

UNIVERSITY OF MODENA AND REGGIO EMILIA

INFORMATION ENGINEERING DEPARTMENT

---

DEGREE IN COMPUTER ENGINEERING

**Analysis and Development  
of Functions in Rest Logic:  
Application to the “DataView” Web App**

Advisor:

Char.mo Prof. Sonia Bergamaschi

Candidate:

Alberto Rigenti

---

Academic Year 2009 /2010

UNIVERSITA' DEGLI STUDI DI MODENA E REGGIO EMILIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

---

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

**Analisi e Sviluppo  
di Funzioni in Logica Rest:  
Applicazione alla Web App "DataView"**

Relatore:

Char.mo Prof. Sonia Bergamaschi

Candidato:

Alberto Rigenti

---

Anno Accademico 2009 /2010

Keywords:

Web Application  
Rest Logic  
JSON  
Hibernate  
Struts

# Abstract

There are various types of existing software in the computing landscape. One of these is represented by Web applications, also called WebApps. It is a particular group of applications accessible via web through a network, whether internal or external just like on the Internet.

Taken into account as well as any other software, even the WebApps are characterized by their internal architecture, which is chosen by the developer based on the use of destination and needs.

We are going to talk about the web applications based on REST technology, also called RESTful. REST is an architecture, designed by Roy Fielding, for distributed hypertext systems, among which it is possible undoubtedly to find the well-known World Wide Web.

Through the explanation of this application, the Viewer is going to be presented in order to analyze the great work performed over the training period at the Quix S.r.l. Company. The Viewer is, in fact, a web application which was created as an extension of the DataView<sup>1</sup>.

Briefly introducing the DataView we can say that it is a useful application to make queries. Among the winning features we can mention that it receives input from any data source which might be any kind of DBMS for which exist the JDBC driver to access them (among the ones tested in production we can mention Mysql, SqServer, Oracle, PostgreSQL DB2/AS400). On the data source, you can run the SQL query to display the wished data, and through it, one can define the input parameters, which perform the search filters, output parameters, which in turn are nothing but columns of the SELECT statement that we want to view, and actions, which are activities that may be associated to each record of the outcome or the overall results.

The Viewer was created, as replacement of the DataView default viewer, to display the result related to a "Data View". The results are given by the query and all the above quoted parameters. Unlike the default viewer, the Viewer is stand-alone, independent from the DataView, with all the benefits and problems resulting from that.

Considering this principle as a starting point of development of a web application, we have treated the issues of data persistence and structural approach to its implementation, focusing on the most suitable framework for these purposes. For the first theme, we have chosen to use Hibernate, an open source framework for Object / Relational Mapping (ORM) in Java-based solutions. It takes care of mapping data from the form of Java objects to relational data models (and vice versa). With regard to the structural approach, the framework we have chosen is Struts. Struts is, precisely, the most used Model-View-Controller(MVC) framework to developing J2EE web application.

Going back to the Viewer, among the most frequent problems, there are those relating to transfer data between the two applications, the Viewer and the DataView. For this purpose, it was decided to use JSON, JavaScript Object Notation, which is a suitable format for the exchange of data in

---

<sup>1</sup> For safe of clarity, we will use the denomination "DataView" to mean the application, and the denomination "Data View" to mean the object created by the application itself.

client-server applications. Then we encountered the problems related to display the results of the called Data View whose problems are faced and solved using jQuery and all its plugins. jQuery is the most used JavaScript library on the market (downloadable free from the Internet as well) and it allows us, using a clear and concise syntax, to create web pages of great substance and with a high eye-catching function. This is due to the numerous available plugins, as well as jqGrid is used to display in tabular format our results and jQuery UI, an essential tool for creating graphic effects and customized themes.

# Sommario

Vi sono varie tipologie di software esistenti nel panorama informatico. Una di queste è rappresentata dalle applicazioni Web, dette anche WebApp, ovvero quel gruppo di applicazioni accessibili via web per mezzo di un network, sia esso interno o esterno, vedi Internet.

Come ogni software, anche le WebApp sono caratterizzate da una loro architettura interna, la quale è scelta dallo sviluppatore sulla base dell'utilizzo di destinazione e sulle esigenze dell'utente. Noi parleremo di applicazioni web basate su tecnologia REST, dette anche RESTful. REST è un'architettura, ideata da Roy Fielding, per sistemi di ipertesto distribuiti, tra i quali spicca indubbiamente il conosciutissimo World Wide Web.

Utilizzando questa tipologia di applicazione, sarà presentato il grande lavoro svolto durante il periodo di tirocinio aziendale presso la Quix S.r.l., il Viewer. Il Viewer è, appunto, un'applicazione web che nasce come estensione del DataView<sup>2</sup>.

Introducendo brevemente il DataView possiamo dire che è un'utile applicazione per realizzare interrogazioni, da qui il nome che significa proprio vista dati. Tra le caratteristiche vincenti possiamo menzionare che esso riceve in input una qualsiasi sorgente dati, come possono essere tutti i tipi di DBMS per i quali esiste il driver JDBC per accedervi (tra questi troviamo, testati in fase di produzione: Mysql, SqServer, Oracle, PostgreSQL, DB2/AS400). Su questa sorgente dati si eseguono le query SQL al fine di visualizzare i dati desiderati. Inoltre, sulla stessa sorgente, possiamo andare a definire i parametri di input, che realizzano i filtri di ricerca, i parametri di output, i quali non sono altro che le colonne della SELECT che vogliamo andare a visualizzare, e le azioni, le quali sono attività che possono essere associate a ogni singolo record del risultato o al risultato complessivo.

Il Viewer nasce, in sostituzione al visualizzatore di default del DataView, con lo scopo di visualizzare il risultato ottenuto dalla creazione di un "Data View", cioè il risultato dato dalla query e da tutti i parametri sopra riportati. A dispetto del visualizzatore standard, il Viewer è un'applicazione stand-alone, cioè indipendente dal DataView, con tutti i benefici e i problemi che ne conseguono.

Partendo dal principio dello sviluppo di un'applicazione web, si sono affrontati i temi della persistenza dei dati e dell'approccio strutturale per la sua realizzazione, concentrandosi sui framework più adatti a questi scopi. Per il primo tema ci siamo affidati a Hibernate, framework open source per l'Object/Relational Mapping (ORM) in soluzioni Java-based. Esso si occupa di mappare i dati, sotto forma di oggetti Java, in modelli di dato relazionali (e vice versa). Per quanto riguarda l'approccio strutturale, il framework scelto è Struts. Struts è, appunto, uno dei framework Model-View-Controller (MVC) più utilizzati per lo sviluppo di applicazioni web J2EE.

Tornando al Viewer, tra i problemi riscontrati, troviamo quelli relativi al trasferimento dati tra le due applicazioni, Viewer e DataView. A tal scopo si è scelto di utilizzare il JSON, JavaScript Object Notation, il quale è un formato adatto per lo scambio dei dati in applicazioni client-server.

---

<sup>2</sup> Per chiarezza, useremo la parola "DataView" per intendere l'applicazione e "Data View" per intendere l'oggetto creato dall'applicazione.

Troviamo poi i problemi relativi alla visualizzazione vera e propria dei risultati del data Data View richiamato. Problemi affrontati, e risolti, utilizzando jQuery e tutti i suoi plugin. jQuery è la più utilizzata libreria JavaScript in circolazione, essa ci permette, con l'utilizzo di una sintassi chiara e concisa, di realizzare pagine web di grande sostanza e di grande impatto visivo. Questo grazie anche ai numerosi plugin disponibili, come jqGrid usato per la visualizzazione in formato tabellare dei nostri risultati e jQuery UI, strumento indispensabile per la creazione di effetti grafici e di temi personalizzati.

# Contents

Sintesi .....	12
1. Introduction .....	18
1.1 Goals.....	18
1.2 Thesis Abstract.....	18
2. Adopted Technologies.....	20
2.1 Frameworks .....	20
2.1.1 Hibernate 3.x – Relational Persistence for Java .....	20
2.1.1.1 Architecture.....	20
2.1.1.2 Components .....	21
2.1.2 Struts 1.x.....	24
2.1.2.1 Model-View-Controller.....	24
2.1.2.2 Main Components .....	25
2.1.2.3 How does it work?.....	26
2.2 JSON – JavaScript Object Notation .....	28
2.2.1 Structure .....	28
2.2.2 FlexJSON.....	29
2.2.2.1 Serialization .....	29
2.2.2.2 Deserialization .....	31
2.2.3 JSON in JavaScript .....	31
2.2.4 The Advantages of JSON over XML.....	32
2.3 jQuery.....	33
2.3.1 jQuery UI .....	34
2.3.1 JSON Ajax Calling.....	34
2.3.2 jqGrid .....	35
2.4 The DataView .....	36
2.4.1 Data Connection section.....	36
2.4.2 Data View section .....	37
2.4.2.1 Query .....	40
2.4.2.2 Output Parameters.....	41
2.4.2.3 Input Parameters.....	42
2.4.2.4 Actions.....	43
2.4.3 Charts .....	44



3. Core – Rest Logic and the Viewer .....	46
3.1 REST – Representational State Transfer .....	46
3.1.1 Key Features.....	46
3.1.2 Data Elements.....	47
3.1.3 Connectors .....	48
3.1.4 Components.....	49
3.1.5 REST Process View .....	49
3.1.6 RESTful Web Services .....	49
3.1.6.1 When to Use RESTful? .....	50
3.2 The Viewer .....	52
3.2.1 Project.....	52
3.2.1.1 The Server Side.....	52
3.2.1.2 The Client Side.....	53
3.2.2 Functional Architecture .....	55
3.2.3 User Guide .....	57
3.2.3.1 Starting Configurations .....	58
3.2.3.2 Call the Data View .....	58
3.2.3.3 User Interface - Administrator Toolbar.....	59
3.2.3.4 User Interface - Input Parameters.....	60
3.2.3.5 User Interface - Grid Result Set and Actions .....	63
3.2.3.6 User Interface - Locale and Themes.....	66
4. Conclusions and Future Developments .....	67
Bibliography .....	68
Glossary .....	70

# List of Figures

Figure 1 - Hibernate Architecture .....	20
Figure 2 - persona.hbm.xml .....	21
Figure 3 - hibernate.cfg.xml .....	22
Figure 4 - GenericHibernateDAO.java.....	23
Figure 5 - Struts MVC Architecture .....	24
Figure 6 - struts-config.xml .....	26
Figure 7 - ManagerAction.java .....	26
Figure 8 - Struts Flow Diagram.....	27
Figure 9 - JSON "Basic" of a Data View .....	29
Figure 10 - JSON "Deep" of a Data View .....	30
Figure 11 - Data Connection .....	36
Figure 12 - Data Connection List .....	36
Figure 13 - DataView Project List .....	37
Figure 14 - Data Views List .....	38
Figure 15 - Data View New.....	39
Figure 16 - DataView Query .....	40
Figure 17 - Output Parameters List.....	41
Figure 18 - Input Parameters List.....	42
Figure 19 - Input Parameter .....	42
Figure 20 - Actions List .....	43
Figure 21 - Action .....	44
Figure 22 - Charts List.....	44
Figure 23 - Chart New/Edit .....	45
Figure 24 - Chart Example .....	45
Figure 25 - Activity Diagram (Request Data View Definition).....	55
Figure 26 - Request Data View Definition .....	56
Figure 27 - The Viewer (Overview) .....	57
Figure 28 - The Viewer (Administrator Toolbar).....	59
Figure 29 - The Viewer (Cache Cleared).....	59
Figure 30 - The Viewer (Parsed Query) .....	60
Figure 31 - The Viewer (Input Parameters).....	60
Figure 32 - The Viewer (Input Parameters Advanced).....	61
Figure 33 - The Viewer (Input Parameters Default).....	61
Figure 34 - The Viewer (Input Parameter Validation).....	62
Figure 35 - The Viewer (Input Parameter DropDownList) .....	62
Figure 36 - The Viewer (Input Parameter DropDownList Instant Search) .....	62
Figure 37 - The Viewer (Input Parameter CheckBox) .....	62
Figure 38 - The Viewer (Input Parameter PopUp) .....	63
Figure 39 - The Viewer (Grid Result Set & Actions) .....	63
Figure 40 - The Viewer (Grid Toolbar) .....	64
Figure 41 - The Viewer (Output Parameter Headers).....	64
Figure 42 - The Viewer (Style Cell) .....	64
Figure 43 - The Viewer (Actions).....	64
Figure 44 - The Viewer (Total).....	65

Figure 45 - The Viewer (Pager).....	65
Figure 46 - The Viewer (Locale).....	66
Figure 47 - The Viewer (Alternative Theme).....	66

# Sintesi

Lo scopo principale di questa tesi è di mostrare, in dettaglio, le tecnologie utilizzate nello sviluppo di un'applicazione web, concentrandosi particolarmente sull'architettura impiegata. In questo caso specifico andremo ad analizzare le soluzioni scelte al fine di progettare e sviluppare il Viewer, il quale verrà poi illustrato nella sua struttura e nel suo funzionamento. Il Viewer è un'applicazione web, frutto del lavoro svolto durante il periodo di tirocinio aziendale presso la Quix S.r.l. di Soliera.

Per prima cosa introduciamo le tecnologie, esse sono state suddivise in quattro argomenti principali: Framework, JSON, jQuery e DataView.

## Framework

Un framework può essere visto come un insieme di software, documentazioni, politiche e procedure che supportano la realizzazione di elementi software di alto livello. Si è scelto di utilizzare due framework nello specifico: Hibernate e Struts.

Hibernate è un Object Relational Mapping (ORM), per ambienti Java. Il termine Object Relational Mapping si riferisce alla tecnica di mappatura dei dati da una rappresentazione a oggetti a una rappresentazione dei dati nel modello relazionale (e viceversa). Questo è indispensabile quando si sviluppano applicazioni Java-based che devono leggere, e/o scrivere, dati da, e su, un DBMS.

Struts è uno dei più utilizzati framework Model-View-Controller(MVC) nello sviluppo di applicazioni web J2EE. Esso ci fornisce tre componenti chiave:

- Un gestore di richiesta(request), fornito dallo sviluppatore dell'applicazione, il quale viene utilizzato per mappare verso una URI specifica;
- Un gestore di risposta(response), il quale viene utilizzato per trasferire il controllo a un'altra risorsa, responsabile per il completamento della risposta;
- Una libreria di tag(tag library), che aiuta lo sviluppatore nel creare applicazioni basate su form interattivi attraverso pagine server(come possono essere le JSP).

In un modello MCV, un'applicazione è suddivisa in tre parti distinte. Il dominio è rappresentato dal Model, l'output verso l'utente è rappresentato dal View e, l'input da parte dell'utente, è rappresentato dal Controller.

## JSON

Quando si progetta un'applicazione che comunica con computer remoto, si deve scegliere un formato dati e un protocollo per lo scambio di questi. JSON(JavaScript Object Notation) è un formato, leggero, per lo scambio di dati, basato su un sottoinsieme del linguaggio di programmazione JavaScript. È facile per gli umani da leggere e scrivere ed è facile per le macchine da analizzare e generare. JSON è un formato di testo completamente indipendente sotto il punto di vista del linguaggio, ma usa convenzioni famigliari ai programmatori che utilizzano linguaggi della sfera C, incluso C, C++, Java, JavaScript e molti altri. Questi aspetti, fanno del JSON un formato ideale per l'interscambio dei dati.

### jQuery

“jQuery è una libreria JavaScript veloce e concisa che semplifica lo scorrimento dei documenti HTML, la gestione degli eventi, le animazioni e le interazioni Ajax, per uno sviluppo web più veloce.” Questa è la definizione data da John Resig, il creatore di jQuery.

Questa libreria è progettata per mantenere ogni componente semplice e riutilizzabile, inoltre fornisce una vasta gamma di funzioni e metodi, semplici da utilizzare, per creare ottime applicazioni. Poiché è basata su JavaScript, può essere utilizzata con JSP, Servlets, ASP, PHP, CGI, e con quasi tutti gli altri linguaggi di programmazione.

### DataView

Il DataView è un’applicazione web, sviluppata dall’ing. Roberto Pellegrino per conto di Quix, nata con lo scopo di effettuare interrogazioni sui dati. La sua peculiarità consiste nel ricevere i suddetti dati, sui quali andremo ad effettuare le interrogazioni, da una qualsiasi sorgente dati(Data Source), senza conoscere tipologia della sorgente o degli stessi dati. Una sorgente dati può essere rappresentata, per esempio, da un qualsiasi DBMB per il quale esiste il driver JDBC per accedervi. Sulla sorgente ottenuta, l’utente potrà andare a creare uno o più “Data View”. Il Data View è un elemento composto dalla nostra query SQL e da altri componenti, i quali sono:

- Output Parameters: essi rappresentano i campi che vogliamo visualizzare nel nostro risultato finale, non sono altro che le colonne della SELECT eseguita. Per ogni Output Parameter possiamo definire nome, etichetta, tipo(stringa, intero, data, decimale), formato(ovvero il pattern, che per una data potrebbe essere: gg/mm/aaaa), ordinabilità(comprensente l’ordine desiderato, crescente o decrescente), stile(come può essere colore o tipo del carattere), sommatoria(ovvero se vogliamo visualizzare il totale di questa colonna) e infine la visualizzazione(ovvero se vogliamo che l’Output Parameter sia visibile o meno);
- Input Parameters: rappresentano i filtri della nostra query, permettono, infatti, di realizzare query condizionali e parametriche. Per ogni Input Parameter si devono definire nome, tipo(stringa, intero, data, decimale), etichetta, se è obbligatorio, se è avanzato(verrà posizionato in una parte inizialmente nascosta che può essere mostrata tramite l’apposito comando), un valore di default(se lo vogliamo) e la modalità di visualizzazione/inserimento. Quest’ultimo parametro può essere scelto tra: default(una semplice input box), dropDownList(la classica lista a tendina), checkBox, PopUp(apre una finestra contenente un Data View da cui selezionare il valore) e Custom(ovvero personalizzato dall’utente).
- Actions: sono attività che possono essere associate a ogni singolo record del risultato o al risultato complessivo(in questo caso vengono dette globali). Sono identificate da nome, tipo(Action URL, Action JavaScript, Action Communication), contesto dinamico(inserisce il context path prima dell’URL dell’azione), azione globale, PopUp, icona, filtri di Input, campi di output e condizione.

Come vedremo tra poco, è su questa applicazione che si basa il Viewer.

Il secondo macro argomento della tesi è costituito dalla logica REST, ovvero l’architettura alla base del Viewer e dallo stesso Viewer, del quale ne viene illustrato il progetto, l’architettura funzionale e il manuale utente.

## REST

Nel mondo dei servizi web, Representational State Transfer (REST) rappresenta una tipologia di architettura software basata su un modello client-server stateless(ovvero senza stato), nella quale i servizi web sono visti come risorse(resources) e possono essere identificati dalla loro URL.

Parlando in termini di applicazioni web, utilizzando questo tipo di architettura, possiamo dire che un client effettua una richiesta(request) per una risorsa(resource) situata in un server, il quale a sua volta invia una risposta(response) al client, tale risposta include in se una rappresentazione della risorsa.

Un'applicazione web utilizza il protocollo HTTP(Hypertext Transfer Protocol), che è un protocollo senza stato basato su di un client, il quale richiede una risorsa in rete, e su di un server, che fornisce una risposta. Una transazione HTTP comporta quindi una richiesta e una risposta.

Una richiesta HTTP è suddivisa in tre parti:

- Una linea per le richieste, che include il metodo http, l'URI e la versione HTTP;
- Una o più intestazioni HTTP, facoltative, costituite da coppie chiave-valore che caratterizzano i dati richiesti e/o previsti.
- Un corpo del messaggio, opzionale, composto dai dati inviati dal client al server come parte della richiesta.

Una risposta HTTP è, anch'essa, suddivisa in tre parti:

- Il codice di stato HTTP, che indica lo stato dell'URI richiesto.
- Una o più intestazioni HTTP, facoltative, costituite da coppie chiave-valore che caratterizzano i dati che vengono forniti;
- Un corpo del messaggio, opzionale, composto dai dati che vengono restituiti al client in risposta alla sua richiesta.

Ogni URI punta ad una risorsa sul server e ne è presente uno per ogni risorsa.

HTTP definisce diversi metodi per agire su una risorsa: HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT e PATCH. Tuttavia parleremo solo di GET e POST, i più utilizzati.

Una semplice pagina web è un esempio di risorsa REST. Quando accediamo alla risorsa, attraverso il nostro browser, viene inviata una richiesta GET, non visibile. GET si utilizza per recuperare una risorsa e rappresenta l'azione più comune utilizzata in ambito REST. L'altra azione, molto popolare, è POST, la quale viene utilizzata per inviare i dati da elaborare al server.

Parlando del formato nel quale si presentano i dati, che riceviamo da risposte HTTP, possiamo dire che ve ne sono di diversi tipi. Un web server tradizionale consegna i suoi dati in formato HTML, con i relativi JavaScript, CSS e file di immagine. HTML è un ottimo formato, ma in questo contesto si nota la sua limitata potenza espressiva per strutturare i dati, che non siano semplici documenti. I formati, più comuni, utilizzati per trasmettere dati strutturati attraverso HTTP sono l'XML e il JSON. È chiaro come si sia scelto per la nostra applicazione di utilizzare il JSON.

L'implementazione più conosciuta di un sistema, basato su architettura REST, è rappresentata dal World Wide Web.

## Il Viewer

Il Viewer è un'applicazione web, basata su architettura REST, che nasce come estensione del DataView, in particolare si occupa della parte di visualizzazione di un Data View. Il Viewer ha come caratteristica principale di essere un'applicazione stand-alone, ovvero completamente separata ed indipendente dal DataView. Tra i vantaggi principali, dell'essere indipendente, vi è quello di non dipendere dalla versione del DataView installata. Inoltre eventuali migrazioni del DataView non comporteranno alcun tipo di modifica nel Viewer nel caso in cui l'interfaccia di comunicazione resti invariata.

La comunicazione tra il Viewer e il DataView avviene attraverso l'invio di dati da parte del server del DataView, verso il server e il client del Viewer, in formato JSON. È interessante notare come al Viewer non interessi il modo in cui queste informazioni vengano recuperate e generate, ma si preoccupa solo di leggerle ed interpretarle.

Per quanto riguarda l'aspetto della sicurezza, il Viewer non è affetto da nessun tipo di problematica, in quanto è vero sì che si tratta di un'applicazione indipendente dal DataView e vi è tra loro uno scambio di dati, ma è anche vero che entrambi sono installati nella stessa locazione e la loro sicurezza è assicurata dall'uso del CAS(Central Authentication Server). Il CAS è un protocollo SSO(single sign-on) per il web e, il suo scopo consiste nel permettere ad un utente di accedere a più applicazioni, dotate di sistema di accesso tramite credenziali, inserendo quest'ultime una sola volta.

L'unico svantaggio che si presenta dalla separazione tra Viewer e DataView è rappresentato dallo scambio dei dati, non necessario nel visualizzatore di default del DataView. Svantaggio in termini di prestazioni, che si è cercato di colmare immagazzinando la definizione del DataView, richiamato dal Viewer, in cache.

L'aspetto da ricordare, sempre, è che, come il DataView non conosce nulla sulla sorgente e sui dati in input, così anche il Viewer non conosce minimamente quanti dati, quali e di che tipo arriveranno. Questo è fondamentale nel comprendere e capire la difficoltà nello sviluppo di un'applicazione che si basa sull'utilizzo di pochissimi riferimenti e che, ad ogni chiamata, genera una pagina completamente differente dalla precedente.

Per la realizzazione di quest'applicazione si è dovuto modificare la parte server del DataView, in modo che restituisse i dati richiesti in formato JSON, formattati al fine di renderli facilmente riutilizzabili dal Viewer. Il server del DataView restituisce la definizione del Data View richiesto, i risultati della query parsata, i parametri di input complessi(DropDownList, CheckBox e PopUp) e il file, in formato .xls, contenente i risultati della query parsata.

Per quanto riguarda l'applicazione vera propria, essa è suddivisa in due parti: server e client.

La parte server è composta da tutte quelle azioni necessarie al recupero dei dati dal DataView in unione alle azioni per gestire il riempimento e lo svuotamento della cache.

La parte client è composta da due pagine JSP, la prima richiede al server del DataView la definizione, qualora essa non sia stata trovata in cache. La seconda è la vera e propria interfaccia utente, su di essa si concentra la gran parte del lavoro svolto, utilizzando una quantità considerevole di codice Java, JavaScript, Html e CSS. Viene disegnata dinamicamente ogni volta che richiediamo un Data View, ed è composta da 3 sezioni principali:

- Toolbar Amministratore: è visibile solo nel caso in cui l'utente loggato, abbia i permessi di amministratore. In essa troviamo un'azione che ci permette di visualizzare la query parsata dal server del DataView e, una per svuotare la cache del Viewer, utile soprattutto nel caso si siano apportate modifiche alla definizione del DataView. Quest'ultima azione svuota la cache inviando via AJAX il comando al server e, solo dopo aver ricevuto un esito positivo sull'operazione, riavvia l'applicazione.
- Parametri di Input: costituiscono il form di ricerca. Possono non avere nessun valore di default, oppure averne, e sono inizialmente divisi in due categorie principali: standard ed avanzati. Se un parametro è "avanzato", viene inserito in una parte inizialmente nascosta, la quale può essere mostrata/nascosta attraverso l'apposito bottone. I parametri di input vengono rappresentati in varie forme grafiche:
  - Default. Una semplice input box dove possiamo digitare il valore desiderato. In questa modalità, di default, se il filtro è di tipo intero, decimale o data, abbiamo il supporto del validatore. Se andiamo a digitare in un campo un valore di tipo diverso da quello definito, ci verrà mostrato un messaggio che ci avverte dell'errore.
  - DropDownList e le CheckBox. Entrambe le strutture sono state modificate, rispetto a quelle di default, al fine di migliorarne l'usabilità e l'efficienza. Nel caso delle DropDownList, oltre alla classica tendina, dove possiamo selezionare il valore, abbiamo la possibilità di utilizzare l'Instant Search. Ovvero digitando i caratteri iniziali, del valore ricercato, tutti i valori presenti vengono filtrati, e ci vengono mostrati solamente quelli che corrispondono alla nostra ricerca. Nel caso delle CheckBox, esse sono state raggruppate in una lista, così da non occupare uno spazio eccessivo in caso di un numero elevato. Il numero di parametri selezionati ci viene mostrato a video e, al termine della selezione, i valori vengono riportati a fianco, congiuntamente ad un bottone che ci permette di deselezionarli con più facilità.
  - PopUp. Agendo sull'apposito bottone viene aperta una nuova finestra contenente un DataView, differente da quello in uso, sul quale, selezionando la riga corrispondente al valore desiderato, riporta tale valore in una label appartenente al nostro parametro di Input.

Ovviamente è presente un pulsante *Cerca*, con il quale andremo a richiamare un nuova query al server del DataView, passandogli i parametri inseriti. Con questa azione si ricaricheranno solamente i dati interni della griglia e non l'intera pagina, questo grazie all'uso di chiamate AJAX. È presente anche un pulsante *Pulisci*, con il quale ripristineremo la situazione iniziale dei parametri di Input e dei risultati mostrati nella griglia.

- Parametri di Output e Azioni. Sono rappresentati da una tabella dove troviamo, inizialmente, il nome del nostro DataView, le azioni Globali e l'azione Excel. A seguire troviamo le intestazioni dei parametri di Output, costituite dal nome, e differenziate tra loro in base al fatto che il nome sia o meno sottolineato. Se un'intestazione è sottolineata,



allora la griglia è ordinabile, in modo crescente o decrescente, secondo quel parametro. Nella parte centrale troviamo i risultati cercati e, se presenti, le azioni associate ad ogni riga. Infine troviamo il paginatore. In esso ci viene riportato il range dei record visualizzati al momento, rispetto a quelli presenti in totale, e gli appositi pulsanti di navigazione della tabella, in aggiunta ad una casella di testo contenente il numero della pagina, dei risultati, corrente, e ad una label contenente il numero delle pagine totali.

Se il numero di record fosse settato a  $-1$  (tutti), oppure il numero settato fosse minore o uguale di quello dei record presenti, questo modulo centrale del paginatore non sarebbe visibile, in quanto inutile data la presenza di una sola pagina.

In caso di campi calcolati, visualizzabili solo se tutti i record sono contenuti in una pagina, una riga riportante l'intestazione "TOTALI" verrebbe visualizzata alla fine della griglia, congiuntamente ad una riga contenente i totali dei parametri di output interessati.

Aspetto molto interessante del Viewer, e di conseguenza del DataView, è il fatto di supportare completamente il multilingua. Questo significa che se andiamo a specificare, nella chiamata iniziale, l'opzione *language* e passiamo come parametro l'abbreviazione *en*, l'intera applicazione, risultati a parte ovviamente, verrà tradotta in lingua inglese.

# 1 Introduction

## 1.1 Goals

The main purpose of this thesis is to illustrate what is an application based on the REST architecture and how to develop it. Obviously, this includes all matters concerning the choice of this architecture and the technology to apply.

All the choices made, including the frameworks up to the choice of the format for data interchange, have been conceived after careful evaluation of various aspects, from performance to security. These two issues are of primary importance and a developer should always consider them when the software is analyzed and designed.

We also want to show in detail the application derived from these studies, which combines all the issues arising from the above cited goals. We will certainly show its structure and its purpose, but we also provide a small user guide on it. This is going to help us understanding the difficulty that comes from developing an application with the Viewer's features, and the ways used to overcome them.

## 1.2 Thesis Abstract

As already mentioned in the previous abstract, this thesis will treat the main themes about the development of a Web Application based on REST technology, then it will illustrate the web application developed.

The thesis is divided into the following chapters.

CHAPTER 2. In this chapter, the technologies which have been used will be exposed, highlighting the key points distinguishing them. Included in these points we are going to analyze the Hibernate and Struts frameworks. Hibernate is used for data persistence on the DBMS, and Struts is one of the most used Model-View-Controller (MVC) framework for developing J2EE web applications. We are then going to discuss the data format chosen for the transfers in the application, the JSON, arguing the reasons for its choice. Secondly, an overview about jQuery's main features will be given that is to say the most popular JavaScript library in use today. In the last section of this chapter, we will learn the DataView, the web application which my application extends. The DataView has the task to run custom queries on any given input data source, including DBMS, and give the result back. The characteristic that distinguishes it, is the possibility for the user to define Input Parameters, Output Parameters and Actions.

CHAPTER 3. This is the main chapter of the thesis. It speaks about the REST logics and about the developed web application, the Viewer. REST logics is a feature apt to develop applications for distributed hypermedia system, and in this section we are going to explain which its components are and how it works. The most important example using this kind of technology is the World Wide Web.

Finally, we will talk about the Viewer. The Viewer is a web application which takes in input data generated by the DataView, in JSON format, and dynamically generates a JSP page using the data themselves. As far as the Viewer, we will see the project, the functional architecture and a small user manual.

## 2 Adopted Technologies

### 2.1 Frameworks

This section describes the salient features of frameworks used to develop the application.

A Framework can be seen as a set of software(compile/run-time elements, libraries, tools, services), documentation, policies, procedures, that supports the implementation of specific higher level software elements. It performs several tasks:

- It makes easier to work with complex technologies;
- It ties together a bunch of discrete object/components into something more useful;
- It forces the team to implement code in a way which promotes consistent coding, fewer bugs, and more flexible applications;
- Everyone can easily test and debug the code, included the one which has not been written by others.

#### 2.1.1 Hibernate 3.x – Relational Persistence for Java

Working with both Object-Oriented software and Relational Databases can be cumbersome and time consuming. Development costs are significantly higher due to a paradigm mismatch between how data is represented in objects versus relational databases. Hibernate is an Object/Relational Mapping(ORM) solution for Java environments. The term Object/Relational Mapping refers to the technique of mapping data from an object model representation up to a relational data model representation(and vice versa).

Hibernate not only takes care of the mapping from Java classes to database tables (and from Java data types to SQL data types), but also provides data query and retrieval facilities. However, Hibernate does not deny the possibility of using the SQL language.

##### 2.1.1.1 Architecture

The diagram below provides a high-level view of the Hibernate architecture.

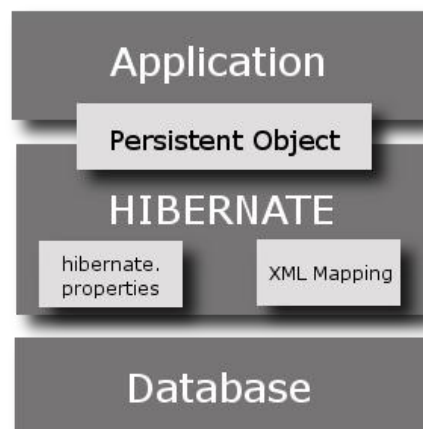


Figure 1 - Hibernate Architecture

### 2.1.1.2 Components

In this paragraph we are talking about the main components forming Hibernate, therefore, there will be the explanation about how Hibernate needs to work and which are the key points to understand how it works.

**Mapping File (*name.hbm.xml*):** Hibernate needs to know where loading and storing objects of the persistent class. This information is provided by the mapping file, which tells Hibernate which table in the database it has to access, and which columns in that table it should be used.

Between the two *hibernate-mapping* tags, a *class* element must be included; all the persistent entity classes need a mapping to a table in the SQL database. Each instance is now represented by a row in that table.

The *id* element and the *name* attribute of the *property* element tells Hibernate which *getter* and *setter* methods must be used.

The *type* mapping also lacks a *type* attribute. The types declared and used in the mapping files are not Java (or SQL database) data types. These ones are called "Hibernate mapping types", converters which can translate from Java into SQL data types and vice versa. If the *type* attribute is not present, Hibernate will try to determine the correct conversion and mapping type itself.

```
<hibernate-mapping package="it.ar.mytest.model">
  <class name="Persona" table="persona">

    <id name="id" column="id" type="long" length="10">
      <generator class="assigned"></generator>
    </id>

    <property name="nomeP" type="string" column="nomeP" length="45"/>
    <property name="cognomeP" type="string" column="cognomeP" length="45"/>

    <list name="indirizzoList" table="indirizzo">
      <key column="persona_id" foreign-key="FK" />
      <composite-element class="Indirizzo">
        <property name="via" type="string" column="via" length="45"/>
        <property name="cap" type="long" column="cap" length="5"/>
      </composite-element>
    </list>

  </class>
</hibernate-mapping>
```

Figure 2 - persona.hbm.xml

**Configuration File (*hibernate.cfg.xml*):** Hibernate will be connecting to the database on behalf of your application, therefore it needs to know how to obtain connections. Hibernate comes with *built in* connection pool and also has a support for the two third-part open source JDBC connection pools: c3p0<sup>3</sup> and proxool<sup>4</sup>.

Through this file, it is possible to configure Hibernate's Session Factory: it is a global factory responsible for a particular database (in the next paragraph, what Session Factory is will be more deeply explained).

The first *property* elements contain the necessary configuration for the connection.

<sup>3</sup> <https://sourceforge.net/projects/c3p0>

<sup>4</sup> <http://proxool.sourceforge.net>

```

<hibernate-configuration>

  <session-factory>

    <!-- Database connection settings -->
    <property name="connection.datasource">java:comp/env/jdbc/mytest</property>

    <!-- Enable Hibernate's automatic session context management -->
    <property name="current_session_context_class">thread</property>

    <!-- Enable the second-level cache -->
    <property name="cache.provider_class">org.hibernate.cache.EhCacheProvider</property>

    <!-- Echo all executed SQL to stdout -->
    <property name="show_sql">TRUE</property>

    <!-- Pretty print the SQL in the log and console --!>
    <property name="format_sql">>true</property>

    <property name="hibernate.bytecode.use_reflection_optimizer">>false</property>

    <!-- mapping files -->
    <mapping resource="it/ar/mytest/model/Persona.hbm.xml"/>

  </session-factory>

</hibernate-configuration>

```

Figure 3 - hibernate.cfg.xml

**Java Start-Up Class.** It takes care of start-up and makes accessing the *org.hibernate.SessionFactory* more convenient. A SessionFactory is designed to represent a single unit of work and is the central point for the development in Hibernate. The SessionFactory loads the information specific for the current application context from the configuration file(*hibernate.cfg.xml*).

The *configure()* method of the Configuration object reads configuration information from the classpath and therefore building an instance of SessionFactory. The SessionFactory is normally made available as static in an application, in fact the purpose of SessionFactory is to provide new session instances of Hibernate to the caller via *openSession()* method, so there is no problem of concurrent use of SessionFactory.

**Objects and Main Methods.** The life cycle of objects in Hibernate and the related methods will be taken into account. Firstly, we have to retrieve the Session with the *getSession()* method, which returns a Hibernate session that is linked to the current JTA transaction, and this requires that a JTA transaction exists prior to call.

Now, it is possible to introduce the objects in Hibernate. An object can take four basic stages: Transient, Persistent, Removed and Detached.

**Transient** is the state where you find objects that are instantiated through a new operation: such objects are not associated with a Hibernate session, do not have a persistent representation within the database of reference and therefore do not possess any unique identifier. Items in transient state(or volatile) are considered to be non-transactional by Hibernate and JPA, so any change to the internal state of an object of this type is not considered by the persistence context.

**Persistent** is the state in which there is a candidate object which as to be kept in the database. It is defined candidate because Hibernate applies the pattern *transparent transaction-level*

*write-behind* (TTW) which is trying to delay as much as possible all the synchronization operations necessary to align the internal state of objects with data in the database. The goal is to optimize and minimize the generated queries and lessen the lock probability needed to ensure the consistency within a transaction. A persistent object is always associated with the persistent context and can be considered to all effects a memory copy of data stored in a database.

At any time, you can always delete one instance of an object in different ways. An object is in the **removed** state if any deleted operation has been previously performed on the unit of work, but the object itself is still associated with the persistence context until the unit of work is not terminated. Changing any of the properties of the object means to take it back into a state of persistent and dirty and then only runs an *Update*, and no more a *Delete*. The object, whose corresponding row is deleted on the database, enters a state of detached(end session) and remains alive until it is deleted by the garbage collector and therefore if it is hung, with a *save()* operation, to a new session, this leads the generation of an *Insert*.

An object is **detached** (also called disconnected) if it is obtained during a search or a Creation executed within a transaction, but the transaction must have already been accomplished. You can perform further operations on the object, but none will be made persistent.

The operation of sync with the persistence engine is called flushing(literally emptying the buffer) and that operation is performed in the following cases:

- Upon completion of a transaction when executing a commit.
- Before a query is performed;
- When the *flush()* method is explicitly invoked.

The flush, at the end of a unit of work,(when a session must end, or when a transaction should be submitted to commit operation) ensures that data are synchronized and runs automatically when a work unit needs to make persistent change.

```

public Session getSession() {
    return HibernateSessionLocator.getSession();
}

public void clear() {
    getSession().clear();
}

public void flush() {
    getSession().flush();
}

public void makeTransient(T entity) {
    getSession().delete(entity);
}

public Class<T> getPersistentClass() {
    return persistentClass;
}

```

Figure 4 - GenericHibernateDAO.java

For further information, consult Hibernate reference documentation<sup>5</sup>.

---

<sup>5</sup> <http://www.hibernate.org/docs>

## 2.1.2 Struts 1.x

Struts is one of the most used Model-View-Controller (MVC) framework for developing J2EE web applications, and it provides three key components:

- A **request** handler provided by the application developer is used to map to a particular URI.
- A **response** handler which is used to transfer the control to another resource will be responsible for completing the response.
- A **tag library** which helps developers to create the interactive form based applications with server pages.

### 2.1.2.1 Model-View-Controller

Under MVC, an application is seen as having three distinct parts. The domain is represented by the Model, the output to the user is represented by the View and, the input from the user is represented by Controller.

Below, it is going to be illustrated the MVC model according to Struts.

The Model(System State and Business Logic JavaBeans). The Model components provide a model of the business logics behind a Struts program. It provides interfaces to databases or back-end systems. Model components are generally a java class. There is not any such a defined format for a Model component so it is possible for us to reuse the Java code which is written for other projects. We should choose the model according to our client requirement.

The View(JSP Pages and Presentation Components). The View components are responsible for presenting information provided by the Model components. Mostly, we use the Java Server Pages (JSP) for the view presentation. To extend the capability of the view, we can use the Custom tags, JavaScript or others.

The Controller(ActionServlet and ActionMapping). Whenever a user request for something, the request is handled by the Struts ActionServlet. When the ActionServlet receives the request, it intercepts the URL and based on the Struts Configuration files, it gives the handling of the request to the *Action* class. Action class is a part of the controller and is responsible for communicating to the model layer.

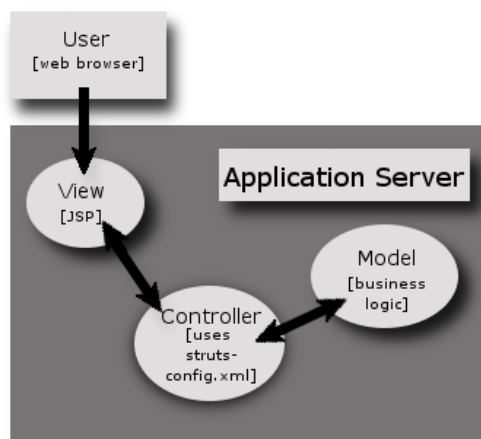


Figure 5 - Struts MVC Architecture



### 2.1.2.2 Main Components

ActionServlet. It is the central component of the Struts controller. This servlet extends the *HttpServlet* and it basically performs two important tasks:

1. When the container gets start, it reads the Struts Configuration files and loads it into memory in the *init()* methods;
2. It intercepts the HTTP request in the *doGet()* and *doPost()* method and handles it properly.

Configuration File(*struts.config.xml*). It is an XML document which describes all or part of Struts application. This file has all the information about many types of Struts resources and configures their interaction. This file is placed under *WEB-INF* directory of the web application.

Action. Action class acts as wrapper around the business logics and provides an interface to the application's Model layer. An Action works as an adapter between the contents of an incoming HTTP request and the business logics corresponding to it. Consequently the struts controller (ActionServlet) selects an appropriate Action and creates an instance if necessary, and finally it calls execute methods.

ActionMapping. It is a configuration file entry which usually contains a reference to an Action class. This entry can contain a reference to a form bean that the action can use, and can also define local forwards and exceptions which are visible only to this action.

ActionForm. An ActionForm is a JavaBean which extends *org.apache.struts.action.ActionForm*. ActionForm maintains the session state for web application and the ActionForm object is automatically populated on the server side with data entered from a form on the client side.

ActionForward. The ActionForward class represents a destination to which the controller may send control once an action has been completed.

Custom-tags. A custom-tag is a tag that it is possible to write in your Java Server Page. When the Jsp compiler encounters that tag, it knows what to do with it. It generates the proper HTML after doing whatever it is supposed to do. Struts will be provided by a set of libraries containing its custom-tags.

### 2.1.2.3 How does it work?

1. Each application we develop has a deployment descriptor i.e. *WEB-INF/web.xml*. This is the file that the container reads. This file has all the configuration information that we have defined for our web application. The configuration information includes the index file, the default page, the mapping of our servlets including path and the extension name, any init parameters, information related to the context elements. In the *WEB-INF/web.xml* of Struts application, we need to configure the Struts ActionServlet which handles all the requests made by the web browsers to a given mapping.
2. In Struts application, we have another xml file which is a Struts configuration file named as *struts.config.xml*. The name of this file can be changed and can be configured in the *web.xml* file. This file is used to associate paths to the controller components of our application, known as Action classes like `<action path="/json" type="JsonAction">`. This tag tells the Struts ActionServlet that whenever the incoming request is `http://myhost/simple/showDv.do`, then it must invoke the controller component `JsonAction`. In the previous line, it is possible to notice that we have written `.do` in the URL. This mapping is done to tell the web application that whenever a request is received with `.do` extensions then it should be appended to the URL.

```
<struts-config>

  <action-mappings>

    <action path="/json" type="it.ar.mytest.action.JsonAction" parameter="task">
      <forward name="jsonDvDeserializer" path="/pages/DataView.jsp" />
      <forward name="showDv" path="/pages/ShowDv.jsp" />
    </action>

  </action-mappings>

</struts-config>
```

Figure 6 - struts-config.xml

3. For each action we also have to configure Struts with the names of resulting pages that will be shown as a result of that action. In our application there can be more than one view which depends on the result of an action. The struts knows how to forward the specific page to the concerned destination.

```
public class JsonAction extends DispatchAction {

  /** Show Data View */
  public ActionForward showDv(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response) throws Exception {

  /** Deserialize Data View */
  public ActionForward jsonDvDeserializer(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response) throws Exception {

  /** Clear DataView Cache */
  public ActionForward dataViewCacheClear(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response) throws Exception {

}
```

Figure 7 - ManagerAction.java

4. Action can also be associated to a JavaBean in our Struts configuration file. Java bean is nothing but a class having *getter* and *setter* methods which can be used to communicate between the view and the controller layer. These java beans are validate by invoking the *validate()* method on the ActionForm by the help of the Struts system. The client sends the request using the normal form submission and by using *Get* or *Post* method, therefore the Struts system updates the data in the Bean before calling the controller components.
5. The View we use in the struts can be either Jsp page or other. In struts there are sets of JSP tags which have been bundled with the struts distribution, but it is not mandatory to use only Jsp tags, even plain HTML files can be used within our Struts application but the disadvantage of using the HTML is that it cannot take the full advantage of all the dynamic features provided in the struts framework.

The framework includes a set of custom tag libraries which facilitate the user in creating interfaces which can interact gracefully with ActionForm beans. The struts Jsp tag libs have a number of generic and struts specific tags which helps you to use dynamic data in our view. These tags help us to interact with our controller without writing much Java code inside our Jsp.

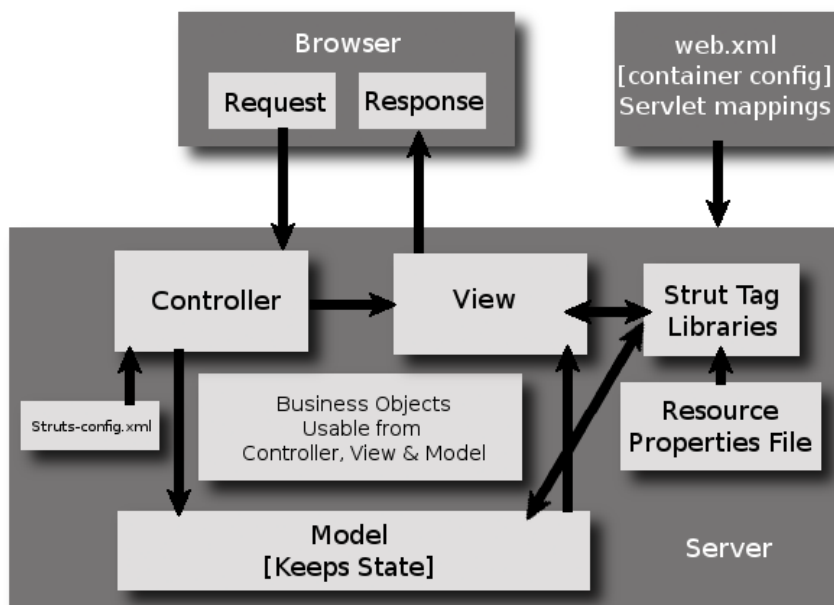


Figure 8 - Struts Flow Diagram

## 2.2 JSON – JavaScript Object Notation

When designing an application which will communicate with a remote computer, a data format and exchange protocol must be selected. There are a variety of open, standardized options, and the ideal choice depends on the applications requirements and pre-existing functionality.

JSON(JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write and it is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3<sup>rd</sup> Edition – December 1999. JSON is a text format which is completely language independent but it uses conventions which are familiar to programmers of the C-family of languages, including C, C++, Java, JavaScript and many others. These make JSON an ideal data-interchange language.

### 2.2.1 Structure

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures and in JSON they take on these forms:

**Object.** An object is an unordered set of name/value pairs.

Syntax: an object begins with { (left brace )and ends with } (right brace) . Each name is followed by : (colon) and the name/value pairs are separated by , (comma).

**Array.** An array is an ordered collection of values.

Syntax: An array begins with [ (left bracket) and ends with ] (right bracket). Values are separated by , (comma).

**Value.** A value can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested.

**String.** A string is a sequence of zero or more Unicode characters, wrapped into double quotes, using backslash escapes. A character is represented as a single character string. A string is very similar to a C or Java string.

**Number.** A number is very similar to a C or Java number, except that the octal and hexadecimal formats are not used.

## 2.2.2 FlexJSON

In order to use JSON in conjunction with a Java application, we need to transform Java objects into JSON and vice versa.

FlexJSON is a library for serializing and deserializing Java objects into and from JSON. What distinguishes FlexJSON from others is its control over what gets serialized allowing both deep and shallow copies of objects. In fact, most JSON serializers mimic object serialization libraries and tries to serialize the whole object graph from the object being turned into JSON. This causes problems when you want a connected object model in your server, but you cannot send that object model to the client because the serialization library will try to send the entire object graph.

Serialize and deserialize is very easy with FlexJSON, now it will be explained what is meant for “serialize” and “deserialize” and what these steps consist.

### 2.2.2.1 Serialization

Serialization is the process which picks our Java Objects and transforms them into a JSON type. FlexJSON takes a different approach allowing you to control the depth to which it will be serialized.

Below, it is presented the main methods and the ones which have been used.

Basic. If you use “basic” serialization, the output will include the first “layer” of your object. By default, FlexJSON serializes the immediate fields of the object, it is just a shallow representation of it. So, all collections are not serialized by default and this means that one to many and many to many relationships are not serialized.

For example it will be shown how the JSON output is if we make a “basic” serialization of a Data View object:

```
{
  "id": "DV_TEST2",
  "name": "Data View Test",
  "recordPerPage": 10,
  "type": null,
  "description": "Data View di Prova"
}
```

Figure 9 - JSON "Basic" of a Data View

Deep. If Deep serialization is used, the output will include all “layer” of your object. Just like *serialize()* method, *deepSerialize()* method will use *includes*, *excludes*, and *annotations* to understand what are the serialized things needed. However, *includes* are generally redundant except in the case where you want to override an excluding annotation on a field. Deep serialization will not serialize cycles in your graph, so if you have a bi-directional relationship which says the parent has a child and the child has a parent, the back references from child to parent will not be included in the JSON output.

As for “basic”, it is going to be shown the output for the same Data View object. As it is possible to see, in using deep serialization, we have more *actionList* field. For brevity, the Output and Input Parameter have been omitted, and they are presented in the JSON obtained by deep serialization.

```

{
  "id": "DV_TEST2",
  "name": "Data View Test",
  "recordPerPage": 10,
  "type": null,
  "description": "Data View di Prova",
  "actionList": [
    {
      "context": "L",
      "displayCondition": null,
      "dynamicContext": true,
      "icon": "accept.png",
      "id": 205,
      "name": "AddPeople",
      "openPopUp": false,
      "parameterSet": [
        {
          "fieldName": "cognomeP",
          "fieldType": "I",
          "name": "cognomeP"
        }
      ],
      "popUpName": null,
      "style": null,
      "type": "ActionUrl",
      "url": "http://localhost:8080/simple/persona.do?task=edit"
    }
  ]
}

```

Figure 10 - JSON "Deep" of a Data View

**Include.** With the `include()` method, we can tell the serializer which fields from the target object are to be included. Collections are serialized in full, including the shallow copy of the object they contain.

Continuing the series of examples using the same Data View object, it will be shown how the `include()` method works. If we use "basic" serialization but we want to have also the `actionList` attribute, we can do that with the follow notation:

```
String json = serializer.include("actionList").serialize(dataView);
```

If you want to include only one specific field, you can do that using the dot notation. For example, if we want only the `icon` of `actionList` attribute, we have to write:

```
String json = serializer.include("actionList.icon").serialize(dataView);
```

**Exclude.** The `exclude()` method allows you to exclude certain fields from serialization. It is very useful if you have fields which you do not want to send to the client. Using dot notation with `excludes` has a subtle difference in it occurs when compared to `includes`, in fact if you exclude a nested field that implies that the rest of the parent object is included.

For example, if we have “deep” serialization, but we do not want `icon` of `actionList` attribute, we have to write:

```
String json = serializer.exclude("actionList.icon").deepSerialize(dataView);
```

**Root Name.** The `rootName()` method allows you to specify an outer object for collections. There are some JavaScript libraries like jqGrid (which is used to draw the data table into the DataView Viewer) which require this for their JSON data models.

**Pretty Print.** The `prettyPrint()` method is essential to the readability and it is necessary to check that the output matching corresponds to what we expect. Without this method the JSON that we produce is viewed all on a line and not in the characteristic JSON structure.

### 2.2.2.2 Deserialization

Deserialization is the process of taking JSON text binding those values into Objects. This process can be quite complex as JSON text contains no typing information. `JSONDeserialzer` class has the task of mapping JSON data types onto static Java objects. It takes as input a JSON string and produces a static typed object graph from the JSON representation. By default, it uses the class property in the JSON data in order to map the untyped generic JSON data into a specific Java type.

If you want to know the other methods that FlexJSON implements, you can visit the online guide<sup>6</sup>.

## 2.2.3 JSON in JavaScript

JSON is a subset of the object literal notation of JavaScript, so it can be used in the language without any difficulty. Members can be retrieved using dot or subscript operators:

To convert a JSON text into an object we can use the `eval()` function, which invokes the JavaScript compiler. The compiler will correctly parse the text and produce an object structure. The use of `eval()` is indicated when the source is trusted and competent because there are security issues when we use it. It is much safer to use a JSON parser. A JSON parser will recognize only JSON text, rejecting all scripts.

A JSON stringifier goes in the opposite direction, converting JavaScript data structures into JSON text, but JSON does not support cyclic data structures. If the `stringify` method sees an object that contains a `toJSON` method, it calls that method, and stringifies the value returned. This allows an object to determine its own JSON representation.

---

<sup>6</sup> <http://flexjson.sourceforge.net/>

## 2.2.4 The Advantages of JSON over XML

Both JSON and XML can be used to represent native, in-memory objects in a text-based, human-readable, data exchange format. Therefore, when deciding upon a data exchange format, it is not a simple matter of choosing one over the other, but rather what format has the characteristics that make it the best choice for a particular application.

Therefore, why we choose to use JSON instead of XML?

- It provides better language-independent representation of data structures.
- It has a simpler spec.
- It is shorter to be configured. Syntax is very terse and yields formatted text where most of the space is consumed by the represented data.
- It adapts better to technology chosen for the display part, jQuery, and then JavaScript.
- Debugging is also easy.
- Native array support.
- Native object support.
- Native recognized the null value.
- No additional application code required to parse text; can use the JavaScript's `eval()` function.

On the downside, there is the fact that JSON is probably less used and versatile of the most popular XML and then, as it can be easy to use, is another programming language to learn.



## 2.3 jQuery

“jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development”, this is the definition of jQuery by its creator John Resig<sup>7</sup>.

The jQuery library is designed to keep the things very simple and reusable and it is providing many easy to use functions and methods to make rich application. Since it is based on JavaScript it can be used with JSP, Servlets, ASP, PHP, CGI and almost all the web programming languages.

The following will list the major features of jQuery:

DOM element selections function. One of the most powerful aspects of jQuery is its ability to make selecting elements in the DOM (Document Object Model) easy. We can use any type of selector in jQuery and we must always have to start with: `$()`, where we can placing inside the parentheses, wrapped in quotation marks, just about anything that can be used in a stylesheet. Three building block of these selectors are tag name, ID and class. They can be used either on their own or in combination with other selectors.

DOM traversal and modification. With jQuery’s DOM traversal methods we can select a parent or ancestor elements of our DOM. With these methods we can go up, down and all around the DOM tree with ease. They can be useful for styling specific cells, for example, using the `next()` method in conjunction with the `addClass()` method, we can add, to the DOM following that we have specified, a CSS class. If we have something like this in our HTML code:

```
< span id="example"> 1 </ span >
< span > 2 </ span >
< span > 3 </ span >
```

And we use this JavaScript code:

```
$( 'span.example' ).next () .addClass ( 'redFont' );
```

Whit the `redFont` class:

```
redFont{font-color:red;}
```

Our result on the HTML page is as follows:

1 2 3

This traversal-method combination, and many other, illustrate jQuery’s chaining capability. It is possible with jQuery to select multiple sets of elements and do multiple things with them, all within a single line of code.

Events. jQuery has many methods and constructs for event handling. These include the ability to perform tasks on page load or simple events, like `onload()`, `onclick()`, `onresize()`, and also compound events, that intercept combinations of user actions and respond to them using more than one function, like `.ready()` and `.hover()` methods.

---

<sup>7</sup> John Resig (born 1984) is a JavaScript Tool Developer for the Mozilla Corporation.

CSS manipulation. There may be times when we need to apply styles that have not been defined in a stylesheet. jQuery offers the `.css()` method for such occasions. This method acts as both a *getter* and a *setter*. Using the HTML code defined in the previous example, we can use this JavaScript construct to show the use of the `css()` method as a setter:

```
$('#span.example').css('font-color','red');
```

And the output is:

1 2 3

Effects and animations. Through the use of jQuery, we can easily add impact to our actions through a set of simple visual effects, and even craft our own, more sophisticated animations. With those effects we can make elements which gradually slide into view instead of appearing all at once, we can also choose the speed of their appearance, and many other things.

Ajax. Ajax is just a means of loading data from the server to the web browser, or client, without a visible page refresh. This data can take many forms, like JSON files, and we have many options for what to do with it when it arrives.

JavaScript Plugins. Its plugin architecture has allowed developers to extend jQuery, making it an even more feature-rich library. Below, we will explore the official jQuery UI plugin library.

## 2.3.1 jQuery UI

jQuery UI (user interface) is a library built on top of jQuery. It is responsible for interactions, widgets, components, themes, advanced effects and animations. Interaction components include methods for dragging, dropping, sorting and resizing items. The current stable version of widgets included an accordion, date picker, dialog, slider and tabs. Additionally, jQuery UI provides an extensive set of advanced effects to supplement the core jQuery animations.

A recent addition to the jQuery UI library is the ThemeRoller, a web-based interactive theme engine for UI widgets. The ThemeRoller makes creating highly customized, professional-looking elements quick and easy.

## 2.3.1 JSON Ajax Calling

We want to focus on the Ajax calling of a JSON, because it makes save a lot of time and improves the usability of the entire application. First of all, what is an Ajax call?

Ajax stands for “Asynchronous JavaScript and XML” and is a technique for creating fast and dynamic web pages. It allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes, so this means that it is possible to update parts of a web page, without reloading the whole page.

Through the help of jQuery make an Ajax call it is simple and we can also specify a set of attribute to satisfy our needs. The Ajax technique really comes into its own only when the server can dynamically shape the data based on input from the browser. We are helped along jQuery in this task as well. Therefore, we can make a *GET* or *POST* request, and also associate events in light of the response, whether successful or not.

## 2.3.2 jqGrid

jqGrid is a JavaScript plugin, based on jQuery, designed to work with table's format data. Since the grid is a client-side solution loading data dynamically, it can be integrated with any server-side technology, including of course Java Servlets and JSP.

jqGrid has many features and parameter to set, making it adaptable to the needs of the developer and, the reasons why we choose jqGrid are quite simple:

- A) Supports JSON data type. With jqGrid it is possible to use data in JSON format using the simple notation, where we configuring the grid, to read data:

```
datatype: "json",
url: dataUrl
```

and formatting the JSON to be compatible, entering root, page, total and records, as follows(where *DV\_PERSONA\_OVERALL* is the root name):

```
{
  "totalpages": "nOfPages";
  "currpage": "positionOfCurrentPage";
  "totaltrecords": "nOfTotalRecords";
  "DV_PERSONA_OVERALL": [ ...
```

- B) It can works with Ajax call. We are able to call, for example, a new set of data, after a search performing perhaps, in this simple way:

```
jQuery("#gridID").jqGrid().setGridParam({
  url: dataUrl
}).trigger("reloadGrid");
```

- C) Its graphics are editable through the themes of jQuery Ui. It is enough to put our preferred theme in the *Theme* folder of our project, and it styles the entire grid.
- D) It is easy to configure and it has a great support by the community. As we can see use a JSON or style our grid is really trivial and, as for these things, also use other parameter is very simple. In fact, if we want to define the names of our columns, we can simple do that by passing the list of names that we want to see displayed as the headers(note that it is possible to pass an array containing those names). Or, we can set the characteristic of each column by passing a list of attributes(with each attribute in the syntax *attribute:'value'*), which could be the text alignment inside the cell(*align*), the width of the column (*width*) or if the field is sortable or not(*sortable*). There are obviously many other parameters and fields that we are not going to list for brevity.

## 2.4 The DataView

The DataView is a web application developed by Ing. Roberto Pellegrino for Quix srl, which has the unique feature to accept in input a data source from any DBMS, for which there is the JDBC driver to access it, without knowing any information about data or DBMS(during production the following DBMS have been tested: Mysql, SqServer, Oracle, PostgreSQL, DB2/AS400). Its task does not end here because we can interact on these data through queries on the database and by providing Input Parameters, Output fields and Actions.

Primarily the DataView is divided into two sections:

- Data Connection;
- Data View.

### 2.4.1 Data Connection section

Data Connection(Data Connection or Data Source) contains all the information to connect to DBMS for extract data using our Data Views.

Gestione DataConnection			
Id	DC_PERSONA	Tipo	DataConnection
Driver	MySql	Url	jdbc:mysql://localhost/mytest
UserName	root	Password	••••
		<input type="button" value="SALVA"/> <input type="button" value="CHIUDI"/>	

Figure 11 - Data Connection

The Data Connections are described by the following attributes:

- ID unique ID of the connection;
- URL of the database;
- USERNAME of the user to access database;
- PASSWORD of the user to access database;
- TYPE of connection to Data Source
  - Data Source: where we have to specify only the URL attribute;
  - Data Connection: where we have to specify all the attributes;
- DRIVER of the database.

LISTA DATACONNECTION			
Trovati 3 elementi.			
ID	URL	TIPO	
DC_PERSONA	jdbc:mysql://localhost/mytest	DataConnection	1 2 3
DV_C	java:comp/env/jdbc/dataview	DataSource	
DV_CATALOG	jdbc:mysql://srvmsnet/extensibility	DataConnection	

1

Figure 12 - Data Connection List

As the screenshot above shows, we can have defined more than one connection at the same time and we can modify(2) or delete(3) them or, obviously, define a new one(1).

An interesting aspect is presented to us when we try to save a new data connection or a modified one. In fact with this action, the connection is tested and the saving operation is successful only if the test is positive.

## 2.4.2 Data View section

This is the main section of the Web Application, and it begins showing us the list of the “Data View Projects”, that are nothing else that folders which contains the Data Views with an interior affinity.

code	name	3	4	5	6
DATAVIEW_PROJECT	Progetto DataView	1	2	3	4
PRJ_PERSONA	Viste persona	1	2	3	4
PRJ_TEST	Progetto di test	1	2	3	4

1

CHIUDI

Figure 13 - DataView Project List

In this section of application we find two global actions:

- Create new DataView Project(1): any DataView Project is composed of two fields, code and name. Code is the unique id the project and name is a short description;
- Import DataView Project(2): this action permits to importing a DataView Project previously exported. When we importing a DataView Project we can choose which of Data Views that the project contains we want to import and, if we need, we can choose a different Data Connection. Both importing and exporting are based on a XML file with a fixed structure.

For each DataView Project we find a set of default actions which are:

- Access to Data View(3): it permits to view the list of Data Views that the project contains;
- Modify DataView Project(4): though the use of this action we can only change the *Name* attribute of the project;
- Delete DataView Project(5): this action provides to delete the Project, but it is allowed only if the Project does not contain any Data View;
- Export DataView Project(6): with this action we can save the Project folder and all Data Views inside of it.

From this section, through appropriate action, we have direct access to the list of Data Views of the selected Project. In this page we found all the Data Views of a Project and the actions to manage them.



Figure 14 - Data Views List

First of all there are two global actions:

- Create a new Data View(1).
- Import a Data View(2): as for the importing of a DataView Project, this action permits to importing a Data View.

For each Data View we find a set of default actions which are:

- Modify Data View(3): it opens the configuration page of the Data View and permits to modify everything on it.
- Copy Data View(4): it makes a copy of the selected Data View.
- Play Data View(5): it shows the created Data View.
- Report Data View(6): it reports function of an existing Data View. Reports have been designed to enable the DataView to create PDF files. After the DataView has finished to running the query sends the results to the Report which encapsulates them in a PDF file.
- Chart Data View(7): it opens the Chart section where we can manage the charts associated at the Data View.
- Export Data View(8): with this action we can add the Data View to the exporting list.
- Delete Data View(9): it deletes the Data View.

We are going to show how to create a Data View and which sections composing it. We can divide Data View into four main sections, plus a part of initial configuration.

**Gestione DataView**

Id  Nome

Progetto  DataView di sistema

Sorgente dati

Descrizione

Record per pagina

**Parametri di input**

Non sono presenti parametri di input

**Query**

**Campi di output**

Non sono presenti campi di output

Ordinare di default per:  Tipo di ordinamento:

**Azioni**

Non sono presenti azioni

Figure 15 - Data View New

The first set of attributes(1) to be included, to create a new Data View are:

- Unique ID of the Data View.
- Name. It is a short description about the Data View.
- Project which it belongs.
- System Data View. If a Data View has this attribute it cannot be deleted.
- Connection to the data source.
- Description of the Data View content.
- Number of Records per Page. We can choose how many records we want to view in the grid. If this attribute is "-1" we will see all the records.

After this short configuration we have to define the main features of our Data View.

### 2.4.2.1 Query

Query(2) is an SQL interrogation for extracting data.

The query must be written according to the table structure and SQL syntax provided by the DMBS in use. Many options are available for the user, in fact we can use parametric query(associated with Input Parameters), it is possible to use implicit parameters, like `username(_USERNAME)` or `user language(_USERLANGUAGE)` and parameters can also be subject to conditions, creating conditional query. For the creation of this last type, we must write query with a scripting language that permit to define conditions inside them. The language in use is Freemarker.

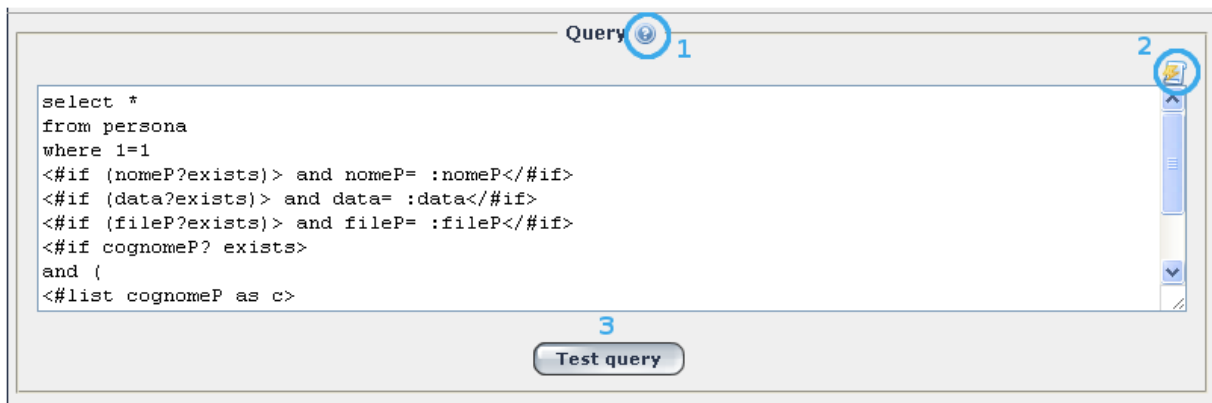


Figure 16 - DataView Query

The DataView provides to the user three very useful tools:

- Inline Help(1).
- Activation/Deactivation Syntax support(2).
- Test Query, for immediately check(3).

Queries associated with the Input Parameters can be of two types:

#### Parametric Query.

Parameters must be insert in the query whit the `:parameterName` annotation. In queries we can use of course all the SQL operators.

For example:

```
select * from tableName where columnName = :parameterName
```

#### Conditional Query.

Through optional input parameters it is possible to set the existence condition of filter's enhancement. To do this we must use Freemarker.

For example, assuming that ID parameter is optional:

```
select * from people <#if (ID?exists)> where idPeople = :ID </#if>
```



### 2.4.2.2 Output Parameters

Output Parameters(3) are the set of parameters that we want to view, nothing else that the columns of SELECT operation, coming from SQL interrogation defined in the Query field.

Name	Label	Type	Format	Style	Vis.	Σ	Ord.	3
Nome	Etichetta	Tipo	Formato	Stile	Vis.	Σ	Ord.	
cognomeP	Cognome	String			<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	↓ ↑
id	ID	Integer			<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	↓ ↑
nomeP	Nome	String		<#if id == 25>color:r	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	↓ ↑
fileP	Curriculum	String			<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	↓ ↑
data	Data Assunzione	Date	dd/MM/yyyy	dd/MM/yyyy	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	↓ ↑
prezzo	Busta Paga	Decimal	###.##		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	↑

Default Sorting: Ordinare di default per: cognomeP  
 Sorting Type: Tipo di ordinamento: Crescente

Figure 17 - Output Parameters List

The functions and parameters in this section are:

- Insert New(1): we can add one column of SELECT operation that we have not yet added.
  - Delete(2): it obviously delete the selected Output Parameter.
  - Sequence(3): we can decide the order of the Output Parameters with the up and down arrows.
  - Sortable(Ord.): an Output Parameter can be sortable or not in the result grid..
  - Visible(Vis.): an Output Parameter can be visible or not in the result grid.
  - Σ: this function sum the value of that Output Parameter.
  - Label: it allows to rename the column header in output.
  - Type: we have to choose from the list the type of the Output Parameter, it can be a string, integer, date or other.
  - Format: in this field we can put the format of Output Parameter, for example, with a date, we can use dd/mm/yyyy.
  - Style: It is possible to define the HTML style for the Output Parameters, also through conditions using Freemarker.
- For example (ID Output Parameter):

```
<#if (ID ==66)> color : red <#if>
```

With this condition we change the font color to red of the Output Parameter value where we set this style if the ID Output Parameter on the same row is equal to 66.

- Default Sorting and Sorting Type: at the end of the form we find Default Sorting field and Sorting Type (that can be increasing or decreasing).

### 2.4.2.3 Input Parameters

Input Parameters(4) are a set of parameters which are the filters on the query contained in the Query field.

Nome	Etichetta	Tipo	Default	Obb.	4	2	3
cognomeP	Cognome	String			↓	✎	✖
nomeP	Nome	String			↓ ↑	✎	✖
data	Data Assunzione	Date			↓ ↑	✎	✖
fileP	Curriculum	String			↑	✎	✖

Figure 18 - Input Parameters List

In this section we can find the default actions like New(1), Edit(2), Delete(3) and, as in Output Parameters, the Sequence(4) action.

Figure 19 - Input Parameter

An Input Parameter is composed by:

- **Label:** it is the name in the filter's column when we are in view mode.
- **Type:** we have to choose from the list the type of the Input Parameter, it can be a string, integer, date or other.
- **Text Case:** if the Input Parameter type is string, we can choose if the Input Parameter is case sensitive or lower/upper case.
- **Required:** if we setting this option, the Input Parameter is required.
- **Advanced Search:** if a Input Parameter is "Advanced", is not shown directly, but is viewable only by user request.
- **Default Value:** it is the default value of Input Parameter set by user.
- **View.** We have to choose through many types of Input Parameter.
  - o Default: by default the Input Parameter is a free input box where we can write strings, numbers or dates.

- DropDownList: with the DropDownList the Input Parameter is a menu list composed by another Data View.
- PopUp: it opens another window. It is used for complex Input Parameter.
- CheckBox: each item in the Input Parameter is represented by a check box.
- Custom: it calls a custom page.

#### 2.4.2.4 Actions

Actions(5) are operation that we can associate at each record of the result set or at the entire result set (global actions).

Nome	Tipo	Ord. 4	2 X
Nuovo Dipendente	ActionUrl	↓ ↑	✎ ✕ 3
Modifica Dipendente	ActionUrl	↓ ↑	✎ ✕
Elimina Dipendente	ActionUrl	↓ ↑	✎ ✕
Dowload Curriculum	ActionJavaScript	↑	✎ ✕

Figure 20 - Actions List

Even for the Actions we found the four basic functions, which are New(1), Edit(2), Delete(3) and Sequence(4).

An Action is composed by:

- **Name**: the name of the Action.
- **Type**: it can be chosen from a variety of types.
  - Action URL: to call the URL of the page that contains the action.
  - Action JavaScript: to write JavaScript code.
  - Action Communication: to create Actions those correlate Data View into Liferay's Portlet.
- **Global Action**: if an Action is not set to Global is repeated for each result of the result set.
- **PopUp**: if enable the Action open a new window.
- **Dynamic Contest**: it tells if the contest is dynamic.
- **Icon**: it permits to choose the icon associated with the Action. If no icon is being selected the Action is represented by the Name parameter.
- **Parameter**: the set of Input and Output Parameters involved in the Action must be selected. For example if it is called another Data View, you must specify the parameters that will be related to its Input Parameter.
- **Condition**: JavaScript conditions that may be written on the parameters.

**Gestione azioni**

Name  
Nome

Global Action  
Azione globale

Dynamic Contest  
Contesto dinamico

Url  
Url

Type  
Tipo ActionUrl

PopUp  
PopUp

Icon  
Icona

Parameter  
Parametri

Condition  
Condizione

**Filtri di ricerca**

	Nome	Nome alternativo
<input type="checkbox"/>	cognomeP	<input type="text"/>
<input type="checkbox"/>	nomeP	<input type="text"/>
<input type="checkbox"/>	data	<input type="text"/>
<input type="checkbox"/>	fileP	<input type="text"/>

**Campi di output**

	Nome	Nome alternativo
<input type="checkbox"/>	cognomeP	<input type="text"/>
<input type="checkbox"/>		<input type="text"/>

**AGGIUNGI** **CHIUDI**

Figure 21 - Action

## 2.4.3 Charts

As we previously mentioned, you can create charts associated with existing Data View.

**Lista Grafici associati a DV\_PERSONA\_OVERALL - Visione Dipendenti**

**LISTA GRAFICI**

Trovati 1 elementi.

id	Nome	Descrizione	1	2	3	4
1	Visione Dipendenti Chart	Grafico di Test				

**CHIUDI**

Figure 22 - Charts List

A Chart can be Created(1), Modified(2), Displayed(3) or Deleted(4).

Gestione grafici	
DataView	DV_PERSONA_OVERALL - Visione Dipendenti
Name Nome	Visione Dipendenti Chart
Type Tipo	Barre 3D Verticali
Description	Grafico di Test
Width Larghezza	400 px
Height Altezza	400 px
X Label Etichetta X	
Y Label Etichetta Y	
Category Categoria	Busta Paga
Series Serie	<input type="checkbox"/> ID    Color    Colore <input type="checkbox"/> Labels    Etichette <input checked="" type="checkbox"/> Busta Paga    Color    Colore <input type="checkbox"/> Etichette
<input type="button" value="SALVA"/> <input type="button" value="CHIUDI"/>	

Figure 23 - Chart New/Edit

A Chart is composed by:

- Name: short description of the chart.
- Type: choice of graphics solution(histogram, pie, etc.).
- Description: the description of the chart's content.
- Width: the width of the chart in pixel.
- Height: the height of the chart in pixel.
- X Label: the x-axis label name.
- Y Label: the y-axis label name.
- Category: the category of the chart(proposal among the variables of the Data View).
- Series: the chart series, after the choice of the Category.

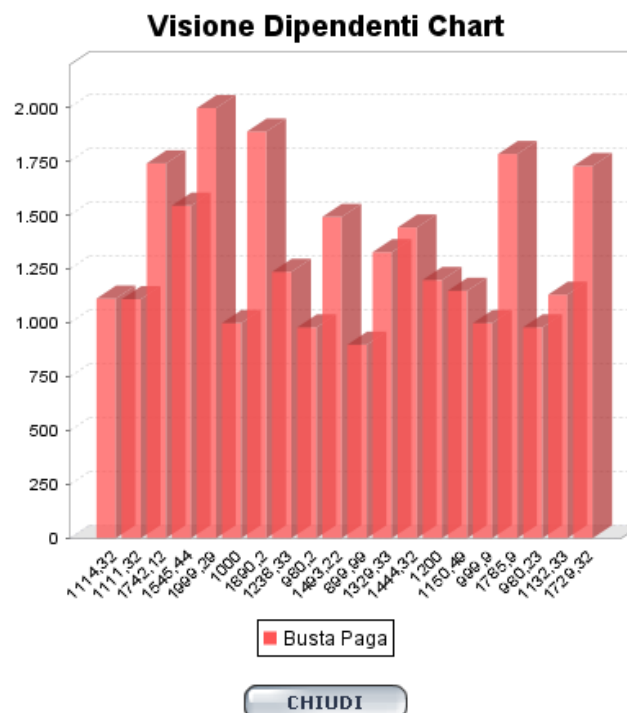


Figure 24 - Chart Example

## 3 Core – Rest Logic and the Viewer

This chapter talks about the REST Logic, going to touch and explain the whole fundamental aspects of this architecture. The application developed, the Viewer, will then be shown. The project, the architecture and a small user manual will be explained.

### 3.1 REST – Representational State Transfer

In the web services world, Representational State Transfer (REST) is a key design idiom that embraces a stateless client-server architecture in which the web services are viewed as resources and can be identified by their URLs. Web service clients that want to use these resources access a particular representation by transferring application content using a small globally defined set of remote methods that describe the action to be performed on the resource.

A basic web page is an example of a REST resource. When we access the page with our browser, a *GET* request is sent behind the scenes. *GET* is the most common action used in REST. The other popular action in REST is *POST*, which adds data to the service.

Two other common actions associated with REST are *PUT* and *DELETE*, however the implementation of these last two actions is spotty.

The largest known implementation of a system conforming to the REST architectural style is the World Wide Web.

The main points of this architecture are to be shown, largely by taking inspiration and using as reference the dissertation “Architectural Styles and the Design of Network-based Software Architectures” written by Roy Thomas Fielding<sup>8</sup>, the founder of the REST logic and architecture.

#### 3.1.1 Key Features

When an application in REST logic is designed, we start without constraints, and then incrementally identify and apply constraints to elements of the system in order to differentiate the design space. Therefore, we can begin to add the constraints:

Client-Server architectural style. Separating the user interface from the data storage, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components. That separation also allows the components to evolve independently.

Stateless. Communication must be stateless, such that each request from client to server must contain the whole information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.

This constraint induces the properties of visibility, reliability and scalability.

- Visibility is improved because a monitoring system does not have to look beyond a single request datum in order to determine the full nature of the request;
- Reliability is improved because it eases the task of recovering from partial failures;

---

<sup>8</sup> Roy Thomas Fielding (born 1965) is an American computer scientist. He is one of the principal authors of the HTTP specification (RFC 2616).

- Scalability is improved because not having to store state between request allows the server component to quickly free resources, and further implementation because the server does not have to manage resource usage across request.

The main disadvantage of stateless is that may decrease network performance by increasing the repetitive data sent in a series of request, since that data cannot be left on the server in a shared context.

Cache. Cache constraints require that the data within a response to a request be labelled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests. The advantage of caching is that it has the potential to partially or completely eliminate some interaction, improving efficiency, scalability and user-perceived performance. The disadvantage is that a cache can decrease reliability if stale data within the cache differs significantly from the data that would have been obtained had the request been sent directly to the server.

Uniform Interface. REST bases his style on a uniform interface between components. The overall system architecture is simplified and the visibility of interactions is improved. As disadvantage, we find that a uniform interface degrades efficiency, since information is transferred in a standardized form rather than on which is specific to an application's need. The REST interface is designed to be efficient for large quantity of hypermedia data transfer, optimizing for the common case of the Web.

Layered System. The layered systems style allows an architecture to be composed of hierarchical layers by constraining component behaviour such that each component cannot "see" beyond the immediate layer with which they are interacting. Layers can be used to encapsulate legacy services and to protect new services from legacy clients, simplifying components by moving infrequently used functionality to a shared intermediary. First disadvantage of layered systems is that they add overhead and latency to the processing of data, reducing user-perceived performance. For a network-based system that supports cache constraints, this can be offset by the benefits of shared caching at intermediaries.

Code-On-Demand. REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented. This improves system extensibility but it also reduces visibility.

### 3.1.2 Data Elements

When a link is selected, information needs to be moved from the location where it is stored to the location where it will be used by. REST provides a hybrid, taken characteristics from options of a distributed hypermedia architect, by focusing on a shared understanding of data types with metadata, but limiting the scope of what is revealed to a standardized interface. REST components communicate by transferring a representation of a resource in a format matching one of an evolving set of standard data types, selected dynamically based on the capabilities or desires of the recipient and the nature of the resources.

Resource. Any information that can be named can be a resource: a document or image, a temporal service or others. A resource is a conceptual mapping to a set of entities, not the entity

that corresponds to the mapping at any particular point in time. REST uses a Resource Identifier to identify the particular resource involved in an interaction between components. REST connectors provide a generic interface for accessing and manipulating the value set of a resource.

Representations. REST components perform actions on a resource by using a representation to capture the current or intended state of that resource and transferring that representation between components. A representation consists of data and metadata describing the data. Metadata is in the form of name-value pairs, where the name corresponds to a standard that defines the value's structure and semantics. Response messages may include both representation metadata and resource metadata.

### 3.1.3 Connectors

REST uses various connector types(client, server, cache, resolver, tunnel) to encapsulate the activities of accessing resources and transferring resource representations. As all REST interactions are stateless, so each request contains all of the information necessary for a connector to understand the request, independent of any requests that may have preceded it.

This restrictions accomplishes four functions:

- It removes any need for the connectors to retain application state between request, thus reducing consumption of physical resources and improving scalability;
- It allows interactions to be processed in parallel without requiring that the processing mechanism understand the interaction semantics;
- It allows an intermediary to view and understand a request in isolation;
- It forces all of the information that might factor into the reusability of a cached response to be present in each request.

Client and Server. First connector types are client and server, the difference between the two is that a client initiates communication by making a request, whereas a server listens for connections and responds to requests in order to supply access to its services.

Cache. The cache connector, can be located on the interface to a client or server connector in order to save cacheable responses to current interactions so that they can be reused for later requested interactions. Some cache connectors are shared, meaning that its cached responses may be used in answer to a client other than the one for which the response was originally obtained.

Resolver. A resolver translates partial or complete resource identifier into the network address information needed to establish an inter-component connection. For example, most URI include a DNS hostname as the mechanism for identifying the naming authority for the resource. In order to initiate a request, a Web browser will extract the hostname from the URI and make use of a DNS resolver to obtain the Internet Protocol address for that authority.

Tunnel. At least, there is the tunnel, which simply relays communication across a connection boundary, such as a firewall or lower-level network gateway.



### 3.1.4 Components

REST components are typed by their roles in an overall application action.

User Agent. A user agent uses a client connector to initiate a request and becomes the ultimate recipient of the response. The most common example is a Web browser, which provides access to information services and renders service responses according to the application needs.

Origin Server. An origin server uses a server connector to govern the namespace for a requested resource. It is the definitive source for representations of its resources and must be the ultimate recipient of any request that intends to modify the value of its resources.

Intermediary components act as both a client and a server in order to forward request and responses.

Proxy. A proxy component is an intermediary selected by a client to provide interface encapsulation of other services, data translation, performance enhancement, or security protection.

Gateway. A gateway component is an intermediary imposed by the network or origin server to provide an interface encapsulation of other service, for data translation, performance enhancement, or security enforcement.

The difference between a proxy and a gateway is that a client determines when it will use a proxy.

### 3.1.5 REST Process View

A process view of an architecture is primarily effective at eliciting the interaction relationships among components by revealing the path of data as it flows through the system. REST's client-server separation of concerns simplifies component implementation, reduces the complexity of connector semantics, improves the effectiveness of performance tuning, and increases the scalability of pure server component. Layered system constraints allow intermediaries proxies, gateways and firewalls to be introduced at various point in the communication without changing the interfaces between components, thus allowing them to assist in communication translation or improve performance via large-scale, shared caching. REST enables intermediate processing by constraining messages to be self-descriptive: interaction is stateless between requests, standard methods and media types are used to indicate semantics and exchange information, and responses explicitly indicate cacheability.

### 3.1.6 RESTful Web Services

Starting from the definition of REST architecture, we realize that in a web application based on this logic, a client makes a **request** for a **resource** on a server, and the server issues a **response** that includes a representation of the resource.

A web application use the HTTP(Hypertext Transfer Protocol), which is a stateless protocol based on a client requesting a resource across a network and the server providing a response. As such, an HTTP transaction, entails a request and a response.

An HTTP request has three parts:

- The request line, which includes the HTTP method, the URI and the HTTP version.
- One or more optional HTTP headers, which are key-value pairs that characterize the data being requested and/or provided.
- An optional message body, which is data being sent from the client to the server as a part of the request.

An HTTP response also has three parts:

- The HTTP status code, indicating the status of the requested URI.
- One or more optional HTTP headers, which are key-value pairs that characterize the data being provided.
- An optional message body, which is the data being returned to the client in response to the request.

Each URI points to a resource on the server. Think of URIs as nouns, with one URI for each resource.

As previously said, HTTP defines several methods to execute on a resource: HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT and PATCH. However we are talking only about the GET and the POST.

To retrieve a resource, issue an HTTP GET request. GET request have no side effects, meaning they do not produce any change in the resource. GET requests do not include a message body, but GET responses usually do.

To submit data to be processed, issue an HTTP POST request. POST request require a message body, i.e. the data to be processed.

Talking about HTTP response data formats, we can say that a conventional web server delivers its data in HTML format, with related JavaScript, CSS and image files. HTML is an excellent format for making up textual data for human use, but it has very limited expressive power for structuring data beyond simple documents.

The most common formats used to transmit structured data across HTTP are XML and JSON. It is clear that we have chosen the JSON format.

### 3.1.6.1 When to Use RESTful?

After explain the key features of the REST architecture and RESTful web services, we can certainly say that RESTful design may be appropriate when:

- The web services are completely stateless.
- A caching infrastructure can be used for performance. If the data that the web service returns is not dynamically generated and can be cached, then the caching infrastructure that the web servers and other intermediaries inherently provide can be leveraged to improve performance. However we must take care because such caches are limited to HTTP *GET* method for most servers.

- The service producer and service consumer have a mutual understanding of the context and content passed along. Because there is no formal way to describe the web services interface, both parties must agree out of band on the schemas that describe the data being exchanged and on ways to process it significantly.
- Bandwidth is particularly important and needs to be limited. REST is particularly useful for limited-profile devices such as PDAs and mobile phones, for which the overhead of headers and additional layers of SOAP elements on the XML payload must be restricted.
- Web service delivery or aggregation into existing web sites can be enabled easily with a RESTful style. Developers can use technologies such as AJAX to consume the services in their web applications.

## 3.2 The Viewer

The Viewer is a Web Application, which is one of the DataView's extension. Specifically, it is the part showing us the Data View, then the Output Parameters, the result of the Query, the Input Parameters and the Actions.

Obviously, the application DataView already has a viewer inside. So, what are the differences between the DataView default viewer and the Viewer? And, what are the benefits?

### 3.2.1 Project

The key word of the Viewer is: decoupled.

In this context, the word decoupled means that the Viewer is a stand-alone web application compared to the DataView. The most important advantage resulting from this operation is the independence of the Viewer from the single version of the DataView which is currently being used. For example, if the communication interface remains unchanged, possible migration of the DataView will not involve any change in the Viewer.

The communication between the two applications is realized through the exchange of information and data in JSON format. Therefore, the Viewer does not care how the information it requests has been generated, but only cares to read and interpret them.

SECURITY AND SSO. Security issues do not affect the Viewer. This aspect is guaranteed through the use of the CAS(Central Authentication Server), both in the DataView and the Viewer. The CAS is a SSO(single sign-on) protocol for the web and, its purpose, is to allow a user to access multiple applications while providing their credentials only once. Par consequence, a user can log into the DataView and the Viewer by entering a username and password only once.

DISADVANTAGES. The exchange of information between the Viewer and the DataView is the only weak point because, obviously, the JSON file transfer is not necessary in the DataView integrated viewer. The disadvantage is in terms of performance, even if we tried to minimize the gap by storing the Data View definition into the cache.

The Viewer has, as logical, two main parts: the server and the client side.

#### 3.2.1.1 The Server Side

The Server side can be further divided into two parts.

The Viewer Server Side. This consists of three tasks:

- ***showDV*** : this is the first task which is directly called by the user. Its role is very simple, namely seeing if the required Data View is cached or not. If the answer is affirmative, it calls the *jsonDvDeserializer* task, otherwise it performs the forward to the *showDv.jsp* page passing it, through the request, the *id* and the *language* parameters;

- *jsonDvDeserializer*: this task sends the map containing the definition of the Data View to the main Jsp page (*DataView.jsp*). The map can be retrieved in two ways:
  - o If the definition of the Data View is cached, it is retrieved from the cache.
  - o Otherwise, the task tries to retrieve from the request, the URL containing the location of the JSON which has the definition of our Data View. Aftermath, it takes the JSON file, which is de-serialized in a map and after deleted.
- *dataViewCacheClear*: as the name suggests, this task does nothing more than empty the cache.

The DataView Server Side. The server side of the DataView is composed of three tasks as well, needed by the Viewer:

- *getDvDefinition*: this task is called by the *ShowDv.jsp* page and has the role of recovering the definition of the Data View and serialize it into a file in JSON format. The newly created file is then saved to a temporary folder, and its URL is serialized into another JSON file, which is sent to the *jsonDvSerializer* task of the Viewer.
- *getDvResultSet*: the current task is called whenever the query result set is needed and, it is likely the most complex task of the whole application. The output is a JSON file containing the query result plus other parameters needed for the Viewer. The complexity is the need to comply with a particular JSON formatting to make it readable to the grid of our application. The output is a JSON containing the parsed query, the current page and the number of total pages and total records, plus the rows which are the output of the query. For each row, it contains a list of map of Output Parameters in the key-value syntax, and the code needed to draw the associated actions. If there are calculated fields, it postpones a map containing the necessary data to populate the *Total* row in the grid.
- *getDvInputParameter*: if the main Jsp page of the Viewer checks that there are Input Parameters, it calls this task which returns a JSON file and containing the code to draw properly Input Parameters of CheckBox, DropDownList and PopUp types.

### 3.2.1.2 The Client Side

The Client side is responsible for sending the parameters, which are requested by the user, to the server, and to draw the Data View received in response.

Essentially, the client part consists in two pages that the user sees through the browser and with which he/she interacts.

ShowDv.jsp. The first page is just to forward to the DataView *getDvDefinition* task the *id* and the *language* of the wanted Data View. If successful, the server of the DataView returns a JSON file containing a URL. This URL displays the location of the JSON containing the definition of our Data View. The URL is used by the page, as parameter, for the call of *jsonDvDeserializer* task.

This page is skipped, by the Viewer server, if the requested Data View definition can be found in the cache memory.

DataView.jsp. The second page is the real front of the application. This last one receives as input, from the request, the map, that the Viewer server has generated, which contains the Data View definition. And it also receives the language that the user has chosen. It is responsible for designing the whole page based on received information.

Because of the Data View definition, through JavaScript, it performs the whole work:

- It generates the URLs needed to retrieve the Input Parameters and the Results Set, and to call the tasks which empty the cache and to generate the Excel file to download;
- It defines the arrays which contain the Output Parameters headers and definitions necessary to jqGrid to draw the grid, along with other parameters it needs, such as the grid sorting and the Result Set URL that the grid has to call;
- It checks for Input Parameters, and if there is the presence of them, calls the function which returns to their definition and draws them according to their properties. Input Parameters using the definition of the Data View even when the search form is clean, checks to see if they have default values and brings them back in form: this operation is realized through personalized functions;
- It draws the global Actions, if there is the presence of them.

It contains calls to the style sheets defining the theme of the application, and to the external JavaScript scripts. These scripts contain the majority of functions needed to run the client side. We have some jQuery extensions realized by various developers, like the same jqGrid, which is used to show the results of the query, or like the jQuery Ui DatePicker for picking dates. We moreover have some scripts developed by myself, like the form validator, which is responsible to check that the parameters, that we have inserted in the form fields, are of the same type like those found in the Data View. This script also shows us the error messages, in case of type mismatch, and has the task of composing the URL with parameters to pass to the query when doing a search.

This specific step of the analysis takes us to an interesting aspect of the Viewer. It possesses the ability, because of jqGrid and Ajax, to show us the data, received in response to request performed, without reloading the whole page, but only refreshing the grid. Therefore, if the vision of the records on the second page of the grid is needed, or if a search is performed, there will be just the reloading of the grid. This feature shows its advantages both in user experience and in loading times.

As a further aspect, it should be emphasized that some components of inputs, such as checkboxes and dropdown lists, have been re-implemented than the defaults ones in order to make their use more practical and understandable. For this purpose the instant search, for instance, has been used.

Talking about the look and feel, the whole application has been styled using the jQuery Ui themes, which are really easy to design and to use. Without considering that the most important component, the grid, is completely stylized through them without having to act on it.

As a final appearance, but not least, the application fits entirely on the language selected by the user, through the use of files containing the necessary translations for the various components, and because of DataView which returns the definition of a Data View in the required language.

### 3.2.2 Functional Architecture

In this section the Functional Architecture of the Viewer will be taken into account.

The diagram which will be showed is an Activity Diagram representing the scenario when the client requires the definition of a Data View. In particular, it is structured in the “SwimLane” format, to make it clear which are the actors at various stages (Client Viewer, Server Viewer, Server DataView).

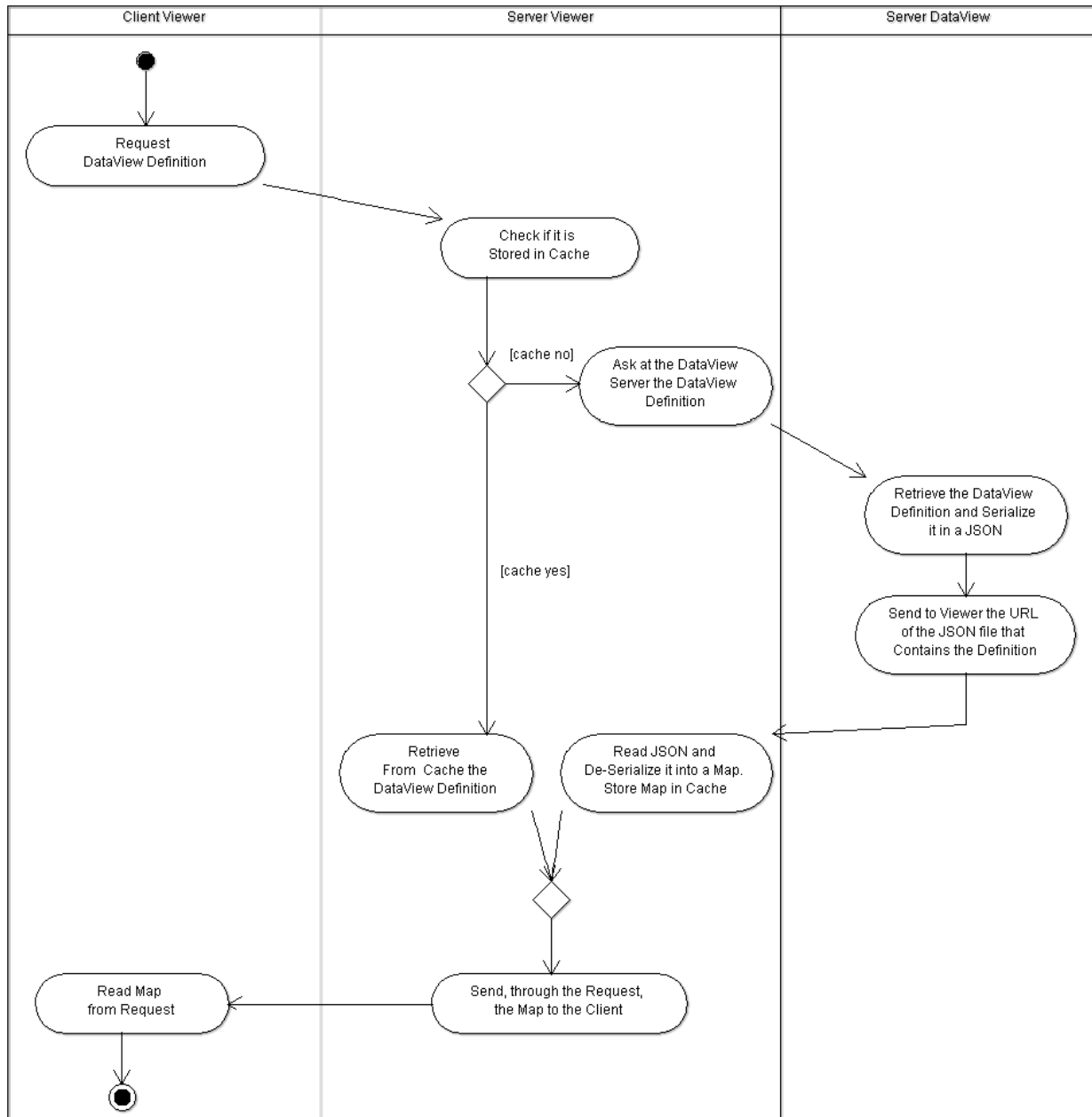


Figure 25 - Activity Diagram (Request Data View Definition)

To remove any doubt about the functioning of the Viewer, this further scheme is going to be used.

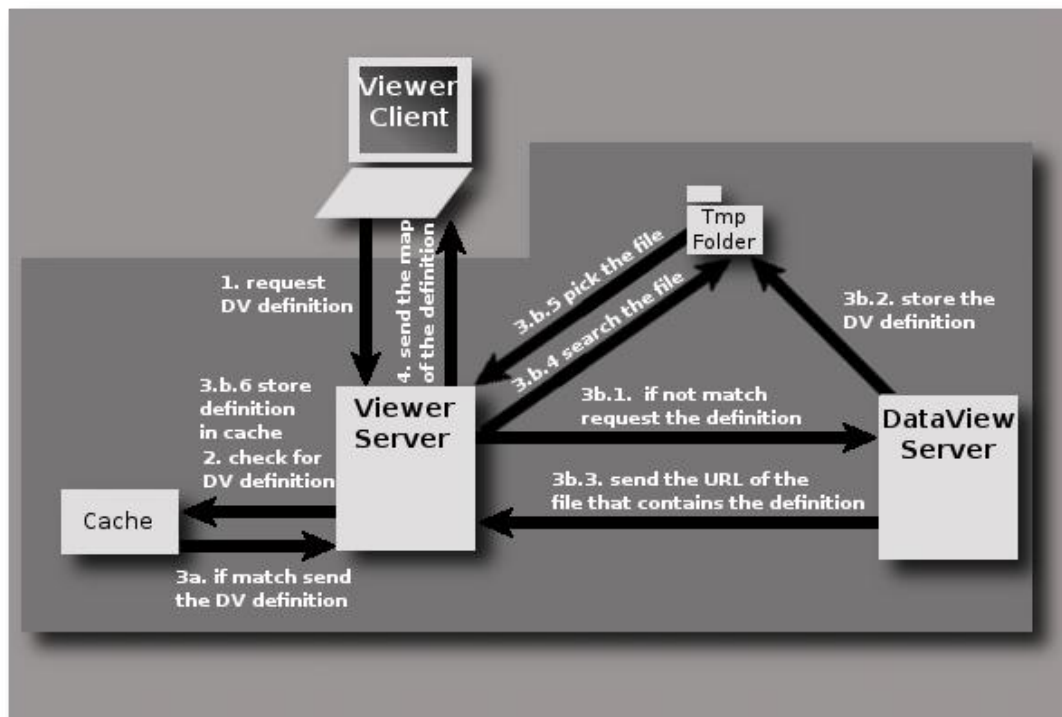


Figure 26 - Request Data View Definition

As it is possible to see, the Viewer client is completely separated from the whole server side. At the moment, both Viewer and DataView servers are installed in the same environment but, in future, they also may be separated.



### 3.2.3 User Guide

In this section the features of the Viewer are going to be taken into account and the ways in which they work. To do that, some screenshots taken from the application and a related Data View specifically designed for the demonstration are going to be used.

This section starts with the assumption that our Data Connection and our Data View are properly configured and working.

The screenshot displays the 'Viewer' application interface. At the top right, there is a user profile icon and the name 'administrator'. Below this is a 'Search' section with a dropdown menu for 'Surname' (currently showing 'Select..'), a text input field for 'Name', and a '+ Advanced' link. There are 'SEARCH' and 'CLEAR' buttons. The main area is titled 'Staff View' and contains a table with columns: Surname, Name, Curriculum, Recruitment Date, and Monthly Fee. Each row includes a set of three icons (edit, delete, print). The table lists 20 staff members, with 'Andrea' highlighted in red. A 'TOTAL' row at the bottom shows a total monthly fee of 27966.25. The page number 'View 1 - 22 of 22' is visible at the bottom right.

Surname	Name	Curriculum	Recruitment Date	Monthly Fee
Fava	Federico	fava-federico.pdf	08/02/2009	1114.32
Franchini	Luca	franchini-luca.pdf	19/02/2011	1111.32
Garavini	Federico	garavini-federico.pdf	02/10/2009	1742.12
Gianaroli	Matteo	gianaroli-matteo.pdf	30/03/2008	1545.44
Giovanelli	Vittorio	giovanelli-vittorio.pdf	06/03/2008	1999.29
Grandi	Stefano	grandi-stefano.pdf	20/11/2009	1000
Menabue	Daniela	menabue-daniela.pdf	01/01/2008	1890.2
Menabue	Giuseppe	menabue-giuseppe.pdf	05/02/2011	1238.33
Menabue	Raffaella	menabue-raffaella.pdf	03/03/2011	980.2
Montorsi	Olga	montorsi-olga.pdf	11/03/2008	1493.22
Pedretti	Sara	pedretti-sara.pdf	08/02/2011	899.99
Pedretti	Cinzia	pedretti-cinzia.pdf	10/12/2009	1329.33
Pellegrino	Roberto	pellegrino-roberto.pdf	05/03/2011	1444.32
Prandini	Andrea	prandini-andrea.pdf	02/01/2011	1200
Rigenti	Alberto	rigenti-alberto.pdf	03/01/2011	1150.49
Rigenti	Luciano	rigenti-luciano.pdf	01/02/2011	999.9
Rigenti	Ettore	rigenti-ettore.pdf	01/01/2010	1785.9
Rizzo	Francesco	rizzo-francesco.pdf	22/07/2010	980.23
Tedeschini	Tiberio	tedeschini-tiberio.pdf	15/05/2009	1132.33
Vanzetti	Emma	vanzetti-emma.pdf	20/10/2010	1200
Vanzetti	Giorgio	vanzetti-giorgio.pdf	05/01/2011	1729.32
<b>TOTAL</b>				<b>27966.25</b>

Figure 27 - The Viewer (Overview)

### 3.2.3.1 Starting Configurations

First of all we have to define our preferences. Preferences have been designed in order to place the Viewer in a Portal, where we can have more than one Viewer, which communicates with each other in the same page. In this way we can, for example, view the Input Parameters only in one Viewer, or we could have one large and two small Viewers.

At present, the application can only make these settings via code in the appropriate Jsp page.

Starting configurations includes:

- Page Height: this parameter allows us to choose the height of the page in pixel. If we set it to *auto*, the page will dynamically grow in height to suit its needs.
- Page Width: this parameter allows us to choose the width of the page in pixel. If we set it to *auto*, the page will expand to the full width available from the parent container.
- Input Parameters View Mode: this parameter changes the viewing mode of the Input Parameters. It is possible to choose from four predefined settings:
  - o ShowMode: in the ShowMode, the Input Parameters are entirely viewed, and we can interact with them to set the query parameters.
  - o CollapseMode: in the CollapseMode, the Input Parameters are hidden, we only view a navigator bar containing the “*Search*” label and the “*Show/Hide*” button. Through this button, it is possible to expand (and collapse) the whole Input Parameters section.
  - o ViewMode: in the ViewMode, we only view the Input Parameters which have a default value. In this view mode, we cannot interact and use the Input Parameters, we can only view their values as labels.
  - o HideMode: in the HideMode, the Input Parameters are not viewed at any time.
- Excel Download Actions: if it is set to *true*, a global action appears in our page and allows us to download an Excel file, dynamically generated from the result set of the current query containing the Output Parameters.

### 3.2.3.2 Call the Data View

To start it, we have to call the Data View by inserting the URL calling the initial action. We can call this URL in two ways:

- Directly from the Web browser;
- By defining a web page called *index.jsp* containing the specified URL. In this way we can just call one Data View predefined in the URL.

The URL is obviously composed of a specific set of parameters. The only required custom parameter to be included is the *id* of the Data View. But we can also specify the *locale* that we would like to use.

*Locale* is a set of parameters defining the user’s language. Through that parameter it is possible to select the wished language by typing the standard abbreviation of your country. For example, if the British language is needed, one can use the abbreviation *en*. By default, if we do not specify any *locale*, the standard is *it*(Italy). In this web application the *locale* is responsible for translating all the labels and all the system messages.

For example, to call a Data View with id `DV_PERSONA_OVERALL` and English language, we have to compose the following URL:

```
viewerContextPath/json.do?task=showDV&_idDV=DV_PERSONA_OVERALL&language=en
```

Where id key is `_idDV`, locale key is `language` and the task to perform is `showDV`.

### 3.2.3.3 User Interface - Administrator Toolbar

The administrator toolbar appears to us only if the user has logged into as Administrator.



Figure 28 - The Viewer (Administrator Toolbar)

It is positioned in the top-right corner of the Viewer and it shows the name of the administrator user logged(1) and three actions:

- Cache Clear.
- Send Input Parameters to Other.
- View Parsed Query.

Cache Clear(2). This action clears the whole system cache and recalls the current Data View with the initial settings. It is very useful when we report changes in the structure of our Data View. If clearing of the cache is not performed, the former definition of the Data View remains in memory so, even doing a page refresh, you would not see any change.

 A screenshot of the Viewer interface. At the top right, a white notification box says 'Cache Cleared! Reload...'. Below it is a search section with fields for 'Surname' and 'Name', and a 'SEARCH' button. The main content is a 'Staff View' table with columns: Surname, Name, Curriculum, Recruitment Date, and Monthly Fee. The table contains five rows of staff data. At the bottom, there is a pagination control showing 'Page 1 of 5' and 'View 1 - 5 of 21'.
 

Surname	Name	Curriculum	Recruitment Date	Monthly Fee
Fava	Federico	fava-federico.pdf	08/02/2009	1114.32
Franchini	Luca	franchini-luca.pdf	19/02/2011	1111.32
Garavini	Federico	garavini-federico.pdf	02/10/2009	1742.12
Gianaroli	Matteo	gianaroli-matteo.pdf	30/03/2008	1545.44
Giovanelli	Vittorio	giovanelli-vittorio.pdf	06/03/2008	1999.29

Figure 29 - The Viewer (Cache Cleared)

As it is possible to notice, if the cleaning operation is successful, a message appears at the top right telling us that a reload will be made.

Send Input Parameters to Other Data Views (3). This action is being developed for a future use of the Viewer as a Portlet. Its role is to send to all the Data Views present in the Portal, the Input Parameters of this Data View. All the Data Views receive these Input Parameters and they catch only the Input Parameters corresponding to theirs. After that, all the Data Views reload themselves with the new Input Parameters. At present, it sends to our Data View a set of predefined Input Parameter.

View Parsed Query(4). This action shows us a hidden div containing the last query parsed by the DataView. It is used by the administrator especially in case of mismatch about the expected and generated result set. For example, the parsed Query(1) is going to be showed if we search by surname two parameters:

The screenshot shows the 'administrator' user interface. At the top right, there are icons for home, refresh, and search, along with the text 'administrator'. Below this is a 'Search' section with a 'Surname' dropdown menu showing '2 Checks' and 'Fava' and 'Franchini' as selected items. There is an empty 'Name' input field and a '+ Advanced' link. Below the search fields are 'SEARCH' and 'CLEAR' buttons. The 'Parsed Query 1' section displays the following SQL query:

```
select *
from persona
where 1=1

and (
cognomeP = 'Fava' or
cognomeP = 'Franchini'
)
```

Below the query is the 'Staff View' section, which contains a table with the following data:

Surname	Name	Curriculum	Recruitment Date	Monthly Fee	
Fava	Federico	fava-federico.pdf	08/02/2009	1114.32	
Franchini	Luca	franchini-luca.pdf	19/02/2011	1111.32	

At the bottom right of the table, it says 'View 1 - 2 of 2'.

Figure 30 - The Viewer (Parsed Query)

### 3.2.3.4 User Interface - Input Parameters

Input Parameters can be of many types, as we have already seen in the corresponding section about the DataView. For every kind, whether String or Integer, Checkbox or PopUp, we have different display modes and data validation.

The screenshot shows the 'Search' section of the interface. The 'Surname' dropdown menu is set to 'Select..'. There is an empty 'Name' input field and a '+ Advanced 3' link. Below the search fields are 'SEARCH' and 'CLEAR' buttons, with the numbers '1' and '2' positioned above them respectively.

Figure 31 - The Viewer (Input Parameters)

As it is visible in the image below, two buttons are present in the bottom of the form: the *Search* button(1) and the *Clear* button(2). The *Search* button allows us to perform the search and the *Clear* button cleans all the Input Parameters. Note that if the Input Parameters have a default value, the cleaning restores these values.

Firstly, in the Viewer, they are subdivided according to whether or not advanced parameters. The advanced parameters block starts hidden, and we can show or hide them with the *Advanced*(3) button.

The screenshot shows a search form with the following elements:

- Surname:** A dropdown menu with a blue arrow and the text "Select..".
- Name:** A standard text input field.
- ID:** A text input field with a calendar icon and a close icon (X).
- Recruitment Date:** A text input field with a calendar icon.
- Curriculum:** A dropdown menu.
- Advanced:** A toggle button labeled "Advanced" with a minus sign.
- Buttons:** Two buttons at the bottom: "SEARCH" (1) and "CLEAR" (2).

Figure 32 - The Viewer (Input Parameters Advanced)

After this first division, we can analyse each Input Parameter based on the *View* parameter.

Default: an Input Parameter in Default View is represented by an HTML *input* tag(1).

The screenshot shows the search form with a *DatePicker* calendar overlaid on the *Recruitment Date* field. The calendar is for March 2011. The form also shows a *Staff View* table at the bottom.

Surname	Name	Recruitment Date	Monthly Fee
Fava	Federico	2009	1114.32
Franchini	Luca	19/02/2011	1111.32

Figure 33 - The Viewer (Input Parameters Default)

Date(2) is the only type that appears in different way, in fact, we find a calendar icon inside it and if a left click, with the mouse, inside the field is being performed, appears to us a jQuery *DatePicker*. The *DatePicker* allows to choose the date from a calendar.

In this mode we have also the Data Validation. Data validation is present only in the Input Parameters which have default *View* and are Integer, Decimal or Date *Type*. It consists in an error

message which is shown if there is a mismatch between the Input Parameter *Type* and what we have inserted into the field.



Figure 34 - The Viewer (Input Parameter Validation)

**DropDownList:** in this view mode, the Input Parameter can have just one value selected from the generated list.

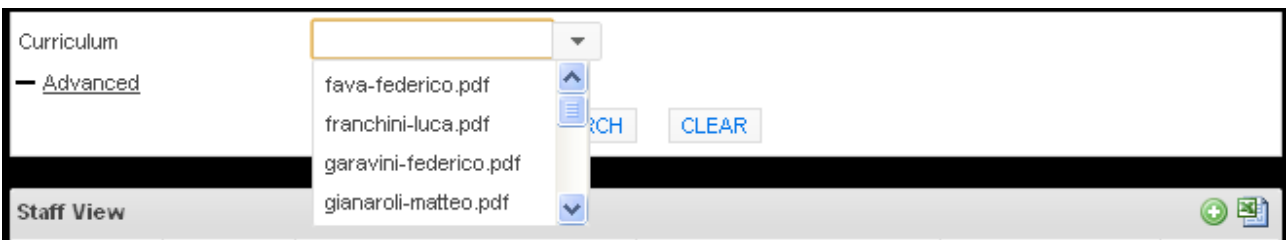


Figure 35 - The Viewer (Input Parameter DropDownList)

The Viewer adds a great feature at the classic drop down list. If text is being typed into the input box of this Input Parameter, the instant search function that shows to us, in real time, just the options that begin with the same chars, is enabled.

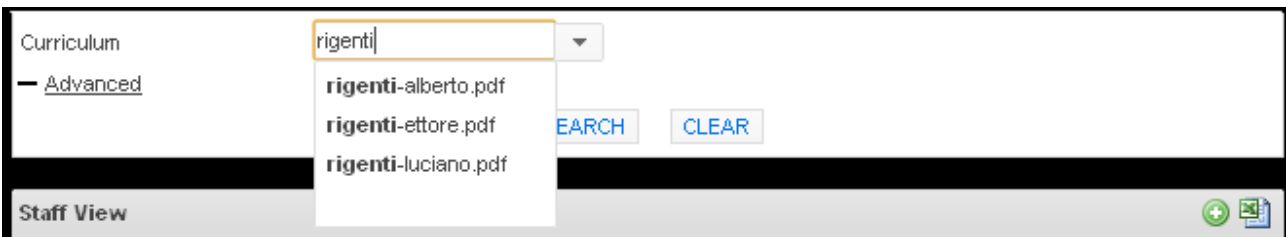


Figure 36 - The Viewer (Input Parameter DropDownList Instant Search)

**CheckBox:** this is the view mode which has been subjected to the major changes among the classic check box.

In the Viewer the checkbox are represented by a drop down list of checkbox. One or more options can be selected and the number of these is shown to us in the Input Parameter label. In addition, each option that we have select is indicated next to the Input Parameter, with a button which allows user to clear the selection.

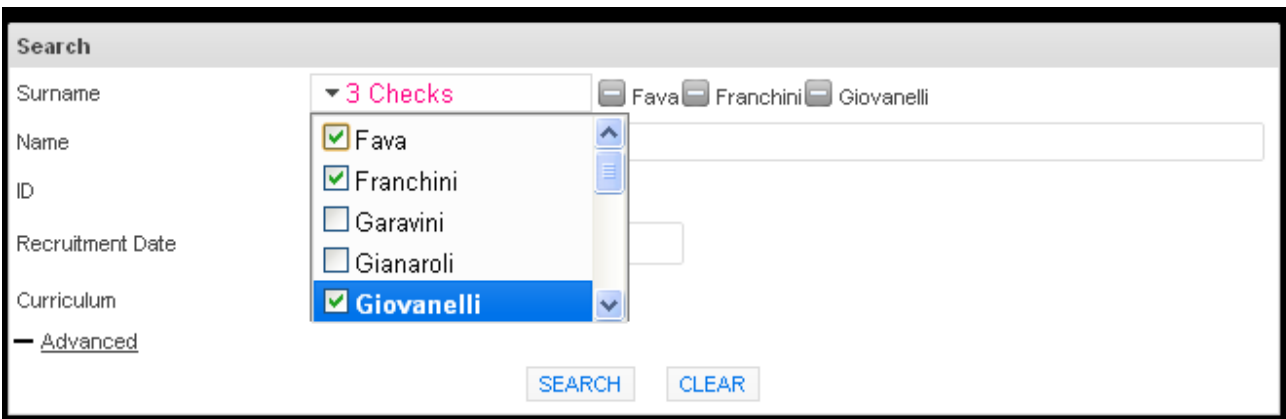


Figure 37 - The Viewer (Input Parameter CheckBox)

PopUp/Custom: this Input Parameter is represented by two buttons and a label.



Figure 38 - The Viewer (Input Parameter PopUp)

The first(1) of these buttons is used to open the popup in which the parameter will be selected. The second(2) button is used to delete the parameter. In the label(3) is shown the parameter that has been selected.

### 3.2.3.5 User Interface - Grid Result Set and Actions

The main grid contains a lot of features and options.

Staff View						
<u>Surname</u> ▲	<u>Name</u>	<u>Curriculum</u>	<u>Recruitment Date</u>	<u>Monthly Fee</u>		
Fava	Federico	fava-federico.pdf	08/02/2009	1114.32		
Franchini	Luca	franchini-luca.pdf	19/02/2011	1111.32		
Garavini	Federico	garavini-federico.pdf	02/10/2009	1742.12		
Gianaroli	Matteo	gianaroli-matteo.pdf	30/03/2008	1545.44		
Giovanelli	Vittorio	giovanelli-vittorio.pdf	06/03/2008	1999.29		
Grandi	Stefano	grandi-stefano.pdf	20/11/2009	1000		
Menabue	Daniela	menabue-daniela.pdf	01/01/2008	1890.2		
Menabue	Giuseppe	menabue-giuseppe.pdf	05/02/2011	1238.33		
Menabue	Raffaella	menabue-raffaella.pdf	03/03/2011	980.2		
Montorsi	Olga	montorsi-olga.pdf	11/03/2008	1493.22		
Pedretti	Sara	pedretti-sara.pdf	08/02/2011	899.99		
Pedretti	Cinzia	pedretti-cinzia.pdf	10/12/2009	1329.33		
Pellegrino	Roberto	pellegrino-roberto.pdf	05/03/2011	1444.32		
Prandini	Andrea	prandini-andrea.pdf	02/01/2011	1200		
Rigenti	Alberto	rigenti-alberto.pdf	03/01/2011	1150.49		
Rigenti	Luciano	rigenti-luciano.pdf	01/02/2011	999.9		
Rigenti	Ettore	rigenti-ettore.pdf	01/01/2010	1785.9		
Rizzo	Francesco	rizzo-francesco.pdf	22/07/2010	980.23		
Tedeschini	Tiberio	tedeschini-tiberio.pdf	15/05/2009	1132.33		
Vanzetti	Emma	vanzetti-emma.pdf	20/10/2010	1200		
Vanzetti	Giorgio	vanzetti-giorgio.pdf	05/01/2011	1729.32		
<b>TOTAL</b>				<b>27966.25</b>		

Figure 39 - The Viewer (Grid Result Set & Actions)

## Toolbar

The grid starts with a toolbar, as the header of the entire table.



Figure 40 - The Viewer (Grid Toolbar)

That toolbar contains in the left position the *Name* of the Data View(1), and in the right the whole global actions(2) plus the *Excel Download Action*(3), if is providing in the initial configurations.

The grid content is composed by the headers of the Output Parameter and by the result set of the query.

## Output Parameter Headers

An Output Parameter header can be or not underline, underline means that the Output Parameter is sortable.

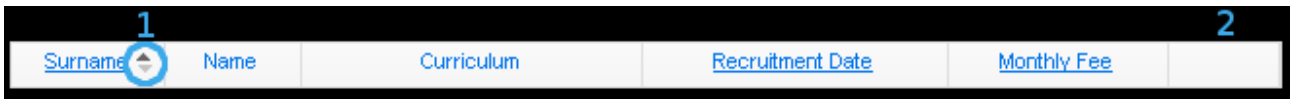


Figure 41 - The Viewer (Output Parameter Headers)

If it is sortable, and the grid is sorted with this Output Parameter, there is two small triangles(1) in the right of the Name, which tells if the order is increasing(apex up) or decreasing(apex down). If there are actions in the Data View, they are displayed in an additional column(2), far right, with no header.

## Grid Data

The grid data do not have a lot of features. If an Output Parameter is String or Date type, the content of the cells is aligned to the left, otherwise to the right.

If a particular Style in the definition of our Data View is been defined, it will applied to the grid cells. For example, in the figure below, the font-color of the Name is set to red.




Prandini	Andrea	prandini-andrea.pdf	02/01/2011	1200	  
----------	--------	---------------------	------------	------	---

Figure 42 - The Viewer (Style Cell)

If there are actions in our Data View, they are displayed in the corresponding column.



Fava	Federico	fava-federico.pdf	08/02/2009	1114.32	  
------	----------	-------------------	------------	---------	---

Figure 43 - The Viewer (Actions)

An action can be represented by an icon(1), if selecting one, or by the action *Name*.



The last two rows contain the label “*TOTAL*”(1) and the amounts of the corresponding Input Parameter(2).











1	TOTAL				
2					27966.25

Figure 44 - The Viewer (Total)



These two lines are shown if at least one Output Parameter has the *Sum* function enabled and, if the number of record per page is -1, or greater than the number of records generated by the initial query.

### Pager

The grid ends with the Pager.

Staff View					
Surname	Name	Curriculum	Recruitment Date	Monthly Fee	
Fava	Federico	fava-federico.pdf	08/02/2009	1114.32	  
Franchini	Luca	franchini-luca.pdf	19/02/2011	1111.32	  
Garavini	Federico	garavini-federico.pdf	02/10/2009	1742.12	  
Gianaroli	Matteo	gianaroli-matteo.pdf	30/03/2008	1545.44	  
Giovanelli	Vittorio	giovanelli-vittorio.pdf	06/03/2008	1999.29	  



Page 1 of 5



2 3 6 4 5
View 1 - 5 of 21

Figure 45 - The Viewer (Pager)

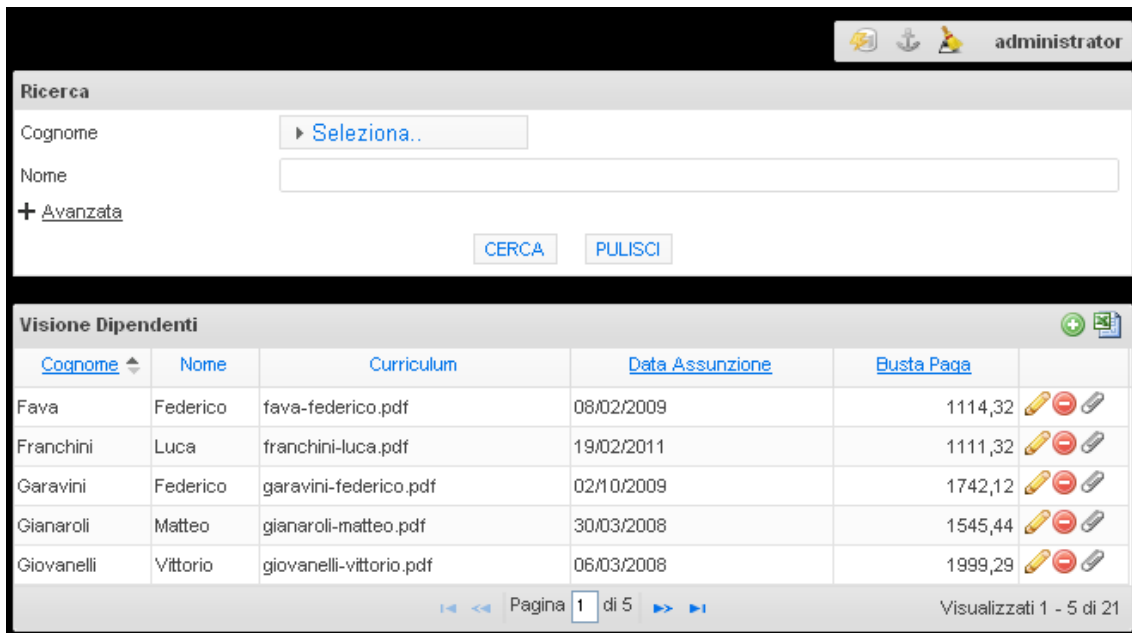
The toolbar contains:

- at the right the range number of records displayed in the current page and the number of total records(1).
- in central position, if more than one page is to be displayed, there is the navigation tool. This contains the buttons to change page (first(2),previous(3),next(4),last(5)), and the number of the current page(6) and total pages.

### 3.2.3.6 User Interface - Locale and Themes

This section reports a couple of screenshots to show the obtained result using the locale parameter and invoking a different theme.

The first one shows the same Data View(*DV\_PERSONA\_OVERALL*) used for the guide, but invoking the server adding the parameter *language=it*, which clearly calls the Italian as language.



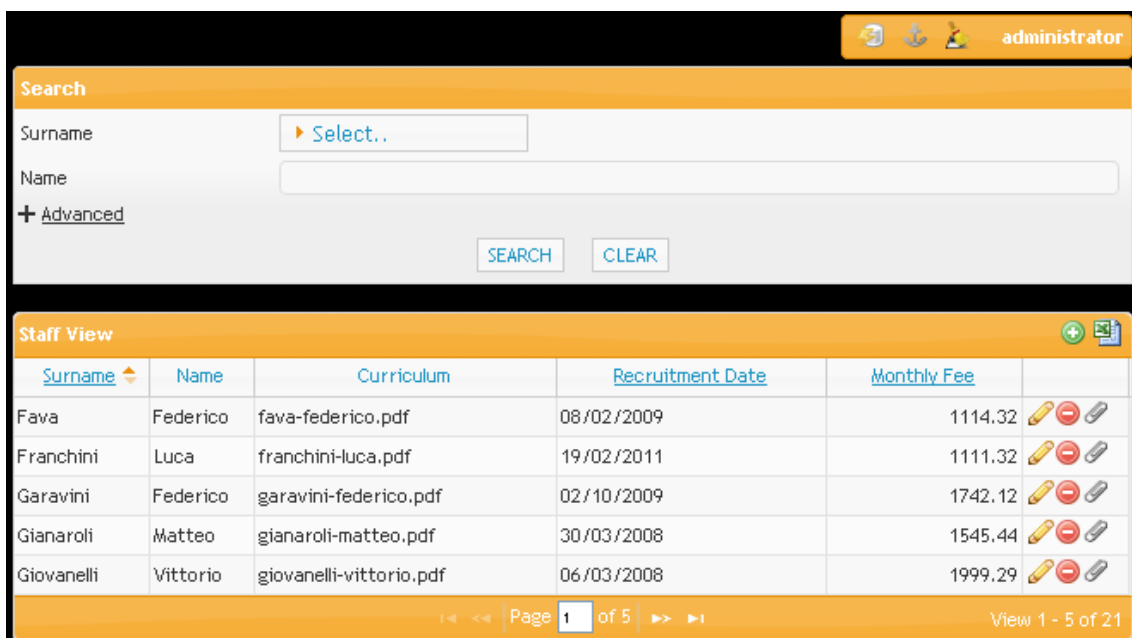
The screenshot shows the application interface in Italian. The search section is titled "Ricerca" and contains input fields for "Cognome" (with a dropdown menu showing "Seleziona..") and "Nome". Below the search fields are buttons for "CERCA" and "PULISCI". The main content area is titled "Visione Dipendenti" and displays a table of employees. The table has columns for "Cognome", "Nome", "Curriculum", "Data Assunzione", and "Busta Paga". The data rows are as follows:

Cognome	Nome	Curriculum	Data Assunzione	Busta Paga
Fava	Federico	fava-federico.pdf	08/02/2009	1114,32
Franchini	Luca	franchini-luca.pdf	19/02/2011	1111,32
Garavini	Federico	garavini-federico.pdf	02/10/2009	1742,12
Gianaroli	Matteo	gianaroli-matteo.pdf	30/03/2008	1545,44
Giovanelli	Vittorio	giovanelli-vittorio.pdf	06/03/2008	1999,29

At the bottom of the table, there is a pagination control showing "Pagina 1 di 5" and a status message "Visualizzati 1 - 5 di 21".

Figure 46 - The Viewer (Locale)

The second one shows the application styled through a different jQuery UI theme. To make it possible, just the name of the theme is to be changed. Therefore, colors, fonts, margin can be changed through switching of just one word.



The screenshot shows the application interface with an alternative jQuery UI theme. The search section is titled "Search" and contains input fields for "Surname" (with a dropdown menu showing "Select..") and "Name". Below the search fields are buttons for "SEARCH" and "CLEAR". The main content area is titled "Staff View" and displays a table of employees. The table has columns for "Surname", "Name", "Curriculum", "Recruitment Date", and "Monthly Fee". The data rows are as follows:

Surname	Name	Curriculum	Recruitment Date	Monthly Fee
Fava	Federico	fava-federico.pdf	08/02/2009	1114.32
Franchini	Luca	franchini-luca.pdf	19/02/2011	1111.32
Garavini	Federico	garavini-federico.pdf	02/10/2009	1742.12
Gianaroli	Matteo	gianaroli-matteo.pdf	30/03/2008	1545.44
Giovanelli	Vittorio	giovanelli-vittorio.pdf	06/03/2008	1999.29

At the bottom of the table, there is a pagination control showing "Page 1 of 5" and a status message "View 1 - 5 of 21".

Figure 47 - The Viewer (Alternative Theme)

## 4 Conclusions and Future Developments

The experience of the training in a company is very satisfactory, and this thesis is the result of the research and development which have been produced. Certainly, changed the way of seeing the world of programming, giving rise to a higher interest, and leading ourselves to want to implement and extend the Viewer application. With the hope of seeing it used and exploited by the customers.

The REST architecture was unknown, and its study allows to understanding how works the majority of web applications in use today. Considering that the World Wide Web is based on it, is certainly a very timely topic and certainly valuable in future.

This thesis has given us the opportunity to understand what a framework is and how to exploit it, as well as understand how indispensable are. We still cannot claim to be able to independently develop a good commercial application, but certainly we are very close.

Since the development time was not very long, the application produced lends itself to a series of implementations and future developments. Below, we find the points divided by category.

### Architectural Implementations:

- Adjust the Viewer in a Portlet, to be included in a Portal environment. This making it ready for use.
- Separation of the server location of the Viewer from the server location of the DataView. Going to meet the Cloud world and all the issues attached, including that of security in the data transfer.

### Functional Implementations:

- Redefining the part of the Input Parameters by adding a customizable column layout. It allows a better optimization of space and greater customization by the user.
- Implementation of the panel that is capable of setting the initial configuration and preferences.
- Improving the Instant Search on DropDownList and CheckBox, going to retrieve data dynamically from the server of the DataView.
- Insertion of the grouping in query results. In order to display the data grid divided into groups, which are selected by the user.
- Inserting a new way of viewing the results through the use of Google Maps. Through the ability to customize the markers and insert customized content into tooltips.
- Add to the DataView the opportunity to associate individual permission for each Data View, which depending on: roles or groups.

### Other Implementations:

- Viewing the Data View, opened within the PopUp windows, using the Viewer.

# Bibliography

[Atif Aziz, 2007] Atif Aziz, Scott Mitchell. *An Introduction to JavaScript Object Notation (JSON) in JavaScript and .NET*.

<http://msdn.microsoft.com>

[FlexJSON] *FlexJSON Reference Site*.

<http://flexjson.sourceforge.net>

[Gavin King] Gavin King, Christian Bauer, Max Rydahl Andersen, Emmanuel Bernard, Steve Ebersole, and Hardy Ferentschik. *Hibernate Reference Documentation 3.6.1 Final*.

<http://www.hibernate.org>

[Jonathan Chaffer, 2009] Jonathan Chaffer, Karl Swedberg . *Learning jQuery 1.3*.

[jQuery] *jQuery Reference Site*.

<http://jquery.com>

[JSON] *JSON Reference Site*.

<http://www.json.org>

[Mokabyte] *Appunti Avanzati di Hibernate*.

<http://www2.mokabyte.it>

[Mokabyte] *La Sessione di Hibernate e la Transazionalità*.

<http://www2.mokabyte.it>

[Packetizer] *Representational State Transfer (REST)*.

<http://www.packetizer.com/ws/rest.html>

[Quix S.r.l.] *Manuale Utente Dataview*.

[Rick Strahl, 2008] Rick Strahl. *An Introduction to jQuery*.

<http://www.west-wind.com/presentations/jquery>

[Roseindia] *jQuery Tutorials and Examples*.

<http://www.roseindia.net/ajax/jquery/index.shtml>

[Roseindia] *Struts Architecture*.

<http://www.roseindia.net/struts/index.shtml>

[Roy Thomas Fielding, 2000] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*.

[Ryan McGreal, 2010] Ryan McGreal. *Designing a RESTful Web Application*.

<http://quandyfactory.com/blog>

[Sameer Tyagi, 2006] Sameer Tyagi. *RESTful Web Services*.  
<http://www.oracle.com>

[SearchSOA] *REST (Representational State Transfer)*.  
<http://searchsoa.techtarget.com/definition/REST>

[The Apache Software Foundation] *Struts User Guide*.  
<http://struts.apache.org/1.x/userGuide/index.html>

[Wikipedia] *Used for the glossary*.  
<http://en.wikipedia.org>

# Glossary

**Activity Diagram.** Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system.

**ASP.** Active Server Pages (ASP), also known as Classic ASP or ASP Classic, was Microsoft's first server-side script-engine for dynamically-generated web pages.

**CGI.** The Common Gateway Interface (CGI) is a standard (see RFC 3875: CGI Version 1.1) that defines how web server software can delegate the generation of web pages to a stand-alone application, an executable file. Such applications are known as CGI scripts; they can be written in any programming language, although scripting languages are often used.

**Connection Pool.** A connection pool is a cache of database connections maintained so that the connections can be reused when future requests to the database are required. Connection pools are used to enhance the performance of executing commands on a database.

**DOM.** The DOM (Document Object Model) is a family-tree structure of sorts. HTML, like other markup languages, uses this model to describe the relationship of things on a page.

**JavaBeans.** JavaBeans are reusable software components for Java that can be manipulated visually in a builder tool. Practically, they are classes written in the Java programming language conforming to a particular convention. They are used to encapsulate many objects into a single object (the bean), so that they can be passed around as a single bean object instead of as multiple individual objects.

**JDBC.** Java DataBase Connectivity, commonly referred to as JDBC, is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database.

**JSP.** JavaServer Pages (JSP) is a Java technology that helps software developers serve dynamically generated web pages based on HTML, XML, or other document types.

**Liferay.** Liferay Portal is a free and open source enterprise portal written in Java and distributed under the GNU Lesser General Public License. It is primarily used to power corporate intranets and extranets.

**MySQL.** MySQL is a relational database management system (RDBMS) that runs as a server providing multi-user access to a number of databases.

**ORM.** Object-relational mapping (ORM, O/RM, and O/R mapping) in computer software is a programming technique for converting data between incompatible type systems in object-oriented programming languages.

**PHP.** PHP is a general-purpose scripting language originally designed for web development to produce dynamic web pages. For this purpose, PHP code is embedded into the HTML source document and interpreted by a web server with a PHP processor module, which generates the web page document.

**Portal.** An enterprise portal, also known as an enterprise information portal (EIP) or corporate portal, is a framework for integrating information, people and processes across organizational boundaries. It provides a secure unified access point, often in the form of a web-based user interface, and is designed to aggregate and personalize information through application-specific portlets.

**Portlet.** Portlets are pluggable user interface software components that are managed and displayed in a web portal. Portlets produce fragments of markup code that are aggregated into a portal.

**Relational Database.** A relational database matches data by using common characteristics found within the data set. The resulting groups of data are organized and are much easier for many people to understand.

**Servlet.** A Servlet is a Java class in Java EE that conforms to the Java Servlet API, a protocol by which a Java class may respond to HTTP requests.

**SOAP.** SOAP, originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks.

**SQL.** SQL, often referred to as Structured Query Language, is a database computer language designed for managing data in relational database management systems (RDBMS), and originally based upon relational algebra and calculus.

**URI.** A Uniform Resource Identifier (URI) is a string of characters used to identify a name or a resource on the Internet. Such identification enables interaction with representations of the resource over a network (typically the World Wide Web) using specific protocols.

**XML.** Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form. It is defined in the XML 1.0 Specification produced by the W3C, and several other related specifications, all gratis open standards.