deve essere valorizzato con una mappatura degli attributi dell'oggetto restituito dal servizio ai campi di input (by name) che verranno valorizzati, separati da punto e virgola Es. Input1=attrib1;Input2=attrib2  Nel caso la risorsa fornisca una risposta multipla (array di oggetti tra loro uguali strutturalmente) è possibile mappare gli indici degli elementi dell'array di risposta su un valore progressivo che componga diversi nomi di input Es. InputA*=attrib1;InputB*=attrib2;	Deve essere valorizzato con il name del campo select di cui popolare le opzioni Es. rest-param-out="Select_Name"
rest-on	Stringa, formato fisso  "<element>:<evento>"	Collega l'invocazione del ws e successivo popolamento dei campi/opzioni ad uno specifico evento Javascript che occorre su un determinato campo, indicato tramite l'ID dello stesso con tanto di notazione # Es. rest-on="#InputID:change"	
rest-allow-edit	"true" "false" "#Name_campo_input"	"true" e "false" indicano in modo diretto e fisso se il campo che riceve il valore dal REST Service possa essere successivamente modificato dall'utente; se le condizioni di modifica sono variabili e deducibili dalla compilazione degli altri campi, è possibile definire un campo nascosto che verrà valorizzato a "true" o "false" da una logica esterna, ed utilizzato per definire i permessi di modifica. NB: la condizione viene analizzata solamente al momento dell'invocazione del servizio; un eventuale cambio di valore della regola di edit non modifica automaticamente i permessi di interazione con il campo	Inutilizzato, in quanto non ha senso popolare una select senza permettere all'utente una scelta



Ogni payload di risposta è composto da un array JSON ("[]"), popolato da un elenco di oggetti ("{}") ad indici numerici progressivi a partire da 0: array ordinato di oggetti sostanzialmente. Ogni oggetto deve essere equivalente all'altro per elenco e tipologia di attributi, o quantomeno rispettare il set minimo definito in configurazione. Gli attributi della tupla di oggetti dovrebbero contenere stringhe: in caso di valori differenti avverrà il casting implicito a stringa.

**Tabella 2: Struttura dei payload di risposta**

Cardinalità risposta	Gestione mod. A	Gestione mod. B
Singola  [0: {attrib1: 'value1', attrib2: 'value2'}]	Ogni attributo mappato nel campo rest-param-out viene estratto dall'unico oggetto e scritto nell'input associato; eventuali attributi non mappati verranno ignorati	Indifferentemente dalla cardinalità, il servizio di popolamento select scorre ogni elemento dell'array e ne estrae gli attributi value e option: il primo utilizzato come valore dell'option creata, l'altro come label visibile e all'utente.
Multipia  [0: {attrib1: 'value1', attrib2: 'value2'}, 1: {attrib1: 'value3', attrib2: 'value4'}]	Ogni elemento del vettore di risposta viene iterato per estrarne gli attributi mappati dalla definizione del servizio; quindi la sostituzione avviene previa sostituzione del carattere jolly "*" con l'indice numerico - incrementato di 1 - dell'elemento corrente dell'iterazione. Es. rest-param-out="Campo*_Valore=value" + [0: {value: 'a'}, 1: {value: 'b'}] = <input name="Campo1_Valore" value="a"> <input name="Campo2_Valore" value="b">	Es. [{option: "A Letter", value: 'a'}, {option: "B Letter", value: 'b'}] = <option value="a">Letter A</option> <option value="b">Letter B</option>

## 4 ESEMPI

### 4.1 Creazione di una vista REST

Per poter utilizzare questa metodologia è necessario pubblicare una vista REST:

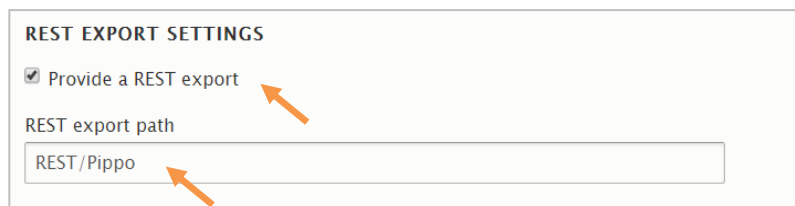
- 1) Creare una Vista



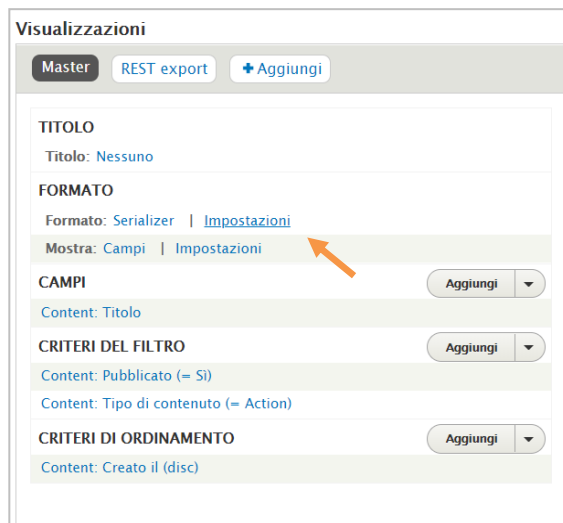
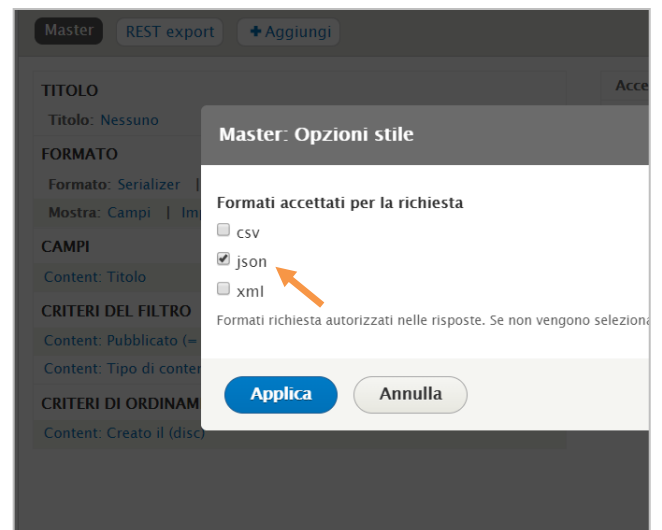
- 2) Selezionare il tipo di contenuto da visualizzare



- 3) Flaggare "Provide a REST export" e assegnare un path



- 4) Impostare il formato JSON

Al termine **[portale]/REST/pippo** sarà raggiungibile.

## 4.2 Esempi di Codice

In questo esempio la guida per dichiarare una Select rest che, dopo aver scelto l'opzione, valorizzerà un secondo elemento HTML (per esempio una textarea)

### 4.2.1 Impostazione

#### PARTE 1 – popolamento della select

- 1) Dichiarare un <div> per la chiamata REST che restituirà un JSON per select  

```
<div rest-name="/REST/[service]/[id_field]" rest-type="select" rest-param-out="[select_name]" rest-on="#[button_name]:[button_event]" rest-allow-edit="[true/false]" />
```
- 2) Dichiarare un bottone nascosto che scatena l'evento di popolamento della select dopo aver ottenuto il JSON  

```
<input id="[button_name]" name="[button_name]" style="display:none" type="button"/>
```
- 3) Dichiarare una select che ospiterà il json fornito dalla chiamata REST  

```
<select name="[select_name]" id="[select_name]" [label] [mandatory] onchange="#[select_name].value()" style="width: 100%;"/></select>
```
- 4) Dichiarare un input hidden che copierà il valore presente nella select  

```
<input id="[select_name]Hidden" name="[select_name]Hidden" style="display:none" dynamic_value="#[select_name].value() != '' ? ([select_name].element.options#[select_name].element.selectedIndex).label) : ''"/>
```

#### PARTE 2 – valorizzazione dinamica di un secondo campo

- 1) Dichiarare un secondo <div> per la chiamata REST che restituirà un JSON per text  

```
<div rest-name="/REST/[service]/[id_field]/[param]" rest-type="text" rest-param-out="[textarea_name]=[param]" rest-on="#[select_name]:[event]" rest-allow-edit="[true/false]" />
```
- 2) Dichiarare una textarea per popolare il testo ottenuto  

```
<textarea class="inputNormal" id="[textarea_name]" name="[textarea_name]" style="width: 100%; height: 5cm;" [mandatory]></textarea>
```

### 4.2.2 Codice di esempio

#### 4.2.2.1 Costruzione <select> REST

```
<div rest-name="/REST/BandiAperti/avviso_mobilita_esterna,avviso_mobilita_interscambio" rest-type="select" rest-param-out="BandoConcorso" rest-on="#ButtonBandoConcorso:click" rest-allow-edit="true"/>
<input id="ButtonBandoConcorso" name="ButtonBandoConcorso" style="display:none" type="button" />
<select name="BandoConcorso" id="BandoConcorso" label="Tipo di partecipazione" mandatory="true" onchange="#BandoConcorso.value()" style="width: 100%;"/></select>
<input class="inputNormal" id="BandoConcorsoHidden" name="BandoConcorsoHidden" style="width: 100%; display:none" dynamic_value="#BandoConcorso.value() != '' ? ([BandoConcorso.element.options#[BandoConcorso.element.selectedIndex].label) : ''"/>
```

## 4.2.2.2 Costruzione <textarea> REST

```
<div rest-name="/REST/BandiAperti/Oggetto/[BandoConcorso]" rest-type="text" rest-param-
out="OggettoBando=Oggetto" rest-on="#BandoConcorso:change" rest-allow-edit="false"/>
<textarea class="inputNormal" id="OggettoBando" name="OggettoBando" style="width: 100%; height: 5cm;" mandatory="true"></textarea>
```

## 4.2.2.3 Risultato

di essere ammesso a partecipare al seguente bando di concorso	
<div> <div>Avviso di selezione per n. 26 tirocinanti di dote comune</div> <div>▼</div> </div>	
<div> <div>Con oggetto</div> <div> <p>È aperta la selezione per n. 26 posti di tirocinio DoteComune da svolgersi in diversi servizi. Per informazioni contattare Luigina Stefanelli 035399533 oppure Sonia Capitanio 035399629</p> </div> </div>	
<div>Tipo</div>	