UNIVERSITÀ DEGLI STUDI
DI MODENA E REGGIO EMILIA

Facoltà di Scienze Fisiche, Informatiche e Matematiche
Corso di Laurea in Informatica

Qery Mobile:
Progetto e Sviluppo di un'Applicazione Web per la Gestione di Dashboard

Samantha Miceli

Tesi di Laurea

Relatore:
Prof. Riccardo Martoglia

Anno Accademico 2014/2015

RINGRAZIAMENTI

Ringrazio

l'Ing. Riccardo Martoglia per la continua disponibilità e assistenza,

l'azienda Quix s.r.l per avermi dato l'occasione di lavorare su questo progetto,

la mia famiglia che in questi anni mi ha sempre sostenuto e sopportato.

PAROLE CHIAVE

QeryMobile

ApplicazioneWeb

Dashboard

Java

AngularJS

Indice

Introduzione 1

Capitolo 1 3

Qery 3

1. Introduzione a Qery 3

1.1 Struttura dell'applicazione: Admin **4**

1.1.1 Connessione Dati 4

1.1.2 Progetti, DataView, Grafici, Mappe e Custom 5

1.1.3 Altre funzionalità di Qery Admin 16

1.2 Struttura dell'applicazione: Viewer e Portale Aziendale **17**

1.3 Database **20**

1.3.1 Schema E/R 20

1.3.2 Progetto Logico 21

1.3.3 Descrizione delle Tabelle 23

Capitolo 2 29

Tecnologie Lato Server 29

2. Introduzione alle Tecnologie dell'applicazione 29

2.1 Struts2 ..... **30**

2.1.1 Breve panoramica su HTTP e Servlet 30

2.1.2 Introduzione a Struts2 (Modello MVC) 32

2.1.3 Ciclo di Vita di una Richiesta in Struts2 34

2.2 Spring **37**

2.2.1 Caratteristiche Principali di Spring 37

2.2.2 Architettura di Spring 38

2.3 Framework Quix **42**

Capitolo 3 48

Tecnologie Lato Client 48

3.1 JavaScript **48**

3.1.1 Introduzione a JavaScript 48

3.1.2 Caratteristiche di JavaScript 49

3.1.3 Elementi del linguaggio JavaScript 50

3.2 jQuery 53

3.2.1 Introduzione a jQuery UI 56

3.3 AngularJS 57

3.3.1 Attributi ng- 58

3.3.2 MVC 60

3.3.3 Il DataBinding 62

3.4 Comunicazione tra Client e Server: JSON 64

3.4.1 FlexJSON 65

4. Introduzione all'applicazione Qery Mobile 69

4.3 Struttura dell'applicazione: Chiamate REST 95

4.3.1 Servizio di Login 95

4.3.2 Servizio di erogazione dell'albero 98

4.3.3 Servizio di definizione Dashboard 105

4.4 Sviluppi Correnti e Futuri 107

4.4.1 Sviluppi Correnti: Dashboard Viewer 107

4.4.2 Sviluppi Futuri 109

Bibliografia 110

Elenco delle figure

1.1	Creazione di una nuova connessione dati	p.4
1.2	Lista dei Progetti	p.6
1.3	Pannello di Creazione/Modifica di un DataView – Lista Filtri	p.7
1.4	Gestione Filtri	p.8
1.5	Sezione query p.9	p.10
1.6	Lista Campi di Output	p.11
1.7	Pannello di Gestione Azioni	p.13
1.8	Lista dei Dataview	p.14
1.9	Pannello di Gestione Grafici	p.15
1.10	Gestione Mappe	p.16
1.11	Esecuzione di un Dataview	p.18
1.12	Visualizzazione di una mappa	p.18
1.13	Visualizzazione di un Grafico	p.19
1.14	Schema E.R.	p.20
2.1	Stack of Web Application	p.30
2.2	MVC Pattern in Struts2	p.32
2.3	Elaborazione di una richiesta in Struts2	p.34
2.4	Architettura di Spring	p.38
2.5	Framework – Struttura di un Progetto generato	p.43
2.6	Framework – File di configurazione	p.46
2.7	Framework – folder main/webapp	p.47
3.1	Two-way binding	p.62
4.1a	Schema E/R Qery Mobile	p.74
4.1b	Schema E/R Qery Mobile con attributi	p.75
4.2	Schema E/R finale	p.76

4.3	Pannello di controllo Dashboard	p.78
4.4	Albero di Navigazione	p.79
4.5	Form Modifica Dashboard	p.83
4.6	Autocompletamento	p.84
4.7	Modale di Composizione Dashboard	p.85
4.8	Esempi di Layout di una Pagina	p.86
4.9	Esempio di Drag&Drop	p.87
4.10	Modale di Debug	p.91
4.11	Loading di cg-busy	p.92
4.12	Alert Campo Richiesto	p.93
4.13	Tipi di sweet-alert	p.94
4.14	Esempio dell’applicazione mobile	p.95
4.15	Risposta al servizio di Login	p.96
4.16°	1° Parte del JSON dell’albero di navigazione	p.98
4.16b	2° Parte del JSON dell’albero di navigazione	p.99
4.17	JSON di definizione Dashboard	p.106
4.18	Barra di Navigazione	p.108
4.19	Esecuzione Dashboard	p.108
4.20	Il nuovo Qery	p.109

Introduzione

Il mio percorso di formazione lavorativa ha avuto inizio grazie al tirocinio presso Ceramiche Ricchetti S.p.A.
In questo periodo di tempo ho avuto modo di imparare ad utilizzare lo strumento Dataview in tutti i suoi ambiti creando report aziendali per il portale aziendale di Ricchetti.
Successivamente, a tirocinio terminato, ho avuto l'occasione di iniziare a lavorare per la società Quix S.r.l.
In Quix S.r.l ho seguito un corso di formazione di un mese, ed in seguito ho iniziato a lavorare, nell'ambito dello sviluppo vero e proprio, all'applicazione web Dataview rinominata oggi Qery.
Partendo da questa applicazione, si è provveduto ad estendere le sue funzionalità introducendo nuove tecnologie per migliorarne il funzionamento.
Quasi un anno fa è poi partito un nuovo progetto riguardante Qery: portare l'applicazione su dispositivi mobili.
Si è pensato di creare una nuova struttura per questa applicazione, che poggiasse sulla precedente e che riuscisse a visualizzare più Dataview senza l'aiuto di un portale Liferay.
Così sono nate le Dashboard e quella che oggi chiamiamo Qery Mobile.
Qery Mobile è composta da un'applicazione mobile e da un pannello di amministrazione delle Dashboard, oggetto della nostra trattazione.
Inoltre per rendere più chiara la trattazione verrà introdotta l'applicazione Qery (su cui si basa Qery Mobile) e le tecnologie utilizzate nello sviluppo della nuova applicazione web.
La struttura di questa tesi è organizzata perciò in quattro capitoli illustrati di seguito.
Il capitolo 1 contiene l'analisi dell'applicazione Qery; nel capitolo viene data un idea generale del suo utilizzo e viene spiegata la struttura generale dell'applicazione stessa.

Nel capitolo 2, vengono presentate le nuove tecnologie lato Server utilizzate nell'applicazione Qery e Qery Mobile. Tali tecnologie sono Struts2, Spring ed il Framework Quix.
Mentre il capitolo 3 contiene la descrizione delle tecnologie lato Client, come Javascript e jQuery, presentate in quanto sono elementi base per il funzionamento di AngularJS.
Infine nel capitolo 4 si parla dell'applicazione Qery Mobile; in particolare dell'implementazione del pannello di gestione delle Dashboard e delle chiamate API REST, le quali rendono possibile il funzionamento dell'applicazione mobile.

Capitolo 1

Qery

1. Introduzione a Qery

Questa introduzione ha lo scopo di mettere il lettore in condizioni di capire l'architettura preesistente presente dietro all'applicazione Qery Mobile, oggetto della trattazione.

Qery è un applicazione web progettata da Quix s.r.l, nata per la reportistica in ambito aziendale.

Qery è lo strumento che mette a disposizione dell'utente metodi semplici ed efficaci per la raccolta di dati aziendali, mediante oggetti grafici chiamati Dataview.

Grazie a tale applicazione qualsiasi azienda può generare report e visualizzarli mediante interfaccia web, senza preoccuparsi di nessun dettaglio implementativo.

L'utente dovrà soltanto scrivere una semplice query SQL, che interroghi il database per recuperare i dati desiderati.

I Dataview si occupano di far visualizzare al cliente, i dati dell'azienda in modo pulito e trasparente, rispetto a quello che sarebbe lo standard mediante interrogazione SQL. Vengono così effettuati report che sono memorizzati nell'oggetto Dataview il quale, nel momento in cui sarà eseguito, riporterà sotto forma tabellare tutti i dati di quella determinata interrogazione.

Da queste interrogazioni si possono poi estrapolare i dati per la creazione di grafici e mappe correlati.

Tali oggetti servono a rendere completa l'interrogazione su Database, in quanto rendono possibile la visualizzazione dei dati in un formato diverso da quello tabellare e più user-friendly.

Qery è strutturato in Qery Admin e Qery Viewer.

In tale trattazione si parlerà nello specifico dell'interfaccia Admin, in quanto è quella in cui è stata implementata la nuova funzionalità di Qery.

Inoltre si accennerà qualche funzionalità del nuovo Viewer.

1.1 Struttura dell'applicazione: Admin

L'interfaccia di Admin è stata progettata per la creazione di Dataview, grafici e mappe, contiene perciò una serie di strumenti per interconnettersi al database e per creare e gestire Dataview.

1.1.1 Connessione Dati

La prima funzionalità di cui andiamo a parlare è quella della connessione dati anche chiamata DataConnection, tale funzionalità è stata implementata per fare in modo di gestire multiple connessioni ai database.
Entrando nel menu delle connessioni è possibile visualizzare la lista di connessioni attive, modificare, cancellare o aggiungerne di nuove.

Figura 1.1 : Creazione di una nuova connessione dati

Come raffigurato in *figura 1.1,* per creare una nuova connessione dati è necessario specificare tutti i dati relativi al database, al quale ci si vuole connettere ed assegnarci un nominativo univoco.
I dati che vengono richiesti per creare una nuova connessione dati sono i seguenti:

- ID: identificativo univoco della connessione dati
- URL: url del database a cui si vuole effettuare la connessione
- Username: utente con cui si accede al database
- Password: password dell'utente con cui si accede al database
- Tipologia di connessione al DataBase

  Data Source: occorre specificare solo l'attributo URL, in quanto viene effettuato la richiesta della connessione al datasource (il quale provvede ad instanziare n connessioni all'avvio del sistema e a renderle disponibili all'applicazione)

  Data Connection: occorre specificare tutti gli attributi per la connessione al database in quanto selezionando tale opzione si apre fisicamente una nuova connessione al database
- DRIVER del database.

Una volta compilati tutti i campi necessari è possibile salvare la nuova connessione dati, in modo da utilizzarla nei nostri Dataview.
Possiamo notare che, nel momento in cui tentiamo di salvare la nostra nuova connessione dati, questa viene prima testata e il salvataggio va a buon fine soltanto se il test risulta positivo.

1.1.2 Progetti, DataView, Grafici, Mappe e Custom

Parliamo ora della parte principale dell'applicazione: i Dataview.
La home dell'applicazione contiene la lista dei Progetti (*figura 1.2*).
I Dataview sono organizzati in Progetti, ogni progetto può contenere infiniti Dataview comportandosi come una cartella.

Tale organizzazione permette di tenere ordinati i report che sono correlati tra loro e di creare una struttura gerarchica nell'applicazione.

Le funzioni legate ai progetti sono: creazione di un progetto, modifica di un progetto, eliminazione di un progetto ed importazione o esportazione di un progetto.

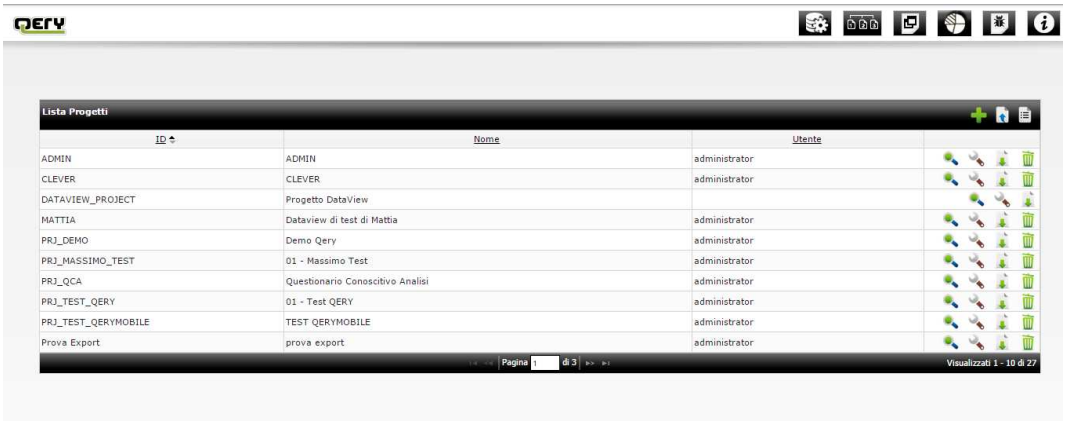

Figura 1.2: Lista dei Progetti

Per la creazione di un progetto è necessario indicare il codice e il nome del progetto stesso; una volta creato il progetto è possibile decidere di visualizzare la lista dei Dataview al suo interno e di creare od importare al suo interno altri Dataview.

Nel momento in cui si sceglie di creare o modificare un Dataview, viene visualizzato un form in cui è possibile specificare le impostazioni del Dataview.

Tale form contiene diversi parametri da impostare come l'id, il nome e la descrizione del Dataview, la sorgente dati, i record per pagina, i parametri di input, la query, i campi di output e le azioni.

Ora andiamo a descrivere le funzionalità dell'oggetto DataView.

La prima serie di funzionalità da includere per la creazione di un nuovo DataView presenti in *figura 1.3* sono le seguenti:

- ID: identificativo univoco del Dataview.
- Nome: descrizione breve del DataView
- Progetto: progetto a cui appartiene il DataView
- Tipo di Dataview: DataView Standard o di Sistema.
- Sorgente dati: apertura della connessione al database o richiesta di una connessione esistente al Datasource.

- Descrizione del contenuto dei dati.
- Numero di record per pagina: riguarda il numero di record che l'utente vuole che vengano visualizzati per ogni pagina nell'anteprima del Dataview. Se questo attributo è "-1" vedremo tutti i record.

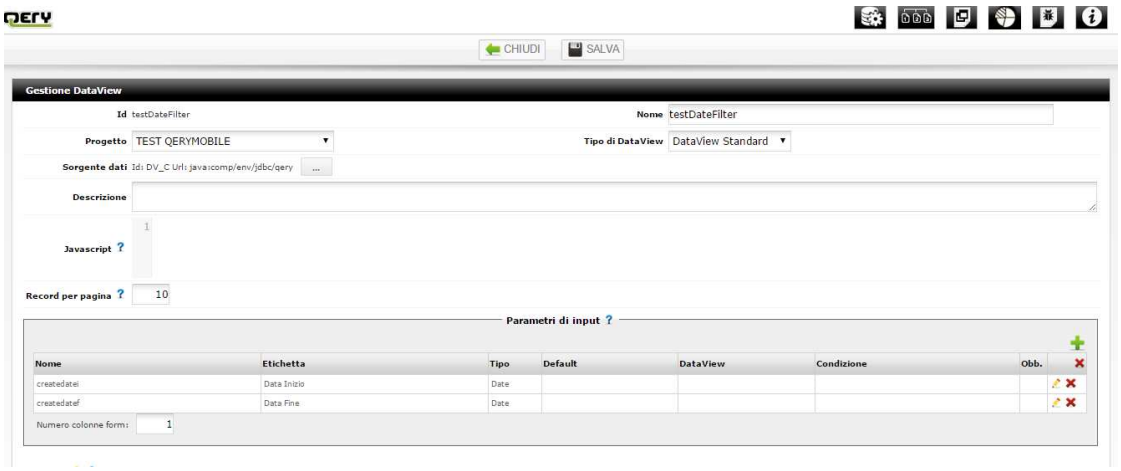

Figura 1.3: Pannello di Creazione/Modifica di un DataView – Lista Filtri

I parametri di Input sono nati per applicare filtri a tempo di esecuzione della query, ossia per impostare condizioni a seconda dei dati richiesti dall'utente.

In *figura 1.4* viene visualizzata la sezione adibita alla creazione o alla modifica dei parametri di input. In tale sezione possiamo creare un filtro il quale verrà impostato dall'utente, nel momento in cui il Dataview sarà eseguito; il Dataview risponderà adeguatamente ritornando i record relativi a tale filtro.

I parametri di input sono composti da diversi attributi:

- Etichetta: nome della colonna del filtro.
- Tipo: scelta del tipo del parametro di ingresso: stringa, decimale, intero o data.
- Text Case: se il tipo di parametro di ingresso è una stringa, possiamo scegliere se il parametro di input è case sensitive oppure no.
- Obbligatorio: se è impostata questa opzione il parametro di ingresso è richiesto.
- Ricerca avanzata: se un parametro d'ingresso presenta tale opzione questo è visualizzabile solo su richiesta dell'utente.

- Nascosto: il parametro di input non è visibile (spesso tale funzionalità è utilizzata nei Dataview che ricevono parametri in ingresso ascoltando i filtri in sessione o da altri Dataview)
- Valore di default: è il valore predefinito del parametro di ingresso.
- Colspan: numero di colonne occupate dal filtro
- Visualizzazione: Si può scegliere tra diversi tipi di parametro di input: default, dropdownlist, popup checkbox:
    - Default: il parametro è una casella di input libero dove è possibile inserire stringhe, numeri o date.
    - DropDownList: il parametro di input è una lista che viene composta mediante un altro Dataview.
    - PopUp: apre una nuova finestra, questo tipo di parametro viene utilizzato per parametri di ingresso complessi.
    - CheckBox: ogni elemento del parametro di input è rappresentato da una casella checkbox.
- Condizione: permette di gestire tali parametri mediante delle condizioni Freemarker e di visualizzare il parametro di input, solo se sono verificate determinate condizioni.

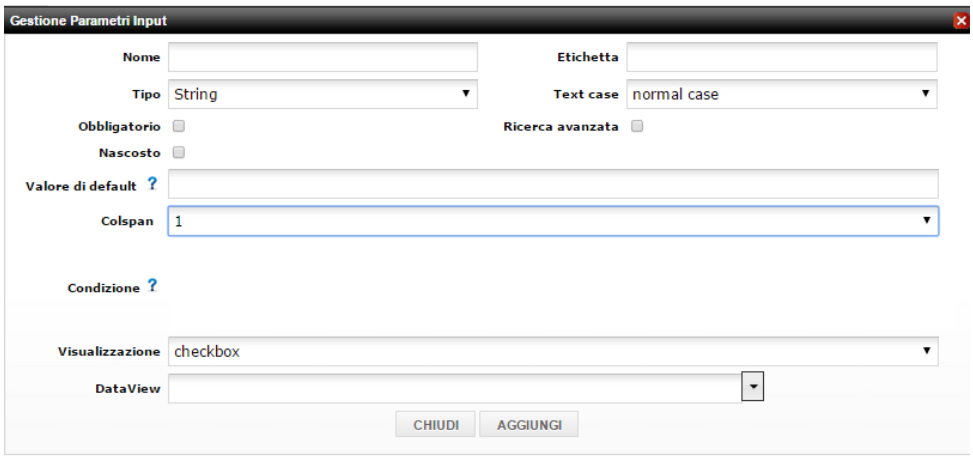

Figura 1.4: Pannello dei Gestione Parametri di Input

Una volta creati i nostri filtri, per far sì che questi funzionino correttamente, occorrerà impostare dopo la clausola WHERE della query, le condizioni freemarker

adeguate. In tale modo la query che verrà eseguita dal Dataview conterrà anche le condizioni prescelte dall'utente mediante i filtri. (vedi query condizionali).

La sezione Query (*figura 1.5*) è dedicata alla scrittura della querySQL, contiene una text area dove deve essere inserita la query, che dovrà essere eseguita nella visualizzazione del Dataview.
Il sistema non permette l'utilizzo di clausole diverse dalla SELECT. La query dovrà essere scritta rispettando la struttura delle tabelle e la sintassi SQL prevista dal DBMS utilizzato.
In tale sezione possono essere inserite query parametriche in cui vengono utilizzati dei parametri, i quali sono definiti opportunamente tramite le funzioni messe a disposizione.
Per esempio, si possono inserire parametri impliciti come lo username e la lingua dell'utente per filtrare i risultati:

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

Oltretutto è possibile creare query condizionali i cui parametri possono essere soggetti a condizioni, tale condizioni sono definite mediante il linguaggio di scripting Freemarker.
Come esempio illustriamo il passaggio dei parametri di input o filtri che vengono inseriti nella query soltanto se questi esistono e sono stati settati dall'utente:

```
select createdate,updatedate
from dv_dashboard
WHERE 1=1
<#if (createdatei?exists)> and ( createdate>:createdatei ) </#if>
<#if (createdatef?exists)> and ( createdate<:createdatef ) </#if>
```

Nella sezione Query è anche presente la funzionalità Test query.
Tale funzionalità è stata progettata per effettuare un test sull'effettivo funzionamento della query prima che il Dataview venga salvato. Questo test include

eventuali messaggi di errore, che vengono visualizzati nel momento in cui la query digitata non sia corretta.

Tale sezione presenta anche una funzionalità di HELP ed un suggeritore di codice (vedi *figura 1.5*).

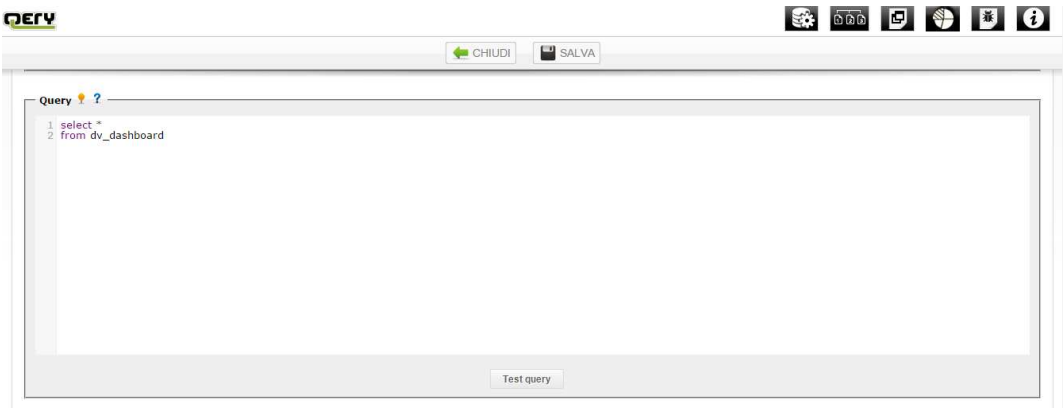

Figura 1.5: Pannello di Creazione/Modifica di un DataView – Sezione query

I campi di output sono l'insieme di parametri che vogliamo visualizzare nell'esecuzione del nostro Dataview, ossia le colonne provenienti dall'interrogazione SQL definita nella sezione query; possiamo trovare la descrizione della sezione relativa a tali parametri in *figura 1.6.*

Funzionalità della sezione campi di output:

- Inserisci nuovo campo di output: aggiunta di una o più colonne recuperate dai results della query
- Elimina: cancella i campi di output selezionati o quello della riga corrente.
- Ord: indica se quel parametro di output può essere ordinato oppure no nella tabella di output
- Vis: indica se quel parametro di output è visibile oppure no nella tabella di output
- Σ: questa funzione effettua una somma di tutti i valori di tale parametro di output.
- Nome: nome della colonna della tabella su database
- Etichetta: permette di rinominare l'intestazione del parametro di output nel Dataview.

- Tipo: il tipo del parametro di output questo può essere una Stringa, un Intero, un Decimale, una Data, un Booleano o un File.

- Formato: in questo campo viene impostato il formato del parametro di output. Ad esempio, con una data possiamo utilizzare tale campo per impostare il formato in “dd/mm/yyyy”.

-Condizioni: Si può condizionare il campo di Output tramite espressioni Freemarker le quali indicano se il campo di output deve essere visibile o no.

- Stile: con tale opzione si imposta lo stile HTML per i parametri di output. È possibile inserire direttamente lo stile html che verrà applicato al campo oppure condizionarlo tramite espressioni Freemarker. Tale opzione Viene utilizzata nel caso si voglia modificare la grafica della tabella evidenziando per esempio con un colore diverso una colonna o una riga della stessa.

Grp: raggruppamento dei Dataview per un Campo di Output.

- Ordinamento di Default e Tipo di ordinamento: alla fine della sezione dei parametri di output troviamo queste due opzioni in cui si può selezionare il campo per cui si vuole effettuare l’ordinamento dei risultati e il tipo di ordinamento che si vuole effettuare (crescente o decrescente).

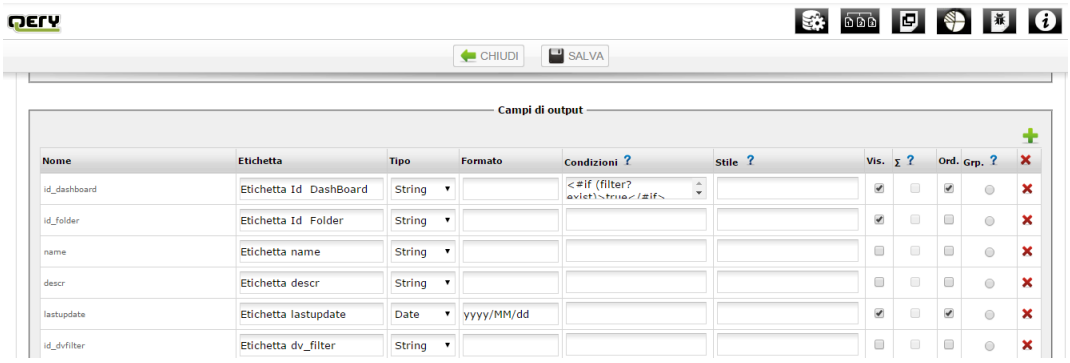

Figura 1.6: Pannello di Creazione/Modifica di un DataView – Lista Campi di Output

L’ultima sezione presentata nella finestra di creazione/modifica del Dataview è quella delle azioni, queste azioni sono delle operazioni che possono essere aggiunte alle tabelle di output e si suddividono in azioni locali o azioni globali.

Funzionalità della sezione Azioni:

-Nome: nome dell'azione.

- Tipo: un’azione può essere di diversi tipi:
    - URL: scelto se si vuole chiamare l'URL di una pagina che contiene l'azione.
    - JavaScript: prevede la scrittura di codice JavaScript.
    - Comunicazione: create per il passaggio di parametri tra due Dataview.
- Contesto:
    - Azione Globale: questa è definita per tutto il Dataview e viene ripetuta per ogni campo del results set.
    - Azione Locale: definita per ogni record.
- PopUp: l’azione che verrà creata aprirà una finestra quando questa verrà richiamata
- Dynamic Contest: indica se il contesto è dinamico.
- Icona: permette di scegliere l'icona associata all'azione. Se viene selezionata la voce “nessuna icona” viene utilizzato il nome dell’azione per la sua identificazione.
- Parametri: selezione dei parametri di input e di output coinvolti nell'azione.
- Condizione: scrittura di una condizione JavaScript che ritorni un valore TRUE o FALSE per fare in modo di visualizzare o no l’azione a seconda che questa condizione sia o meno verificata.

Le azioni locali solitamente vengono effettuate per il passaggio di parametri tra due Dataview. Il primo Dataview definisce un’azione di tipo comunicazione in cui si seleziona il parametro o campo di output del record, tale campo verrà inviato al richiamo dell’azione. Nel secondo Dataview verrà definito un parametro di input con lo stesso nome del parametro definito dall’azione del primo Dataview.

Mentre le azioni globali sono quelle definite sull'intero Dataview, un esempio di tali azioni può essere l'export di un file pdf.

In *figura 1.7* è presente l'interfaccia di creazione o modifica delle azioni.

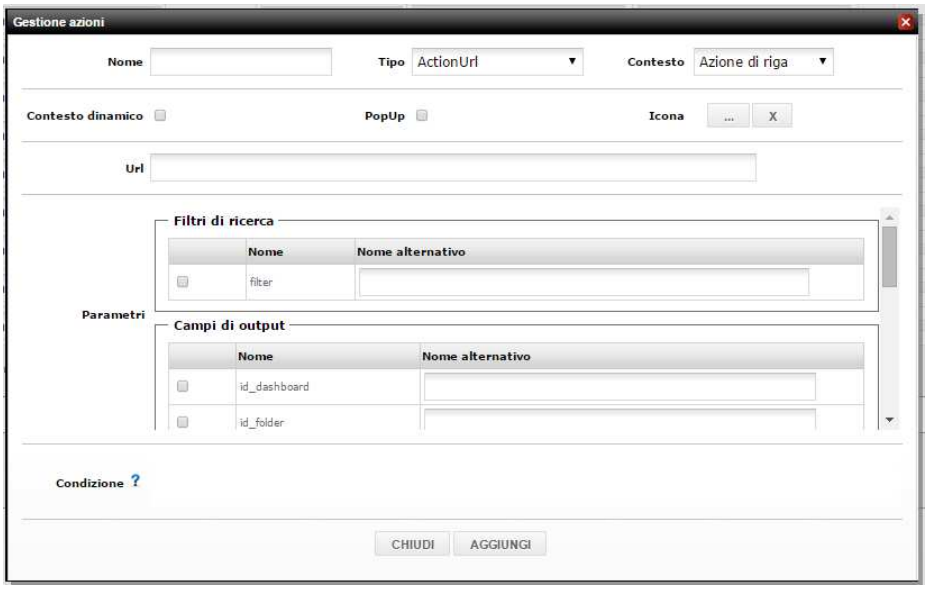

Figura 1.7: Pannello di Gestione Azioni

Un Dataview può contenere Grafici, Mappe e Custom.

Questi elementi sono legati ad un Dataview in quanto essi recuperano i dati necessari per effettuare le operazioni dal Dataview padre, il quale deve perciò esporre i dati necessari a tali elementi.

È possibile trovare la lista dei grafici, delle mappe e dei custom nelle opzioni accanto ad ogni Dataview.

Perciò, come visualizzato in *figura 1.8*, accanto ad ogni Dataview troviamo l'opzione Grafico, Mappa e Custom.

Cliccando su ciascuna opzione si accede ad un pannello simile al precedente, contenente la lista di tutti gli elementi creati ed un'azione per modificare, visualizzare o creare nuovi elementi.

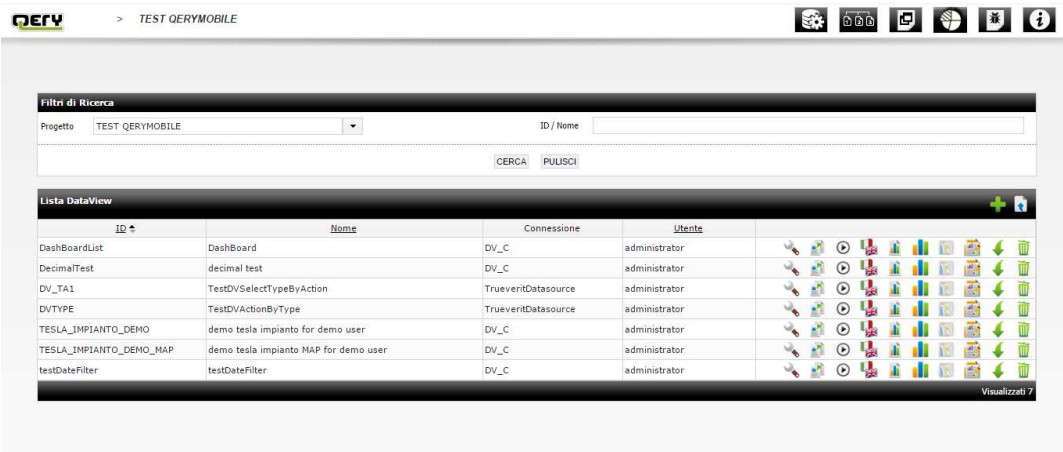

Figura 1.8: Lista dei Dataview

Mediante l'opzione Grafico si accede alla lista dei grafici e mediante l'opzione nuovo o modifica si apre il pannello di creazione/modifica di un grafico.

Tale pannello viene visualizzato in *figura 1.9*, ed è composto da:

- Nome: breve descrizione del grafico.
- Tipo: scelta del tipo di grafico da visualizzare
  - Linee
  - Pila
  - Istogramma
  - Torta.
- Descrizione: descrizione di cosa contiene il grafico.
- Larghezza: larghezza in pixel del grafico.
- Altezza: altezza in pixel del grafico.
- Etichetta X: nome dell'etichetta dell'asse x
- Etichetta Y: nome dell'etichetta dell'asse y.
- Categoria: imposta la categoria della tabella (scelta tra i parametri di output del Dataview).
- Colore: è possibile decidere il colore che ogni dato avrà nella visualizzazione del grafico
- Series: serie del grafico
- Opzioni Manuali: opzioni che verranno scritte nel json per personalizzare il grafico secondo le impostazioni di HighCharts ed Highstock, per esempio

se vogliamo utilizzare un grafico che non è presente nei tipi dell'applicazione basta aggiungere a questa casella di testo l'opzione:

```
chart.type = "Spline";
```

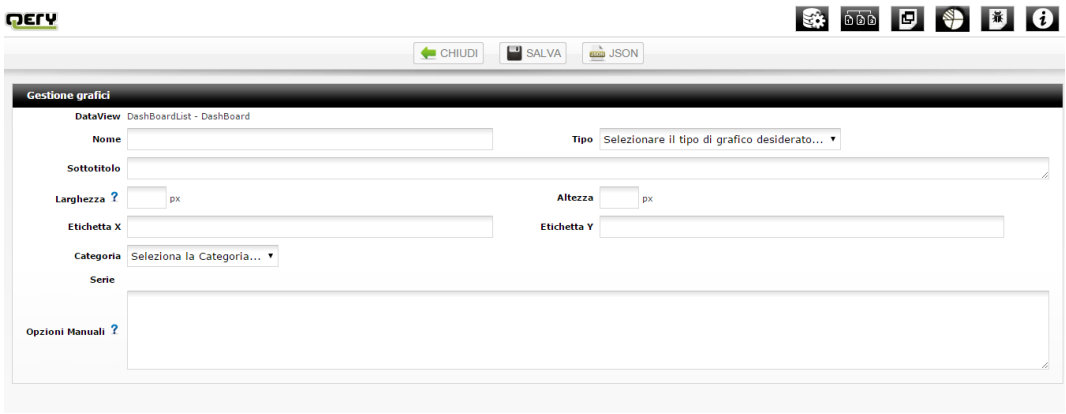

Figura 1.9: Pannello di Gestione Grafici

Con l'opzione Mappe si accede alla loro lista, cliccando sull'azione di creazione o di modifica si entra nel pannello di creazione/modifica delle mappe (*figura 1.10*). Qui è possibile effettuare varie operazioni per impostare le preferenze della mia mappa:

- Nome: breve descrizione della mappa
- Descrizione Mappa: descrizione del contenuto della mappa
- Visualizzazione Mappa: Tipo di mappa che vogliamo utilizzare:
  - RoadMap
  - Satellite
  - Terrain
  - Hybrid
- Marker Personalizzato: sostituzione del marker standard di Google con un marker a piacere caricandolo dal proprio pc
- Latitudine: coordinata geografica di Google maps che corrisponde alla coordinata dell'asse y

- Longitudine: coordinata geografica di Google maps che corrisponde alla coordinata dell'asse x
- Indirizzo: corrisponde alla Latitudine
- Titolo: Titolo della mappa
- Descrizione Marker: breve descrizione del luogo (nella mappa compare nel fumetto affianco al marker)

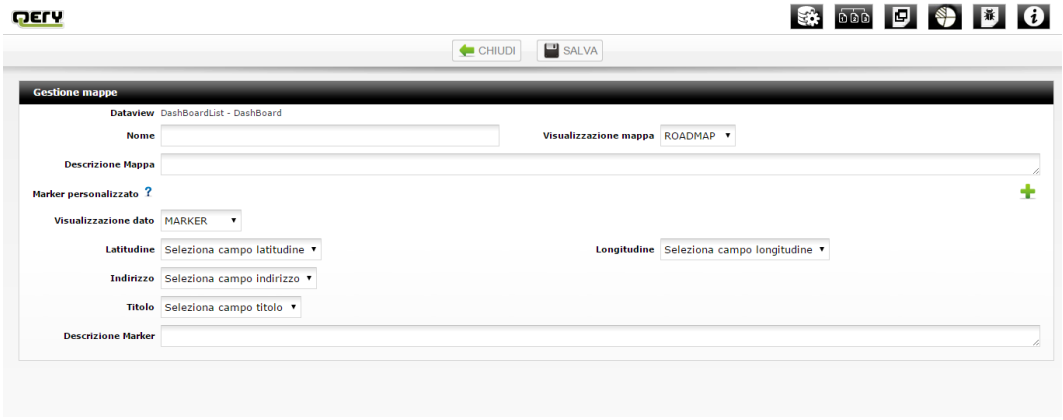

Figura 1.10: Gestione Mappe

Infine i Custom sono degli strumenti utilizzati per creare template HTML.
Si possono creare oggetti completamente personalizzati utilizzando codice HTML ed espressioni freemaker, mediante queste si possono estrapolare i dati dai risultati prodotti dal Dataview.
Per ogni custom è possibile scegliere una cover per poi applicarla al custom e creare così un widget.

1.1.3 Altre funzionalità di Qery Admin

Esistono altre funzionalità legate al qery Admin, tra cui la gestione delle lingue, degli import ed export e la duplicazione di Dataview.
Grazie alla funzione di gestione lingue è possibile modificare i nomi relativi ai Dataview mediante un semplice pannello. In questo pannello sono visualizzate sulla sinistra le label (che abbiamo utilizzato nel nostro Dataview) e alla destra più spazi in cui introdurre le traduzioni delle label nelle diverse lingue.

Le funzioni di import ed export sono definite sia sui progetti che sui Dataview.
La funzione di export permette di salvare un progetto o un Dataview in locale nel formato zip, al suo interno è presente un documento xml contenente tutti i dati del progetto o del Dataview scaricato.
Dopo aver esportato un Dataview o un progetto in un ambiente è possibile importalo in un altro ambiente mediante la funzione import. Questa richiede all'utente di immettere il file da importare ed automaticamente ricostruisce il Dataview o il progetto definito nel xml selezionato.

La funzione di duplicazione invece è una funzione propria del Dataview, la quale serve per copiare un Dataview esistente. Nel momento in cui tale opzione viene selezionata l'applicazione crea una copia di tutto il Dataview e dei suoi figli. Viene poi aperto il pannello di modifica del nuovo Dataview già popolato con i dati del precedente, che si è voluto duplicare.
In tale modo si velocizza la creazione di Dataview simili, in quanto l'unica cosa necessaria da modificare è l'id del Dataview che deve essere unico per evitare di sovrascrivere il Dataview precedente.

1.2 Struttura dell'applicazione: Viewer e Portale Aziendale

Il Viewer si occupa di visualizzare l'anteprima degli oggetti creati con l'Admin: Dataview, Grafici, Mappe e Custom.
È possibile accedere all'esecuzione/visualizzazione di tali oggetti mediante l'opzione play, presente accanto ad ognuno di essi nei pannelli di amministrazione dell'admin.
Un esempio di visualizzazione dei Dataview è presente nella *figura 1.11*.
Nella schermata si nota la presenza di due filtri uno di default e uno di tipo dropdownlist, sono anche presenti delle azioni globali di export di file excel, pdf e csv.

Inoltre è presente un paginatore, che raccoglie i record per pagina, in modo da rendere più veloce il caricamento dei risultati.
Grazie alla funzione “visualizza query eseguita”, in alto a sinistra, è possibile visualizzare la query eseguita dal Dataview per recuperare il result set ed il tempo di esecuzione di quest’ultima.

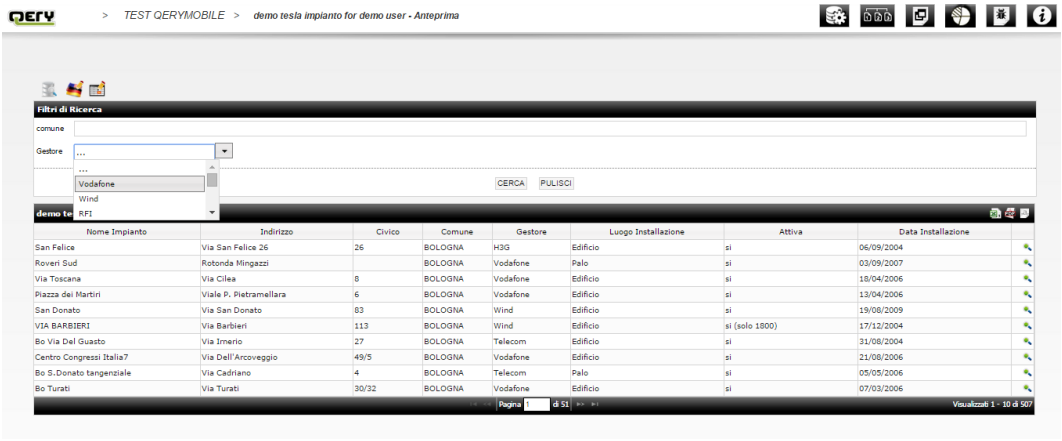

Figura 1.10: Esecuzione di un Dataview

A tale Dataview è legata una mappa, la Google maps presente *in figura 1.12.*

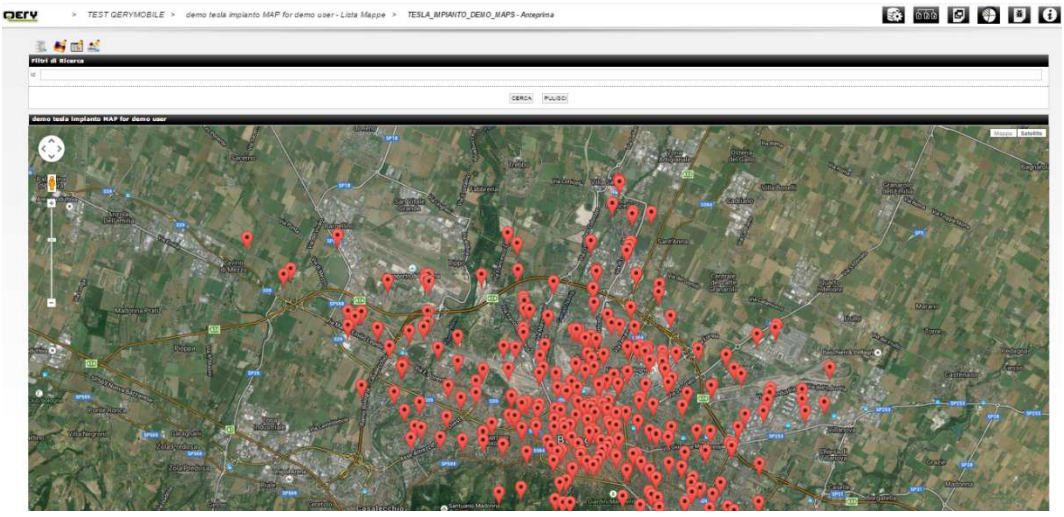

Figura 1.11: Visualizzazione di una mappa

Mediante l’azione locale presente in *figura 1.11,* contraddistinta da una lente di ingrandimento, è possibile visualizzare sulla mappa la posizione relativa al civico selezionato nel Dataview.

Un Esempio di Report Grafico è esposto in *figura 1.13*: nel caso che si vogliano visualizzare le statistiche del fatturato aziendale, si può creare un grafico a torta che visualizza l'andamento annuale del fatturato dell'azienda per agente.
Anche in questo caso cliccando sul pulsante azione, l'informazione riguardante gli agenti verrà passata al grafico che sposterà leggermente la fetta di torta evidenziando l'agente selezionato.

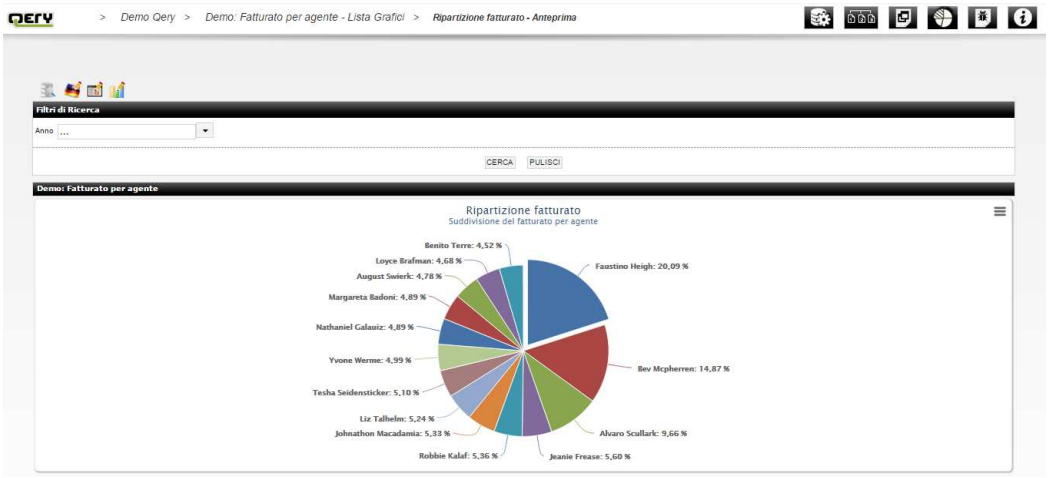

Figura 1.12: Visualizzazione di un Grafico

1.3 Database

Passiamo ora alla definizione della struttura della Base di Dati dell'applicazione Qery.
Inizieremo la trattazione descrivendo la struttura del database mediante uno schema E.R. passeremo poi alla definizione degli attributi mediante progetto logico.

1.3.1 Schema E/R

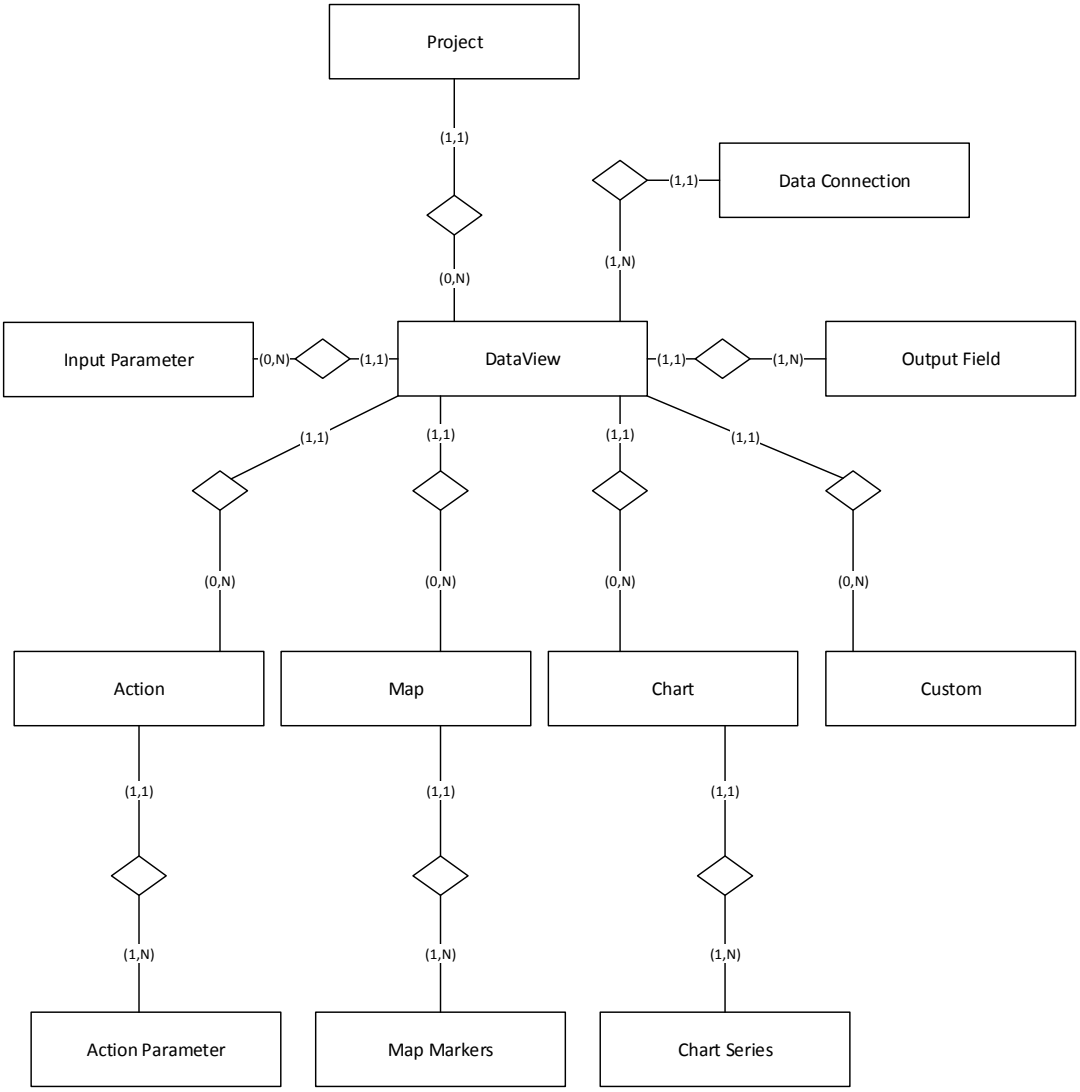

Figura 1.13: Schema E.R.

Come possiamo vedere dal diagramma E.R. l'entità principale dell'applicazione è il Dataview, a cui sono collegate tutte le altre entità, per migliorare la comprensione del seguente schema si è deciso di presentarlo senza gli attributi.
Dallo schema in *figura 1.14* è possibile osservare che l'oggetto Dataview è contenuto all'interno dell'oggetto Project e che ogni progetto può contenere più Dataview.
Allo stesso Modo anche gli oggetti Action, Map, Chart, Custom, Input Parameter ed OutputField sono contenuti all'interno Dataview il quale può essere collegato a più istanze di tali oggetti.
Le mappe, i grafici e le azioni hanno a sua volta dei figli che sono gli action parameters, i chart series e i mapmarkers i quali possono riferirsi ad una sola delle entità padre.
Infine possiamo notare un aspetto molto importante dell'applicazione che riguarda le connessioni al database, per cui una connessione dati può essere legata ad un solo Dataview mentre un Dataview può essere definito su una sola connessione.

1.3.2 Progetto Logico

In tale paragrafo viene riportato lo schema logico del database in cui sono riportai tutti gli attributi delle entità del database comprese le chiavi primarie e le chiavi esterne.
Le chiavi primarie vengono sottolineate mentre le chiave esterne sono indicate mediante la parola chiave FK.

```
DataConnection(id,url,username,password,driver,type,createus
er,updateuser,createdate,updatedate)

Project (code, name, createuser, updateuser, createdate,
updatedate)

Dataview(id,name,description,javascript,query,type,recordper
page,inputcolsnumber,dataconnectionid,projectid,createuser,u
pdateuser,createdate,updatedate)
```

FK: projectId REFERENCE Project

FK: dataconnectionid REFERENCE DataConnection

InputParameter(dataViewId,name,label,type,renderer,required,defaultvalue,colspan,conditions,textcase,advanced,style,popupname,vli_descriprion,vli_code,vli_url,vli_dataviewid,sorting,hidden)

FK: dataViewId REFERENCE Dataview

OutputField(dataViewId,name,label,type,pattern,visible,roles,style,calculatefield,grouping,sortable,defaultsort, sorttype,conditions,sorting)

FK: dataViewId REFERENCE Dataview

Action(id,name,dataViewId,icon,type,url,context,popup,refreshonclose,style,popupname,dynamiccontext,displaycondition,sorting)

FK: dataViewId REFERENCE Dataview

ActionParameter(actionid,name,fieldname,fieldtype,fieldgrouping)

FK: actionid REFERENCE Actione

Chart(id,name,description,type,width,height,category,xaxislabel,yaxislabel,dataViewId,sorting,categorydynamic,customoptions,manualoptions)

FK: dataViewId REFERENCE Dataview

ChartSeries (chartid, series, rgbcolor, labelvisible, sorting, type, label)

FK: chartid REFERENCE Chart

Map(id,name,description,render,dataViewId,markertype,longitude,latitude,address,title,markerdescription,sorting)

FK: dataViewId REFERENCE Dataview

```
MapMarkers(mapid,conditions,markericon,sorting)
     FK: mapid REFERENCE Map


Custom (id, name,
description,template,customtype,customcover,dataViewId,
sorting)
     FK: dataViewId REFERENCE DataView
```

1.3.3 Descrizione delle Tabelle

Segue una descrizione delle entità presenti nel database e dei loro attributi.

- DataConnection: Tabella che memorizza i dati relativi alle connessioni utilizzate da Dataview per connettersi ai vari database definiti
  - Id : identificativo univoco della connessione dati
  - url: url del database
  - username: nome utente database
  - password: password dell'utente del database
  - driver: driver necessario per la connessione al database
  - type: tipo della connessione (DataSource o DataConnection)
  - createuser: utente di creazione
  - updateuser: utente di ultima modifica
  - createdate: data di creazione
  - updatedate: data di ultima modifica

- Project : Tabella che rappresenta il contenitore dei DataView
  - Code: identificativo univoco del progetto
  - Name: nome del progetto
  - createuser: utente di creazione
  - updateuser: utente di ultima modifica
  - createdate: data di creazione

- updatedate: data di ultima modifica

❖ DataView: Tabella che memorizza le informazioni principali dell'oggetto Dataview e i legami presenti con le altre entità.
- Id: identificativo univoco del Dataview
- Name: descrizione breve del Dataview
- Description: descrizione del Dataview
- Javascript: scrittura di un pezzo di codice javascript che può essere utilizzato in pagina
- Query: query eseguita nel momento in cui viene eseguito il Dataview
- Type: tipo di Dataview (standard o di sistema)
- Recordperpage: numero di record che il Dataview deve visualizzare per ogni pagina
- Inputcolsnumber: dimensione delle colonne della tabella
- Dataconnectionid: id della connessione dati utilizzata dal Dataview
- Projectid: identificativo del progetto in cui è contenuto il Dataview
- createuser: utente di creazione
- updateuser: utente di ultima modifica
- createdate: data di creazione
- updatedate: data di ultima modifica

❖ InputParameter: Tabella contenente i dati relativi ai possibili filtri del DataView
- dataViewId: identificativo del Dataview
- name: descrizione breve dei parametri di input
- label: Etichetta del campo di input
- type: tipo del campo di input (Stringa, intero, decimale,…)
- required: indica se il campo di input è obbligatorio oppure no
- defaultvalue: valore di default del filtro
- colspan: spazio che deve contenere il filtro in pagina

- conditions: condizioni per la visualizzazione del filtro
- textcase: tipo del testo dei campi di input (upper o lowercase)
- advanced: il filtro deve comparire nella ricerca avanzata
- Style: stile del parametro
- popupname: nome della popup che deve visualizzare i filtri
- vli_description: descrizione della tendina o popup contente i campi di input
- vli_code: codice del campo di input a cui corrisponde la tendina
- vli_url: url utilizzato per aprire in popup filtri abilitati in link esterni all'applicazione
- vli_dataviewid: id del Dataview che popola la tendina
- hidden: visibilità del parametro di input

❖ OutputField: contiene le informazioni legate ai campi di output di ogni DataView

- dataViewId: identificativo univoco
- name: nome colonna della tabella su database
- label: Etichetta del campo di output
- type: tipo del parametro di output
- pattern: formato del campo di output
- visible: visibilità del parametro di output
- style: stile del parametro di output
- calculatefield: somma della colonna (visualizza una riga contenente la somma dei valori della colonna in fondo alla tabella)
- grouping: raggruppamento
- sortable: ordinamento
- defaultsort: ordinamento di default del result type
- sorttype: tipo ordinamento (ASC/DESC)
- conditions: condizioni di visibilità del campo di output

- Action: Tabella contenente le informazioni base delle azioni
  - Id: identificativo univoco dell'azione
  - Name: descrizione breve dell'azione
  - dataViewId: identificativo del Dataview a cui è legata l'azione
  - icon: icona dell'azione
  - type: tipo dell'azione (ActionUrl,ActionCommunication,…)
  - url: url nel caso il tipo dell'azione sia ActionUrl
  - context: contesto
  - popup: indica se l'azione è visualizzata in popup
  - refreshonclose: indica se effettuare il refresh dopo che è stata eseguita l'azione
  - style: stile
  - popupname: nome della popup se esistente
  - dynamic context: contesto dinamico
  - displaycondition: condizioni di visualizzazione dell'azione

- ActionParameters: Tabella contenente i parametri di input e di output delle possibili azioni presenti nel Dataview
  - Actionid: identificativo dell'azione
  - Name: descrizione breve dei parametri
  - Fieldname: nome dei campi o parametri dell'azione
  - Fieldtype: tipo dei campi o parametri dell'azione
  - Fieldgrouping: raggruppamento dei campi dell'azione

- Map: Tabella con le informazioni base delle mappe
  - Id: identificativo univoco della mappa
  - Name: descrizione breve della mappa
  - Description: descrizione della mappa
  - dataViewId: identificativo univoco del Dataview a cui è legata la mappa

- markertype: tipo del marker
- longitude: longitudine
- latitude: latitudine
- address: indirizzo
- title: titolo del marker
- markerdescription: descrizione del marker

❖ MapMarkers: Tabella che contiene i marcatori delle possibili mappe
- Mapid: identificativo univoco della mappa
- Conditions: condizioni di visibilità del marker
- Markericon: icona del marker

❖ Chart: Tabella con le informazioni base dei grafici
- Id: identificativo univoco del grafico
- Name: descrizione breve del grafico
- Description: descrizione del grafico
- Type: tipo del grafico (Torta, Linee, Istogramma,…)
- Width: larghezza del grafico
- Height: altezza del grafico
- Category: categoria
- Xaxislabel: etichetta dell'asse x
- Yaxislabel: etichetta dell'asse y
- dataViewId: identificativo univoco del Dataview
- categorydynamic: categoria dinamica
- manualoptions: opzioni manuali

❖ ChartSeries: Tabella contenente le serie dei grafici (se presenti)
- Chartid: identificativo univoco del grafico
- Series: serie del grafico
- Rgbcolor: colori del grafico

- Labelvisible: visibilità etichetta del grafico
- Type: tipo della serie
- Label: etichetta della serie

❖ Custom: Tabella con le informazioni principali degli oggetti custom

- Id: identificativo univoco del custom
- Name: descrizione breve
- Description: descrizione del custom
- Template: template HTML del custom
- Customtype: tipo del custom
- Customcover: cover/interfaccia del custom
- dataViewId: identificativo del Dataview a cui il custom è collegato

Capitolo 2

Tecnologie Lato Server

2. Introduzione alle Tecnologie dell'applicazione

Nei prossimi due capitoli andremo a presentare le tecnologie utilizzate per lo sviluppo del progetto.
Innanzitutto il progetto utilizza principalmente il linguaggio di programmazione Java ed è stato sviluppato mediante l'ambiente di sviluppo integrato Eclipse Kepler 4.3.
In questi due capitoli si parlerà in modo specifico di alcuni Framework e Linguaggi di Programmazione utilizzati nello sviluppo della nostra applicazione Web.
Un Framework può essere definito come un pezzo della struttura del software che si occupa di definire una piattaforma composta da funzioni comuni, con cui le applicazioni che utilizzano tale strumento possono essere costruite in modo più semplice e rapido.
In sostanza il Framework è una struttura software che provvede soluzioni a problemi generali che possono essere facilmente ereditate da applicazioni costruite su tale architettura.
I Framework lato Server di cui parleremo sono: il Framework Quix, Struts2 e Spring.
Successivamente, nel capitolo 3, passeremo a parlare degli strumenti lato Client utilizzati nell'applicazione web come JavaScript: linguaggio di programmazione nato per integrare il linguaggio di markup HTML e jQuery nato per semplificare le funzionalità di JavaScript.
Infine analizzeremo un nuovo framework basato sul linguaggio JavaScript, che è stato utilizzato per la prima volta nell'applicazione: AngularJS.

2.1 Struts2

Introduciamo Struts2[1] partendo da una breve panoramica sull'ambiente in cui esso lavora, ossia sulle tecnologie al quale esso si appoggia.
In seguito parleremo della struttura di Struts2 basata sul modello MVC, ed infine studieremo l'applicazione di Struts2 nel caso di elaborazione di una richiesta.

2.1.1 Breve panoramica su HTTP e Servlet

In questo paragrafo accenneremo le principali componenti necessarie al funzionamento di un applicazione web.

Innanzitutto studiamo la posizione di Struts2 all'interno dello stack tecnologico di un applicazione web.

Come possiamo notare nella figura 1.1, Struts2 si trova sopra a due importanti tecnologie: HTTP e Servlet.

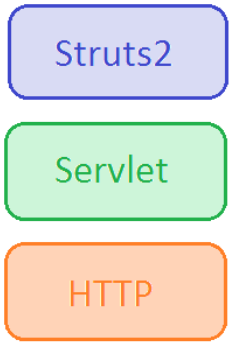

Figura 2.1: Stack of Web Application

Tutte le applicazioni Struts2 utilizzano il protocollo HTTP per l'interconnessione tra client e server e L'API Servlet[2] di Java, che espone tali comunicazioni HTTP al linguaggio Java.

Analizziamo lo stack dall'alto verso il basso, iniziamo perciò a parlare del protocollo HTTP (HyperText Transfer Protocol).

Il protocollo HTTP, in un'architettura tipica client-server, viene utilizzato come principale sistema per la trasmissione di informazioni sul web, questo funziona con un tipico meccanismo request/response in cui il client esegue una richiesta ed il server restituisce una risposta.

In un applicazione web il browser è considerato il client mentre il server è il sito web.

La risorsa richiesta dal client al server può essere un documento HTML statico, oppure può essere un documento generato dinamicamente dal server.

HTTP differisce da altri protocolli di livello 7, per il fatto che le connessioni vengono chiuse nel momento che una o una serie di richieste correlate sono state soddisfatte. Tale comportamento, di non conservazione della sessione di navigazione, rende più efficiente la connessione sia lato client che lato server. Questo incremento di efficienza è dovuto alla diminuzione del numero di connessioni attive, ma allo stesso modo porta svantaggi agli sviluppatori in quanto per conservare lo stato dell'utente occorre utilizzare altri metodi (tipicamente basati sui cookie).

Un altro svantaggio di HTTP è quello di essere basato su scambi di file testuali. Questo crea una grande difficoltà nel tradurre le risposte di un'applicazione, da un linguaggio fortemente tipizzato come Java, in una forma in cui tutti i dati devono essere rappresentati come solo testo. Dal lato opposto anche i parametri di richiesta inviati in forma testuale dal client e consegnati al server, devono essere migrati in ambiente Java.

Vedremo come gestire tali problemi di comunicazione tra client e server nel paragrafo JSON.

Passiamo ora al secondo livello dello Stack: le Servlet Java.

Nello sviluppo dell'HTML presto è stato necessario invocare programmi sul server web, tali programmi sono stati chiamati Common Gateway Interface, si dice che quando un sito web include una CGI questo è nominato applicazione web.

Grazie all'URL si identifica il programma da eseguire, il programma legge la request, processa i dati e genera una risposta, tale risposta è inviata al webserver che la include nella response da restituire al client.

Siccome il tempo di risposta dei programmi CGI è molto alto, per migliorare tale tecnologia sono state ideate le Servlet.

Una Servlet è un programma scritto in java ed eseguito sul server, a differenza dei CGI le Servlet sono eseguite all'interno di un WebContainer ossia una JVM che fornisce l'implementazione delle API dei Servlet.

Le Servlet sono degli oggetti Java il cui scopo è quello di ricevere tutte le richieste, analizzarne i dati, elaborare le risposte e restituirle nella forma corretta dell'HTTP request,

Perciò in sostanza tali oggetti fanno da tramite tra il protocollo HTTP e il linguaggio java in modo da creare un'interfaccia che semplifichi il compito del programmatore.

2.1.2 Introduzione a Struts2 (Modello MVC)

Struts2 utilizza il pattern Model View Controller, tale pattern prevede una separazione dei compiti che ci permette di dividere la complessità di sistemi di grandi dimensioni dividendoli in componenti ad alto livello.

La progettazione MVC suddivide la logica del sistema in tre diversi concetti: Il model, il viewer e il controller.
Questi concetti in Struts2 sono implementati mediante l'Action, il Result e il FilterDispatcher.
Lo schema nella figura 2.2 raffigura tale struttura e in quale ordine questi oggetti vengono invocati

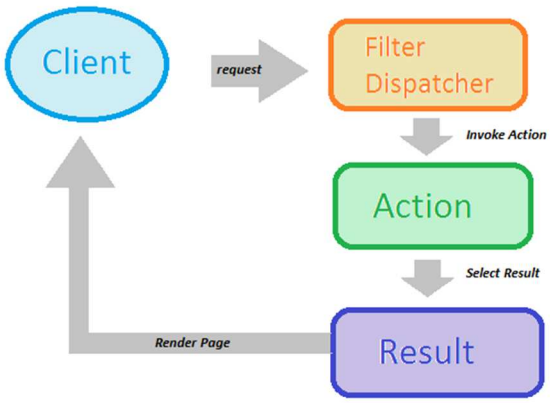

Figura 2.2: MVC Pattern in Struts2

- FilterDispatcher – Controller

  Il compito del Controller è quello di mappare le richieste del client alle Action, perciò tale componente riceve i comandi dell'utente e li attua modificando lo stato degli altri due componenti (Viewer e Model).

  Il ruolo del controllore è svolto dai Struts2 FilterDispatcher.

  Il FilterDispatcher è un oggetto che funge da filtro e che ispeziona ogni richiesta in arrivo per determinare quale Action deve gestire la richiesta.

  In tale modo il Framework gestisce tutto il lavoro e il programmatore ha il compito di implementare le azioni e di informare Struts2 su quali URL mappano tali azioni.

  La mappatura tra gli URL e le action di Struts2 avviene mediante file di configurazione basati su XML o Java.

```
<action name=”dashBoardView” class=”dashBoardViewAction”>
     <interceptor-ref name=”quixStack”>
          <param name=”quixValidationInterceptor.excludeMetods”>
            select
          </param>
     </interceptor-ref>
     <result> name=”qmHome”>/pages/dashboard/Home.jsp</result>
</action>
```

- Action – Model

  Il model è lo stato interno dell'applicazione che fornisce i metodi per accedere ai dati utili all'applicazione.

  Un'azione Struts2 ha due ruoli: l'incapsulamento delle chiamate e il trasferimento dati.

  Le Action perciò vengono definite come classi che fanno da ponte tra le azioni client-side dell'utente e le operazioni della logica applicativa. Questo livello nasconde lo strato fisico in cui vengono memorizzati i dati e si occupa di salvare, eliminare o restituire i dati che il client richiede.

- Result – View

  Tale componente ha il compito di visualizzare i dati contenuti nel model e di interagire con gli utenti dell'applicazione web.

  Il Result restituisce il risultato dell'elaborazione al browser, tale risultato solitamente è una pagina HTML ottenuta da una pagina JSP che contiene un interfaccia utente rappresentante lo stato interno dell'applicazione o uno stato di errore.

Uno dei grossi vantaggi di Struts2 è proprio dato da questa divisione delle componenti. Questo metodo permette di tenere separate le logiche mantenendo un ambiente pulito e sempre disponibile all'inserimento di nuove tecnologie in ogni sua parte o componente.

2.1.3 Ciclo di Vita di una Richiesta in Struts2

Per far sì che Struts2 crei un ambiente pulito di lavoro non è solo necessario seguire un modello MVC, ma anche implementare altri componenti architetturali: Interceptor, Value Stack e OGNL.

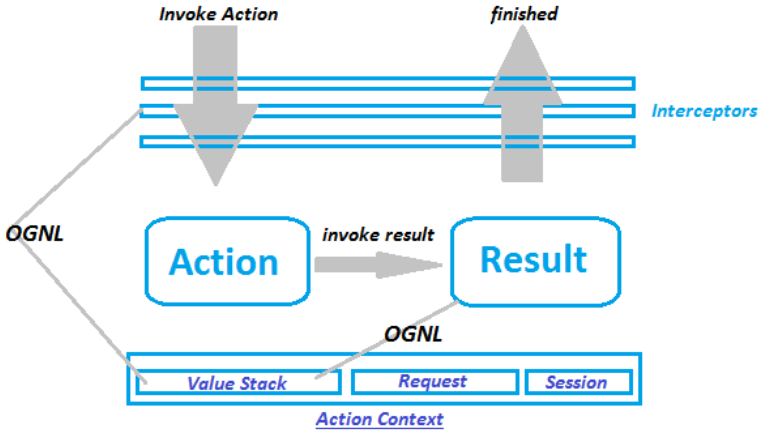

Figura 2.3: Elaborazione di una richiesta in Struts2

Innanzitutto descriveremo l'elaborazione di una richiesta in Struts2 poi, in seguito, passeremo alla spiegazione dettagliata dei nuovi componenti appena accennati.

Nella figura 2.3 vediamo come tutti i nostri componenti interagiscono fra loro nel flusso di elaborazione di una richiesta, possiamo suddividere e perciò analizzare tale flusso di lavoro in fasi:

a) L' Utente richiede una risorsa al server
b) Il FilterDispatcher esamina la richiesta e determina l'azione da invocare
c) Prima che l'azione sia realmente invocata vengono eseguite ulteriori funzionalità: gli intercettori
d) L'azione selezionata viene eseguita per l'operazione richiesta.
e) Anche dopo la che l'operazione è stata eseguita gli intercettori configurati sono applicati, se necessario, a qualsiasi post-elaborazione
f) Infine il risultato viene preparato dal Viewer per essere visualizzato dall'utente.

Gli Interceptor, sono classi stateless che non mantengono uno stato tra invocazioni successive, tali classi possono essere invocate automaticamente prima e dopo una Action.
Struts2 prevede un gruppo di interceptor, anche chiamato stack di default, questi vengono richiamati prima di invocare una qualsiasi Action.
I principali interceptor di default di Struts2 sono:

- Exception: gestisce la risposta da inviare al client quando l'azione non termina in maniera corretta;
- Prepare: per inizializzare le Action;
- I18n: memorizza il locale dell'utente corrente;
- Debugging: attiva funzioni di debug;
- FileUpload: gestisce l'upload dei file;
- Validation: esegue la validazione dei dati forniti nel form

Gli intercettori perciò forniscono un'architettura in cui è possibile definire funzioni trasversali al flusso di lavoro che possano essere riutilizzabili, in quanto indipendenti dalle altre funzioni.

L'ActionContext contiene tutti i dati che costituiscono il contesto in cui un'azione si verifica. Questo include il ValueStack oppure oggetti interni come la richiesta, o la sessione.

Il ValueStack è un'area di memoria che contiene tutti i dati associati all'elaborazione di una richiesta.

I dati nel ValueStack seguono la richiesta attraverso tutte le fasi, infatti questi sono spostati nel ValueStack in fase di preparazione per l'elaborazione della richiesta, manipolati durante l'esecuzione dell'Action e letti nel momento in cui i risultati vengono restituiti alle pagine di risposta.

OGNL è lo strumento che ci permette di accedere ai dati contenuti nel nostro repository centrale: il ValueStack.

Più precisamente OGNL sta per "*Object Graph Navigation Language*" e si tratta di un linguaggio di espressioni che consente di fare riferire e manipolare i dati sul repository.

OGNL offre tag particolari che risolvono molti dei più comuni compiti delle pagine JSP, in quanto tengono la corrispondenza tra le Action e il Result.
Per esempio i tag di tipo:

```
<s:iterator value="#myList">
       //do something
</s:iterator>
```

Eseguono un ciclo riferendosi alla variabile myList definita nell'action legata alla corrispondente pagina JSP.

2.2 Spring

Spring[3] è stato creato per affrontare la complessità dello sviluppo di applicazioni enterprise.

Il Framework Spring vanta di un architettura estremamente modulare, per tale motivo l'adozione di questo framework in un progetto è molto semplice e può avvenire in maniera incrementale. Questa sua caratteristica permette, inoltre, una facile integrazione con altri framework esistenti, come ad esempio Struts2.

Spring mette a disposizione una serie completa di strumenti atti a gestire l'intera complessità di un progetto software.

Tale framework fornisce un approccio semplificato alla maggior parte dei problemi ricorrenti nello sviluppo software come l'accesso al database, la gestione delle dipendenze e il testing.

2.2.1 Caratteristiche Principali di Spring

Di seguito elencheremo i vantaggi e le caratteristiche fondamentali di Spring:

- *Lightweight*: Spring è leggero in termini di dimensioni, l'intero framework può essere distribuito in un unico file JAR che pesa poco più di 1 MB.
- *Ioc (Inversion of Control)*: il container crea le dipendenze degli oggetti e le rende disponibili prima che queste vengano richieste.
- *AOP (Aspect Oriented):* consente uno sviluppo coerente separando la logica applicativa dai servizi di sistema, tale caratteristiche porta vantaggi nelle operazioni trasversali tra più oggetti.
- *Container*: Spring è un contenitore in quanto contiene e gestisce il ciclo di vita e la configurazione degli oggetti dell'applicazione.
- *Framework*: partendo da componenti semplici rende possibile la configurazione e la composizione di applicazioni complesse.

Tali aspetti di Spring consentono di scrivere codice che più pulito, più semplice e facile da testare.

2.2.2 Architettura di Spring

Un altro vantaggio offerto da Spring è quello di poter escludere le parti del framework non necessarie all'applicazione che si sta sviluppando, includendo esclusivamente quelle utili.

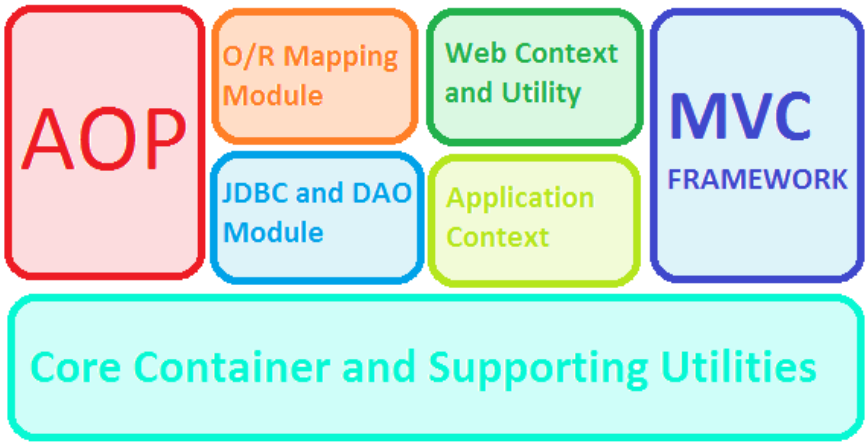

Figura 2.4: Architettura di Spring

Questo è possibile grazie all'architettura modulare di Spring che è composta da sette livelli ben definiti.

Partendo dal livello più basso dell'architettura, rappresentata in *figura 2.4*, analizziamo il Core Container, tale livello rappresenta la parte principale di Spring e sopra di esso è costruito l'intero framework.

In tale modulo troviamo lo Spring's BeanFactory: la parte principale di ogni applicazione Spring-Based.

Sostanzialmente tale componente è definita come un pattern di tipo factory che applica Ioc per separare le configurazioni dell'applicazione e le specifiche delle dipendenze dal codice dell'applicazione stessa.

Il core container, grazie a tale factory, definisce come vengono creati, configurati e gestiti i beans di Spring, tali beans possono essere oggetti di qualsiasi natura.

Proprio per tale motivo questo livello è chiamato Container, in quanto “contiene” tutti i beans dell’applicazione.

I beans vengono dichiararti nel file XML di Spring in tale modo:

```
<bean id=”dashBoardValidator”
class=”it.quix.dataview.validation.DashBoardValidator” />
```

All’interno del Core container si trova anche il modulo Expression Language, il quale fornisce un potente linguaggio per interrogare e modificare gli oggetti a runtime.

Risalendo l’architettura incontriamo il modulo Application Context.

Il modulo context estende i servizi basilari del Core aggiungendo le funzionalità tipiche di un moderno framework come:

- supporto per la risoluzione di messaggi di testo
- metodi generici per caricare le risorse come file ed immagini
- pubblicazione di eventi riguardanti i beans registrati come listener
- metodi per effettuare le validazioni.

Passiamo ora al modulo AOP.

Per capire il motivo della nascita dell’AOP ricordiamo uno dei principi fondamentali della programmazione ad oggetti: la modellazione del programma come uno scambio di messaggi tra oggetti.

Tale principio comporta che gli oggetti siano entità tra loro indipendenti e garantisce la modularità del sistema software sviluppato con un linguaggio object oriented.

Allo stesso modo questo principio rende difficile l'implementazione di alcune funzionalità che per loro natura sono comuni a più oggetti; infatti esistono alcuni

compiti del sistema informativo che non possono essere modellati come oggetti, in quanto non interessano l'applicazione nel suo insieme.

L'Aspect Oriented Programming (AOP) è nato con lo scopo di risolvere tali problemi, gli aspetti modellano le problematiche trasversali agli oggetti stessi, ossia compiti che nell'OOP tradizionale sono difficilmente modellabili.

La funzionalità di Programmazione AOP offre perciò un nuovo modo di programmare che porta vantaggi in tutte quelle operazioni che sono trasversali tra più oggetti. In Spring l'utilizzo di AOP avviene nella gestione delle transazioni.

Per il Data Access Integration Spring offre i moduli DAO e ORM.

Grazie al modulo JDBC and DAO si crea del codice semplice e pulito, in quanto solitamente il codice scritto per aprire, chiudere o gestire una connessione JDBC è lungo e ripetitivo, grazie al DAO (Data Access Object) di Spring tale codice viene astratto. Inoltre tale modulo gestisce gli errori di connessione ed interrogazione al database con una serie di eccezioni significative per le diverse piattaforme esistenti.

L' Object Relational Mapping (ORM) è una tecnica di programmazione che favorisce l'integrazione di sistemi software, aderenti al paradigma della programmazione orientata agli oggetti con sistemi RDBMS.
Un prodotto ORM fornisce, mediante un'interfaccia orientata agli oggetti, tutti i servizi inerenti alla persistenza dei dati, astraendo nel contempo le caratteristiche implementative dello specifico RDBMS utilizzato.
Il modulo ORM fornito da Spring non implementa una propria soluzione ORM ma fornisce degli hooks ai framework ORM più popolari, come ad esempio Hibernate.
Il WebContext è formato dai moduli web, web-servlet, web-struts e web-portlet.
Il modulo web mette a disposizione delle funzioni basilari per la creazione di applicazioni web, mentre il modulo web-servlet implementa il pattern MVC realizzando lo Spring MVC Framework.
Il framework MVC offre una netta separazione tra il codice dell'applicazione e i moduli web, integrandosi con tutte le altre caratteristiche del Framework Spring.

Tale framework MVC è ampiamente configurabile attraverso delle Strategy, può operare sia in contesti servlet che portlet e può essere utilizzato con numerose tecnologie di View comprese le pagine JSP

Il modulo Web-Struts contiene le classi di support per l'integrazione di Struts2 all'interno di una Spring Application.

Web-Portlet è il modulo che fornisce l'implementazione MVC, grazie alla quale il framework Spring può essere utilizzato in un ambiente portlet, tale livello rispecchia perciò la funzionalità del modulo web-servlet.

2.3 Framework Quix

Il Framework[4] di Quix è una raccolta di librerie che contiene tutte le classi sviluppate nel tempo dall'azienda per creare applicazioni basate su Struts2, Spring e Maven (tool per gestire il ciclo di vita di un progetto(a)) in minore tempo.

Oltre a tali classi, il Framework contiene:

- Un generatore di codice: ha come scopo la generazione di tutto il codice ripetitivo, permettendo così di accelerare lo sviluppo del software in fase iniziale.
- Moduli di utilità:
    - SysAttribute: I SysAttribute assolvono al compito di raccogliere e organizzare elenchi di dati, in tale modo si ha un'estrema flessibilità e rapidità di creazione, gestione e manutenzione di un elenco a differenza della solita creazione di una anagrafica ad hoc per ogni elenco.
    - SysConfig: le SysConfig sono utilizzate per racchiudere tutte le configurazione dell'applicazione come ad esempio quelle espresse all'interno di un file di Spring.
- Una serie di best practice per sviluppare task comuni
- Una struttura standard con cui organizzare i progetti

Soffermiamoci sulla struttura organizzativa del Framework.

(a) le funzioni di Maven non vengono descritte nei particolari in quanto non sono considerate fondamentali per l'oggetto della trattazione

In *figura 2.5* viene presentata la struttura interna di un'applicazione in seguito all'importazione di un progetto generato.

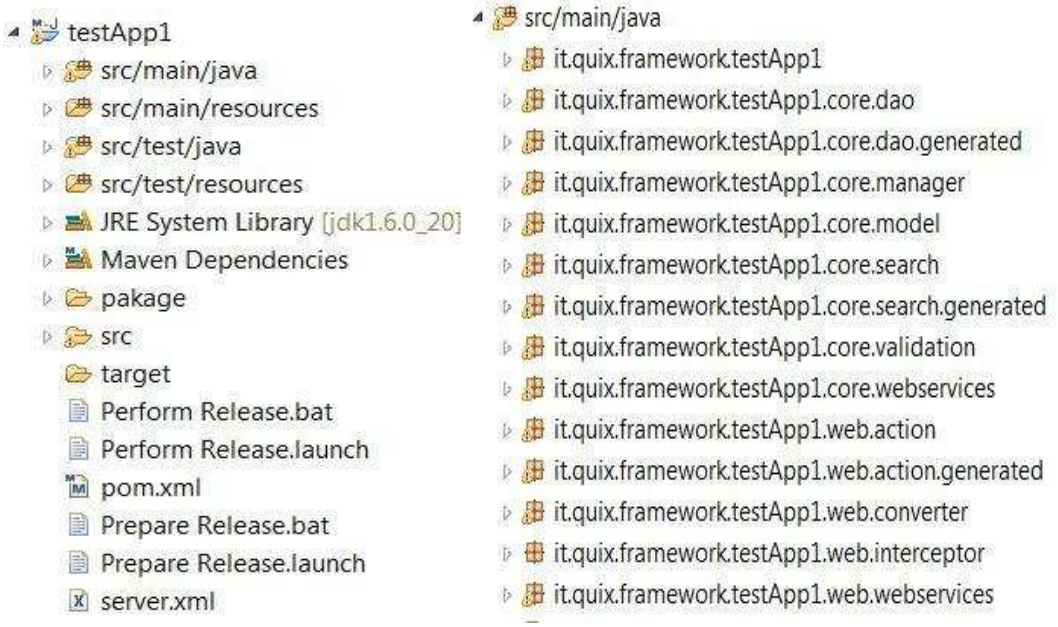

Figura 2.5: Framework – Struttura di un Progetto generato

Da tale struttura possiamo notare che nel codice che viene generato partendo dall'applicazione base, esiste un solo modulo, tale modulo contiene più package.
Alcuni package si contraddistinguono per avere la parola finale comune generated, essi contengono le classi che vengono create dal generatore di codice e che potrebbero essere rigenerate una o più volte durante la vita dell'applicazione.
Per esempio uno di questi package è il package DAO, che analizzeremo tra poco, contenuto nella parte core dell'applicazione.
La nomenclatura dei package generati raffigurati in *figura 2.1* conserva la distinzione tra "core" e "web", questo viene fatto per distinguere le classi usate dalle due componenti.
Nella parte "core" vengono generati, tra gli altri, i pacchetti: DAO, manager, model, search e validation.
Mentre nella parte "web" vengono generati i pacchetti action, converter, interceptor e webservices.

Andiamo ora ad analizzare, tra questi, i pacchetti del progetto generato utili per la trattazione:

- Model

  I model sono le classi degli oggetti principali che vengono utilizzati all'interno dell'applicazione, solitamente questi corrispondo alle tabelle che compongono il database dell'applicazione.

  Tali classi perciò contengono come variabili private gli attributi che le corrispondenti tabelle hanno su database.

  Ogni variabile può essere acceduta mediante i metodi get e set.

  Oltre a tali metodi, solitamente in tali classi, sono anche presenti funzioni di comparazione tra oggetti dello stesso tipo ed altri metodi che può essere utile richiamare su questi determinati oggetti dall'applicazione.

  Ogni Model contiene delle notazioni che servono per legare la struttura dell'applicazione con quella del database e per effettuare la generazione delle classi DAO.

- DAO

  I DAO sono le classi che si occupano di effettuare tutte le interrogazioni SQL necessarie per il corretto funzionamento dell'applicazione.

  Il codegenerator del Framework si occupa di generare gli AbstractDAO nel package dao.generated e le classi DAO nel package DAO.

  La differenza tra i due tipi di classi, sostanzialmente si basa sul fatto che gli abstarctDAO contengono tutti quei metodi di utilità generali che possono essere utilizzati spesso nelle applicazioni per interrogare il database (come ad esempio l'inserimento o l'eliminazione di un record).

  Mentre nelle classi di tipi DAO vengono inseriti dall'utente le interrogazioni personalizzate e non standard di cui l'applicazione necessita.

  Questi due tipi distinti vengono creati in tale modo in quanto nel momento in cui l'utente ha bisogno di un nuovo metodo per interrogare il database, questo lo inserirà nella classe DAO. La classe abstractDAO rimane così

invariata dalla data di ultima generazione, in questo modo se i DAO dovessero essere rigenerati non si perderebbero metodi utili implementati in seguito dall'utente.
Una classe di tipo AbstractDao potrebbe essere rigenerata nel caso che la struttura di una tabella del database cambi, come per esempio se venisse aggiunta o cancellata una colonna.

- Manager
  Le classi di tipo Manager sono le classi che fungono da tramite tra le Action e i DAO.
  Le Action implementano un Manager (il quale è un oggetto unico per tutta l'applicazione e acceduto mediante notazione @Resource) e mediante la notazione puntata richiamano i metodi del manager i quali a sua volta richiamano i relativi metodi del DAO.
  È presente un manager unico per tutta l'applicazione.

- Validation
  Le classi di validazione vengono richiamate nel momento in cui viene richiesto di salvare i dati su database.
  Prima che il salvataggio avvenga perciò vengono richiamati dei controlli sui dati, tali controlli vengono effettuati mediante i metodi implementati in queste classi di validazione.
  È presente una classe Validation per ogni Model esistente.

- Action
  Le Action sono le classi che si interfacciano con la parte web, in quanto tale classe fa da ponte tra le azioni client-side dell'utente e le operazioni della logica applicativa.

  Grazie a tali strumenti il Controller di Struts2 gestisce le attività e le funzioni dell'applicazione stessa.

La cartella **src/main/resources** (*figura 2.6*) contiene tutti i file di configurazione:

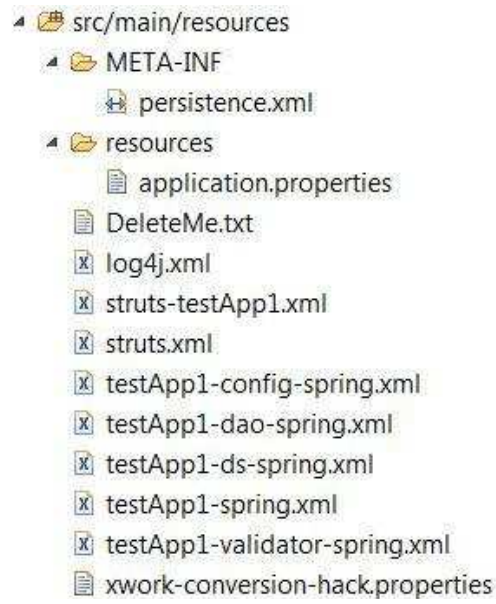

Figura 2.6: Framework – File di configurazione

Tra questi file troviamo:

- application.properties: contiene le traduzioni nella lingua di default delle label dell'applicazione.
- log4j.xml: contiene le configurazioni dei log per l'ambiente di sviluppo
- struts-qery.xml: Contiene le configurazioni delle Action specifiche per l'applicazione
- struts.xml: Contiene le configurazione di base di Struts2
- xxx-config-spring.xml: Contiene le configurazioni dell'applicazione che sono state impostate tramite Spring
- xxx-dao-spring.xml: Contiene la dichiarazione delle classi di accesso ai dati su database
- xxx-ds-spring.xml: Contiene le dichiarazione dei datasource utilizzati dal sistema
- xxx-spring.xml: Contiene le dichiarazione dei bean specifici per l'applicazione
- xxx-validator-spring.xml: Contiene la dichiarazione di tutti i validatori utilizzati dall'applicazione.

Infine troviamo la cartella main/webapp in *figura 2.7* che contiene tutto ciò che riguarda la presentazione dell'applicazione.

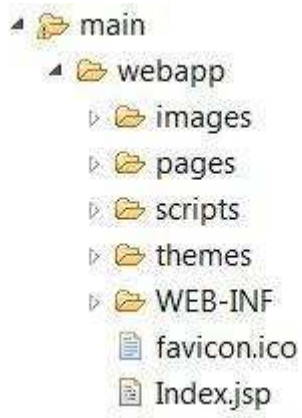

Figura 2.7: Framework – folder main/webapp

In tale sezione verranno perciò incluse sotto la sezione pages tutte le pagine JSP dell'applicazione.

Mentre nella sezione scripts saranno aggiunte le pagine JavaScript e anche le relative librerie JavaScript, come per esempio jQuery, jQuery UI ed AngularJS.

Ed infine nella cartella themes saranno aggiunti tutti i file CSS per personalizzare la grafica dell'applicazione ed eventuali file di Framework grafici come Bootstrap.

Capitolo 3

Tecnologie Lato Client

3.1 JavaScript

Introduciamo il linguaggio web Javascript per comprendere al meglio le successive tecnologie adottate nel progetto, come jQuery ed AngularJS.

3.1.1 Introduzione a JavaScript

JavaScript[5] è nato nel 1995 come un modo per aggiungere dei programmi alle pagine web.
JavaScript[6] è un linguaggio di Scripting orientato agli oggetti e agli eventi, viene utilizzato nella programmazione Web lato client per la creazione di siti e applicazioni web aventi effetti dinamici interattivi.

Tali effetti vengono creati mediante funzioni di script, invocate da eventi innescati sulla pagina web in uso mediante click del mouse o digitazione su tastiera.

Si è così passati dal classico sito web statico ad un sito web moderno, contenente un applicazione con cui è possibile interagire direttamente senza fare ricaricare la pagina per ogni azione richiesta all'utente.

JavaScript deriva la sua sintassi da Java ma per il resto non ha quasi nulla a che fare con tale linguaggio di programmazione. Il nome "JavaScript" è stato affidato a tale linguaggio web per motivi di marketing, in quanto nel momento in cui tale linguaggio veniva introdotto, Java aveva molto successo.

Tale linguaggio web è molto flessibile e lascia molta libertà al programmatore, questa flessibilità ha il vantaggio di rendere implementabili molte tecniche impossibili in linguaggi più rigidi. Il linguaggio di base di JavaScript definisce un API minima per lavorare con il testo, gli array, le date e le espressioni regolari, ma non include nessuna funzionalità di input o di output, queste sono a carico dell'ambiente all'interno del quale è inserito JavaScript (solitamente un browser web).

3.1.2 Caratteristiche di JavaScript

Le caratteristiche principali di JavaScript sono:

- Linguaggio interpretato: il codice non viene compilato, ma interpretato (in JavaScript lato client, l'interprete è incluso nel browser che si sta utilizzando).
- Sintassi simile ai linguaggi C, C++ e Java.
- Linguaggio di programmazione ad alto livello: in quanto contiene strutture di controllo, cicli e consente l'utilizzo del paradigma object oriented.
- Ha una tipizzazione debole: tutte le variabili definite in JavaScript hanno tipo comune var perciò il vero tipo della variabile non è esplicitato nel codice
- Linguaggio debolmente orientato agli oggetti: ha un meccanismo di ereditarietà diverso da quello del linguaggio Java.

In JavaScript lato client, il codice viene eseguito direttamente sul client e non sul server, in tale modo il web server non viene sovraccaricato a causa delle richieste dei client.

Ogni informazione che presuppone un accesso a dati memorizzati in un database remoto, deve essere rimandata ad un linguaggio che effettui esplicitamente la transazione. Quindi il cliente deve richiamare il server che effettuerà l'interrogazione su database, recupererà i dati e li ripasserà al client in una o più variabili JavaScript.

Tali operazioni richiederebbero obbligatoriamente il caricamento della pagina stessa. Con l'avvento di AJAX tutti questi limiti sono stati superati.

Asynchronous JavaScript and XML è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive. Si basa su uno scambio di dati in background fra web browser e server, consentendo l'aggiornamento dinamico di una pagina web senza un esplicito ricaricamento della pagina stessa.

Approfondiremo AJAX nel paragrafo riguardante jQuery.

3.1.3 Elementi del linguaggio JavaScript

- Variabili: come in tutti i linguaggi di programmazione le variabili possono essere locali o globali.
  Le variabili locali sono definite all'interno della funzione e il loro scope(visibilità) è solo all'interno della funzione stessa.
  Mentre le variabili globali sono definite all'esterno di qualunque funzione e hanno perciò visibilità nell'intera pagina web.
  Le variabili sono tipizzate dinamicamente, e hanno una tipizzazione debole in quanto queste possono essere definite semplicemente assegnando loro un valore oppure usando il comando var.

- Oggetti: JavaScript ha diversi generi di oggetti predefiniti:
  - Array
  - Boolean
  - Date
  - Function
  - Math
  - Number
  - Object
  - RegExp
  - String

  Mediante la definizione di un costruttore è possibile definire oggetti.

JavaScript è un linguaggio orientato a oggetti basato su prototipi, perciò l'eredità è fra oggetti e non fra classi.
In quanto JavaScript non possiede classi, gli oggetti ereditano le proprietà dai loro prototipi.
Oltre a permettere la definizione di oggetti, il JavaScript mette a disposizione tutta una serie di oggetti che possono essere utilizzati per i propri script come per esempio: Button, Checkbox, Link, Radio, ecc.

- Strutture Condizionali: Javascript contiene anche strutture di controllo come IF - SWITCH e strutture cicliche come WHILE - FOR.

- Funzioni: una funzione è un blocco di istruzioni che prende in ingresso una lista di argomenti e che restituisce un valore tramite l'istruzione return.

  Il numero degli argomenti passati quando si chiama una funzione non deve necessariamente essere uguale al numero degli argomenti della definizione della funzione stessa, inoltre le funzioni possono essere create ed assegnate come qualsiasi altro oggetto:

```
var myFunc1 = new Function("alert('Hello World')");
var myFunc2 = myFunc1;
myFunc2();
```

- Interazioni con l'utente: JavaScript può interagire con l'utente della pagina web mediante Alert dialog box (finestra di avviso), Confirm dialog box (finestra di dialogo), Prompt dialog box (finestra di richiesta input) e Barra di stato del browser.

- StdOutput: Mediante il comando document.write è possibile scrivere direttamente codice HTML all'interno della pagina web a cui si riferisce il JavaScript.

Per esempio, considerando che in pagina JSP è possibile includere codice JavaScript mediante i tag <script>, possiamo avere:

```
<script>
      document.write("Hello World!");
</script>
```

- Eventi: Gli elementi html possono essere la fonte di vari eventi che possono avviare un'azione.

  Un set di esempio relativi agli eventi sono:

  - onClick: attivato quando si clicca su un oggetto;
  - onMouseOver: attivato quando si passa con il puntatore del mouse su un oggetto
  - onKeyPress: attivato quando si digita qualcosa in una casella di testo
  - onChange: attivato quando un oggetto cambia di stato
  - onSelect: attivato quando si seleziona una voce in una tendina

  È possibile associare ad ognuno di questi eventi una funzione JavaScript, per esempio nel codice riportato sotto al click del bottone nominato “presse me” verrà invocata la funzione JavaScript MyJsFunction.

```
<button onclick=”MyJsfunction()”>
   press me
</button>
```

- Gestione degli errori: avviene mediante il costrutto try-catch-finally in modo simile al Java.

3.2 jQuery

Siccome ogni browser implementa uno specifico motore JavaScript con specifiche proprie, è spesso impossibile essere certi del funzionamento cross-browser(b) di uno script.
Per risolvere tali problemi sono nati framework in grado di garantire il funzionamento cross-browser degli script e di estendere e facilitare le funzioni native di JavaScript.
Uno dei primi framework a vedere la luce è stato Prototype che ha approcciato il problema aggiungendo funzionalità agli elementi che compongono la pagina.
Il problema dell'approccio di Prototype, è che estendendo gli oggetti globali di JavaScript si potrebbero creare incompatibilità con nuove versioni o addirittura conflitti con le funzioni dello sviluppatore o con altre librerie.

Il framework jQuery[7] tenta di risolvere tali problemi con l'intento di rendere il codice più sintetico e di limitare al minimo l'estensione degli oggetti globali per ottenere la massima compatibilità con altre librerie.
Da queste idee è nata tale libreria la quale offre un'ampia gamma di funzionalità, come la manipolazione degli stili CSS, degli elementi HTML, fino ad arrivare agli effetti grafici e ai metodi per le chiamate AJAX.
Tutto queste funzionalità vengono svolte senza toccare nessuno degli oggetti nativi JavaScript.
I vantaggi d'uso di jQuery sono:

- La possibilità di usare tale libreria in tutti progetti senza nessun rischio di incappare in incompatibilità nel codice.
- jQuery ha un semplice sistema di estensione che permette di aggiungere nuove funzionalità (plugin) oltre a quelle predefinite.
- La sua sintassi è sintetica ed efficiente**:** il motto di jQuery è *"Write less, do more"*[8]**.**

(b)Cross-browser: Siti Internet progettati per essere funzionali e risultare graficamente identici quando visualizzati dai diversi browser.

Tutte le funzionalità di jQuery[9] dipendono dall'oggetto/funzione $, il cui utilizzo più comune è quello di selettore CSS.
All'intero del selettore $() si può collocare qualsiasi oggetto utilizzato in un foglio di stile come i nomi e gli id dei tag HTML o la classe CSS.
Le selezioni possono anche essere filtrate, i filtri del motore di selezione si presentano con una sintassi molto simile agli pseudoselettori CSS.

Una volta che si è selezionato l'elemento ricercato mediante le svariate tecniche di jQuery, è possibile applicare sopra a tali elementi dei metodi predefiniti sempre dalla stessa libreria.
Tali metodi vengono utilizzati per manipolare un elemento, un attributo o raffinare ulteriormente la nostra selezione.
Proponiamo di seguito, per chiarezza, un elenco di possibili operazioni effettuate mediante i metodi jQuery:

- Inserire nuovi elementi
- Inserire elementi contigui e avvolgere elementi
- Inserimento dinamico e la manipolazione di elementi
- Sostituire, rimuovere, svuotare e duplicare elementi
- Altre fondamentali operazioni sul DOM(c)
- Manipolare i CSS

Un'altra tecnica offerta da jQuery è quella del Traversing.
Tale tecnica è formata da un insieme di metodi per attraversare dinamicamente e lavorare con gli altri elementi del DOM (solitamente quelli che non sono stati ancora selezionati).
Infine parliamo di Animazioni ed Eventi.

(c)DOM: Il Document Object Model è una forma di rappresentazione dei documenti strutturati come modello orientato agli oggetti.

Grazie a jQuery è possibile creare effetti dinamici senza ricorrere ad interfacce interamente realizzate in Flash.
Infatti la libreria incorpora una serie di metodi nativi per la realizzazione di effetti grafici, come per esempio la visualizzazione o la sparizione di un oggetto html mediante i metodi show e hide.
Inoltre esiste anche un metodo relativo alle animazioni chiamato animate(), con il quale è possibile realizzare effetti personalizzati.

jQuery ha molti metodi e costrutti per la gestione degli eventi, come per esempio la capacità di eseguire compiti al caricamento della pagina, la gestione di eventi semplici, come onload (), onclick (), onResize (), o di eventuali eventi composti.
Un particolare evento gestito da jQuery è AJAX.
Ajax è un mezzo di caricamento dei dati dal server al client (nel nostro caso un browser), senza che avvenga un ricaricamento visibile della pagina.
Questi dati possono assumere molte forme come il formato JSON, di cui si parlerà nel paragrafo successivo.
Ajax sta per "Asynchronous JavaScript and XML" ed è una tecnica per la creazione di pagine web veloci e dinamiche. Permette alle pagine web di aggiornarsi in modo asincrono scambiando piccole quantità di dati con il server.
Tale tecnica consente di aggiornare parti di una pagina web, senza ricaricare interamente la pagina.
Effettuare una chiamata AJAX attraverso l'aiuto di jQuery è semplice. Infatti jQuery ci mette in grado di effettuare richieste mediante GET o POST ed associarne determinati eventi a seconda che l’operazione sia andata o no a buon fine mediante verifica dello stato della risposta.

3.2.1 Introduzione a jQuery UI

JQuery UI è una libreria open source di plug-in basata sulla libreria jQuery. Ogni componente di tale libreria è costruito secondo l'architettura di jQuery ed è completamente personalizzabile.
jQuery UI fornisce interazioni ed animazioni, effetti avanzati e widget, ed è suddivisibile in 4 macro aree:

- Interactions: drag&drop, ridimensionamento, ordinamento e selezione;
- Widgets : numerosi widget
- Effects : transizioni animate
- Utilities: utilità di basso livello da usare per le interazioni, i widget e gli effetti

Nella nostra applicazione verrà utilizzata una libreria chiamata angular-uitree scritta in AngularJS e basata sul funzionamento del drag&drop di jQuery UI.

3.3 AngularJS

AngularJS[10] è un framework Javascript creato da Google.

Viene definito come un'infrastruttura per la creazione di applicazioni, citando la documentazione ufficiale:

Angular è quello che HTML avrebbe dovuto essere se fosse stato progettato per sviluppare applicazioni[11].
Tale framework infatti nella definizione dell'interfaccia grafica potenzia l'approccio dichiarativo dell'HTML, grazie a tale approccio è possibile integrare e utilizzare molte funzioni utilizzando quasi esclusivamente l'HTML.
Inoltre AngularJS fornisce strumenti per costruire un architettura modulare che si adatta alla logica dell'applicazione e favorisce eventuali test effettuati sull'applicazione stessa.

Le principali caratteristiche e vantaggi di AngularJS sono:

- binding bidirezionale (*two-way binding*)
- dependency injection
- supporto a MVC
- supporto ai moduli
- separazione delle competenze
- testabilità del codice
- riusabilità dei componenti

Nei paragrafi riportati di seguito andremo a riportare le funzionalità e le componenti principali del framework.

3.3.1 Attributi ng-

Gli attributi contenuti in AngularJS vengono definiti con il prefisso ng, tali attributi sono chiamati anche direttive e sostanzialmente sono dei marcatori HTML che definiscono il comportamento dell'elemento HTML a cui vengono applicate.
La direttiva principale nell'utilizzo di tale framework è ng-app, tale attributo definisce il contesto dell'applicazione ed è lo strumento che utilizza AngularJS per determinare l'ambito o l'interfaccia grafica in cui esso deve lavorare.

Solitamente tale direttiva si imposta sul body della pagina HTML che rispecchia l'interfaccia della nostra applicazione.

```
<body class="body-admin" ng-app="demoApp">
…
</body>
```

Il funzionamento che sta dietro a tale direttiva è il seguente: dopo che il framework è stato caricato avviene la creazione dell'oggetto globale angular e l'avvio del cosiddetto bootstrap (bootstrap è definito come il processo di inizializzazione del framework e di analisi dell'applicazione). In tale fase il framework scandisce il codice HTML della pagina corrente alla ricerca della direttiva ng-app che individua la root della nostra applicazione, in seguito tutti gli elementi figli dell'elemento principale (marcato con ng-app) verranno analizzati da AngularJS alla ricerca di ulteriori direttive ed espressioni da valutare.

Un'altra direttiva moto importante nell'ambito di AngularJS è quella definita dall'attributo ng-model, tale attributo definisce un modello di dati che viene anch'esso associato ad un elemento HTML. Tale direttiva definisce una variabile AngularJS a cui viene assegnato automaticamente il valore appartenente all'elemento HTML a cui essa è associata.

```
<input ng-keypress="onKeyPress(selectedNode)"
       name="itemname" type="text" required
       class="form-control" id="name" ng-model="selectedNode.name"
       ng-maxlength=50>
```

Perciò se l'elemento HTML è una casella di testo a tale variabile verrà assegnato il contenuto della casella di testo automaticamente e nello stesso momento in cui l'utente digiterà del testo nella casella di testo stessa.
Tale proprietà di AngularJS è chiamata Data Binding.
Altro elemento utilizzato spesso in AngularJS è l'espressione Angular, questa valuta il valore corrente della variabile o model scritto all'interno delle graffe {{…}}.
Solitamente l'espressione Angular è utilizzata per la stampa delle variabili in pagina e può essere integrata mediante dei filtri.

L'utilizzo dei filtri all'interno dell'interfaccia grafica è basato sull'uso dell'operatore `pipe` (|), all'interno di un'espressione i filtri che possono essere applicati sono diversi.

Inoltre possono essere applicati più filtri simultaneamente ad un espressione, in tale caso questi sono applicati in sequenza; l'espressione viene perciò elaborata partendo da sinistra verso destra, ossia dal primo filtro, da cui si ricava il risultato, fino all'ultimo.

Esistono anche Filtri con parametri, tali filtri prevedono il passaggio di parametri utili per la formattazione dell'espressione, anche in tale caso i filtri possono essere multipli, e vengono passati uno di seguito all'altro separati dai due punti (:).

I filtri possono essere creati appositamente, ma esistono anche filtri predefiniti come Lowercase ed Uppercase per formattare le stringhe, Number e Currency per formattare i numeri, Date per cambiare il formato di una data, ecc.

```
<li>Create
    <ul>User: {{selectedNode.createUser}}</ul>
    <ul>Date: {{selectedNode.createDate | date:"dd/MM/yyyy" }}</ul>
</li>
```

Un filtro molto importante è il filtro JSON, il quale può essere utilizzato come strumento di debug nella fase di progettazione dell'applicazione. Tale filtro trasforma in formato JSON l'oggetto al quale viene applicato.

Oltre agli attributi ng-app ed ng-model esistono anche altre direttive importanti, quali:

- ng-show & ng-hide: Tali direttive consentono la visualizzazione o l'offuscamento, in base alla valutazione di una espressione, degli elementi HTML a cui vengono applicate.
- ng-repeat: permette di ciclare su una collezione di dati e generare un elenco di elementi HTML.
- ng-class: tale attributo permette di impostare dinamicamente le classi CSS di un elemento HTML
- ng-include: consente di includere, nella vista corrente, del codice HTML esterno.
- ng-click: è una direttiva di evento simile all'onclick di JavaScript, ma a differenza di quest'ultimo la direttiva ng-click esegue l'assegnamento nel contesto AngularJS, garantendo il Data Binding sugli elementi.

3.3.2 MVC

I pattern di presentazione separano la logica applicativa dalla presentazione, questo comporta ad avere una maggiore indipendenza tra i componenti coinvolti ed una più semplice manutenibilità del software.

Analizziamo l'applicazione del pattern MVC in AngularJS.

In AngularJS una View è il risultato visivo generato a partire da un template HTML eventualmente arricchito da direttive.

Tutto il contesto HTML gestito da un'applicazione AngularJS rappresenta una View. Come è già stato detto la direttiva ng-app prende in carico la gestione di

tutto il codice HTML controllato da essa ed applica automaticamente il binding e la valutazione delle espressioni Angular.

Tuttavia, per un controllo più preciso su cosa visualizzare all'interno di una View e dell'interazione con l'utente, occorre utilizzare un controller.

Il controller di AngularJS viene definito mediante la scrittura di codice JavaScript.

Per associare un controller ad una pagina, ed effettuare perciò il collegamento tra la nostra JSP e il controller, occorre dare un nome alla nostra applicazione mediante la direttiva ng-app.

Per definire un controller innanzitutto occorre introdurre il modulo.

Il modulo è il ponte che collega la pagina JSP al CONTROLLER e viene creato mediante la funzione globale di AngularJS: module.

Una volta creato il modulo occorre far corrispondere il nome del modulo con il nome assegnato alla direttiva ng-app, in tale modo angular saprà che il codice che gestisce l'applicazione si trova in questo modulo.

Nella definizione del modulo, perciò occorrerà includere il nome dell'applicazione ed eventuali dipendenze ad altri moduli.

Il controller perciò verrà incluso all'interno di un modulo mediante il metodo controller, tale metodo consente di associare un funzione ad una stringa corrispondete al nome con cui potremo richiamare il controller.

L'associazione tra la View e il Controller avviene mediante la direttiva ng-controller: siccome ogni modulo può contenere più controller ogni pezzo di codice HTML della pagina da noi definita potrà essere associato e di conseguenza utilizzare controller differenti.

La funzione che definisce il controller ha un parametro *$scope*.

Questo parametro, che viene condiviso con la View, inizia con il dollaro ($) per una convenzione comune a tutti gli oggetti di sistema.

Il compito dello *$scope* è la definizione del modello dei dati e la condivisione di tale modello con la View, perciò tutte le proprietà di questo oggetto sono direttamente accessibili dalla View.

3.3.3 Il DataBinding

Nel momento in cui l'utente da interfaccia grafica modifica un elemento, come per esempio inserisce testo in una casella di testo, se tale elemento è controllato da AngularJS e associato ad una variabile dello *$scope*, il valore che viene inserito nella casella di testo è istantaneamente percepito da modello dati e di conseguenza dal controller.

Tale meccanismo di sincronizzazione automatica in AngularJS è chiamato DataBinding,

La maggior parte dei sistemi di template supporta il DataBinding in una sola direzione (dal modello dei dati verso la View).

Perciò i dati del modello vengono combinati con il template HTML per generare la View visibile all'utente in questo caso, se il modello viene modificato, le modifiche non si riflettono automaticamente sulla View. Allo stesso modo, se l'utente modifica la View le modifiche non vengono automaticamente riportate sul modello dei dati.

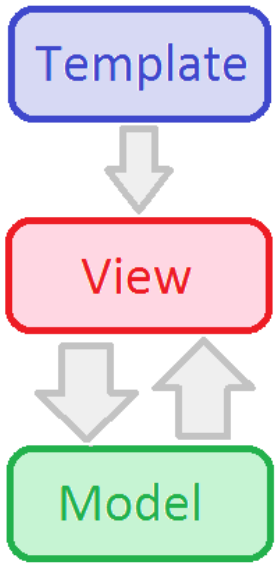

Figura 3.1: two-way binding

Il Meccanismo di DataBinding di AngularJS, rappresentato in *figura 3.1*, invece è bidirezionale (two-way binding) ossia ogni modifica posta al modello dati viene riflessa automaticamente sulla View e allo stesso modo ogni modifica alla View viene riportata sul modello dati.

Questa sincronizzazione avviene senza la necessità di scrivere altro codice, semplicemente è AngularJS che si occupa della sincronizzazione dei dati. È perciò sufficiente associare il modello dati allo *$*scope del nostro controller ed utilizzare la direttiva `ng-model` per richiamare ed utilizzare il modello di dati nella View.

3.4 Comunicazione tra Client e Server: JSON

Introduciamo il formato JSON (JavaScript Object Notation) come un formato adatto all'interscambio di dati fra applicazioni Client-Server.

Ci sono una varietà di opzioni standard per la comunicazione tra computer remoti, la scelta solitamente dipende dai requisiti delle applicazioni e dalle funzionalità che queste offrono.

JSON è un formato di dati che offre un interscambio leggero, semplice da leggere e da scrivere per i programmatori e facile da analizzare e generare per i computer.
Tale formato di testo è del tutto indipendente dal linguaggio di programmazione utilizzato, ma utilizza delle convenzioni familiari ai programmatori di C, C ++, Java, Javascript ecc.
Tali vantaggi rendono il formato JSON un linguaggio ideale per lo scambio di dati, e anche per tali motivi viene utilizzato in AJAX come alternativa al formato XML.

JSON prende origine dalla sintassi degli oggetti letterali in JavaScript. Gli oggetti sono definiti come coppie di proprietà/valori separate dalla virgola, l'intero oggetto viene poi racchiuso tra parentesi graffe. A differenza della notazione JavaScript (contenente anche funzioni e valori complessi) JSON ammette solo valori semplici ed atomici, tra cui:

- stringhe
- numeri
- array
- stringhe
- booleani
- elementi nulli

Un esempio di JSON:

```
{
 dashBoardId: "3ab78697-37b5-4228-9fd0-467595565317"
 descr: "Test DashBoard con Custom"
 id: null
 lastupdate: 1425373286707
 name: "DBTEST5"
 pages: [1]
   0:
       {
          layout: "1column"
          pageId: "d7306ac5-b5be-4561-966b-5939a5f46a40"
          pageNum: 1
       }
}
```

3.4.1 FlexJSON

Per la comunicazione tra client e server vengono perciò inviati dei documenti JSON.

Siccome lato server abbiamo degli oggetti Java, occorre trovare un modo per trasformare tali oggetti in un formato JSON e allo stesso modo, le richieste lato client contenenti variabili ed oggetti JavaScript devono essere convertiti in JSON.

Per tale motivo si utilizza la libreria FlexJSON[12], la quale offre dei metodi per la serializzazione e deserializzazione di oggetti Java.

In sostanza per poter utilizzare JSON insieme ad un'applicazione Java occorre trasformare gli oggetti Java in formato JSON e viceversa, in quanto solo in tale modo questi oggetti possono essere inviati e ricevuti mediante chiamate AJAX.

Ciò che distingue FlexJSON dalle altre librerie, è che con essa è possibile controllare ciò che deve essere serializzato di un determinato oggetto e delle sue dipendenze, permettendo un analisi profonda o superficiale dell'oggetto stesso.

Ora analizziamo come avviene la serializzazione e la deserializzazione di un oggetto con FlexJSON e come è possibile scegliere che parti dell'oggetto includere in tali operazioni.

- Serializzazione: la serializzazione è il procedimento che trasforma i nostri oggetti Java in tipo JSON.

  La serializzazione Basic è quella di default e comprende solo l'oggetto selezionato e le sue variabili semplici ed immediate tralasciando tutti i campi complessi e strutturati come le Collections Java quali mappe, liste, ecc.

  Perciò fa una scrematura di primo livello dell'oggetto da serializzare, siamo noi che possiamo decidere mediante alcune notazioni quali oggetti esclusi da FlexJSON vogliamo vengano inclusi nel nostro JSON.

  Esistono diversi metodi per fare questo come l'utilizzo dell'operazione Includes o della notazione.

  Il primo metodo consiste nel chiamare il metodo JSONSerializer in questo modo:

```
return new JSONSerializer().include("parts").serialize(dashBoard);
```

  Mentre per il secondo metodo occorre includere tale notazione nel codice che definisce la classe del nostro oggetto dashBoard:

```
Public Class DashBoard{

       @JSON(include=true)
       List<DashBoardPart> parts;

      …

}
```

  Tale notazione è da impostare sopra alla collezione o all'oggetto che si vuole venga incluso nella serializzazione, nel nostro caso l'oggetto è la lista parts.

Nel caso si vogliano invece escludere dei campi dalla serializzazione con lo stesso metodo dell'inclusione (il primo citato) è possibile utilizzare la clausola exclude:

```
return new JSONSerializer().
       exclude("date").serialize(dashBoard);
```

Mentre con il secondo metodo citato mediante la notazione sotto riportata escludiamo il campo date dalla serializzazione:

```
Public Class DashBoard{

       @JSON(include=false)
       Date date;

}
```

Un altro tipo di serializzazione che FlexJSON rende disponibile è quella di tipo Deep.

Tale serializzazione, all'opposto di quella descritta in precedenza, effettua una serializzazione profonda, perciò include tutti i campi dell'oggetto che si vuole serializzare comprese le collezioni.

Questa serializzazione deve essere utilizzata con cautela, in quanto gli oggetti potrebbero contenere molte informazioni e dipendenze ad altri oggetti. Questo comporterebbe alla creazione di un JSON molto grande e di conseguenza ad un rallentamento della comunicazione.

Allo stesso modo che con la serializzazione Basic è possibile utilizzare le opzioni o le notazioni include ed exclude.

- Deserializzazione: La deserializzazione è il processo inverso alla serializzazione e verrà esercitato mediante l'oggetto JSONDeserializer.

Dopo aver dato in pasto una stringa JSON al JSONDeserializer, solitamente si memorizza l'oggetto restituito da questa deserializzazione in una mappa. In tale mappa si accede agli oggetti del JSON mediante la chiave che identificano i valori degli oggetti stessi.

Capitolo 4

Qery Mobile

4. Introduzione all'applicazione Qery Mobile

Dopo aver analizzato l'applicazione Qery e le tecnologie principali utilizzate andiamo a presentare nel dettaglio l'applicazione oggetto della trattazione: Qery Mobile.

L'applicazione è nata con lo scopo di creare un visualizzatore di Dashboard per dispositivi mobili, questa è perciò formata da una parte contenente un Visualizzatore Mobile e da una seconda parte contenente un pannello di amministrazione nato per la gestione delle Dashboard.

Inizialmente analizzeremo la struttura dei dati che è stata progettata per questa applicazione su database ed il legame che tale struttura ha con il database di Qery (presentato nel capitolo 1).

Infatti, come sappiamo dal primo capitolo, il Dataview può assumere diverse forme, quali: Griglia, Grafico, Mappa o Custom. La Dashboard si avvale di tale proprietà per effettuare la composizione di elementi diversi e produrre così report aziendali sintetici e consultabili in maniera semplice e veloce mediante quelli che sono all'ordine d'oggi gli strumenti mobili quotidiani quali: Tablet o SmartPhone.

In seguito presenteremo il funzionamento dell'applicazione Qery Mobile da due punti di vista: le chiamate REST e la Gestione delle Dashboard.

Le chiamate REST sono state ideate per il funzionamento dell'applicazione mobile, mentre la Gestione delle Dashboard viene implementata mediante un pannello di amministrazione, utilizzato per effettuare la creazione, modifica, eliminazione e organizzazione di Dashboard all'interno di una struttura gerarchica di Cartelle.

Infine accenneremo a sviluppi correnti ed eventuali sviluppi futuri connessi all'applicazione stessa e a Qery.

4.1 Database

In tale paragrafo analizzeremo la struttura che l'applicazione presenta su Database. Tale struttura poggia sulla struttura preesistente di Qery presentata nel primo capitolo, per questo motivo analizzeremo in principio la progettazione della nuova parte di database e in seguito il legame che tale parte ha con il database preesistente.

4.1.1 Progetto Logico

In tale paragrafo viene riportato lo schema logico del database in cui sono riportai tutti gli attributi delle entità del database comprese le chiavi primarie e le chiavi esterne.

Le chiavi primarie vengono sottolineate mentre le chiave esterne sono indicate mediante la parola chiave FK.

```
Folder(id_folder, id_parent, name, descr, createuser,
createdate, updatedate, updateuser)
      FK: id_parent REFERENCES Folder

Dashboard(id_dashboard, id_folder, name, subtitle, descr,
id_dvfilter, publicflag, createuser, createdate, updateuser,
updatedate)
      FK: id_folder REFERENCES Folder


Dashboard_json(id, id_dashboard, id_user, entrydate, json)
      FK: id_dashboard REFERENCES Dashboard


Dashboard_page(id_page, id_dashboard, pagenum, layout)
      FK: id_dashboard REFERENCES Dashboard

Dashboard_page_part(id_part, id_page, partnum, id_dv,
type_dv)
      FK: id_page REFERENCES Dashboard_page
```

```
Dashboard_target(id, id_dashboard, targetype, targetid)
      FK: id_dashboard REFERENCES Dashboard
```

4.1.2 Descrizione delle Tabelle

Segue una descrizione delle entità presenti nel database e dei loro attributi.

- Folder: Tale entità ha il compito di rappresentare una cartella in cui è possibile organizzare una struttura gerarchica di Dashboard ed ulteriori Folder.
  - id_folder: è l'identificativo univoco della cartella
  - id_parent: è l'identificativo del Folder in cui è contenuta la cartella id_folder, se la cartella è di root tale attributo è nullo.
  - name: descrizione breve della cartella
  - descr: descrizione estesa della cartella e del suo contenuto
  - createuser: identificativo dell'utente che ha creato la cartella
  - createdate: data di creazione della cartella
  - updateuser: identificativo dell'utente che ha effettuato l'ultima modifica alla cartella
  - updatedate: data di ultima modifica della cartella

- Dashboard: è l'entità che rappresenta il centro dell'applicazione, e viene utilizzata come un contenitore di pagine.
  - id_dashboard: identificativo univoco della Dashboard
  - id_folder: identificativo unico della cartella in cui è contenuta la Dashboard
  - name: descrizione breve della Dashboard
  - subtitle: sottotitolo , funge da estensione del'attributo nome
  - descr: descrizione estesa della Dashboard e del suo contenuto
  - id_dvfilter: identificativo del Dataview di filtro della Dashboard

- publicflag: flag utilizzato per indicare se la Dashboard è pubblica oppure privata
- createuser: utente di creazione della Dashboard
- createdate: data di creazione Dashboard
- updateuser: utente di ultima modifica Dashboard
- updatedate: data di ultima modifica Dashboard

❖ Dashboard_json : tale entità è stata inserita nel progetto per fare in modo di utilizzare una sorta di cache delle Dashboard, in modo da rendere più performante l'applicazione lato mobile.

- Id: identificativo univoco del json
- id_dashboard: identificativo univoco della Dashboard, la cui definizione è contenuta nel json
- id_user: identificativo univoco dell'utente che crea il record
- entrydate: data di inserimento del json
- json: json vero e proprio, contiene la definizione della Dashboard e tutti i suoi contenuti quali fitri, pagine, parti e Dataview.

❖ Dashboard_page: questa tabella contiene tutte le possibili pagine di cui è formata una Dashboard. Una Dashboard deve contenere minimo una pagina, nonostante questo non viene definito un limite superiore di pagine che può contenere.

- id_page: identificativo univoco della pagina
- id_dashboard: identificativo univoco della Dashboard in cui è contenuta la pagina
- pagenum: numero della pagina
- layout: layout che la pagina possiede, ossia numero di parti e disposizione di quest'ultime all'interno della pagina

- Dashboard_page_part: entità rappresentate le parti di una pagina. Ogni parte può contenere oggetti quali: Dataview, Grafici, Mappe o Custom.
  - id_part: identificativo univoco della parte
  - id_page: identificativo univoco della pagina in cui è contenuta la parte
  - partnum: numero della parte. Importante per la disposizione delle parti all'interno della pagina.
  - id_dv: identificativo univoco del Dataview contenuto all'interno della parte. è proprio questo attributo che crea il legame tra Qery e Qery Mobile.
  - type_dv: tipo del Dataview incluso nella parte: GRID, CHART, MAP o CUSTOM.

- Dashboard_target: è la tabella relativa ai permessi. Mediante tale entità si gestiscono gli utenti che hanno i permessi per visualizzare la Dashboard. Tale tabella deve essere popolata se il publicflag della Dashboard è settato a false.
  - Id: identificativo univoco del target
  - id_dashboard: identificativo univoco della Dashboard di cui si definiscono i permessi
  - targetype: dice se il permesso è relativo ad un Utente singolo, un Ruolo o un Gruppo.
  - targetid: identificativo dell'utente, ruolo o gruppo che hanno il permesso di visualizzare la Dashboard.

4.1.3 Schema E/R

L'entità principale dell'applicazione è la Dashboard, a questa sono collegate tutte le altre entità.

Per migliorare la comprensione del seguente schema si è deciso di presentarlo senza gli attributi, il diagramma verrà presentato in modo completo nella *figura 4.1b* alla fine del paragrafo.

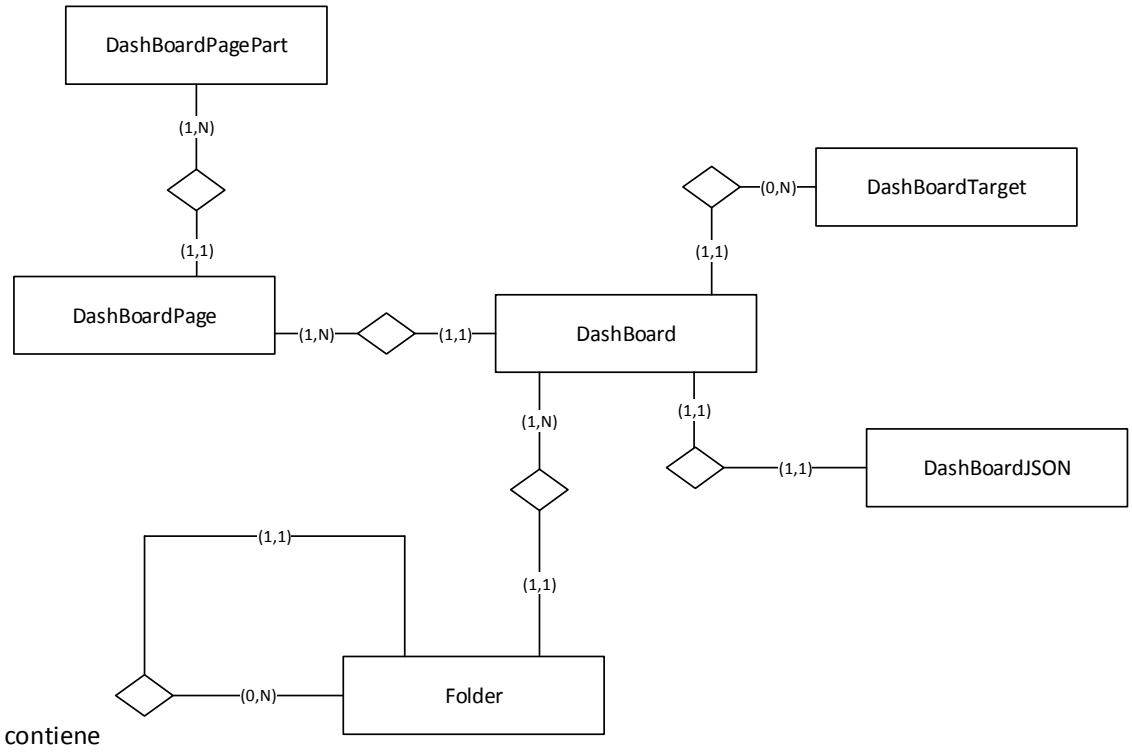

Figura 4.1a: Schema E/R Qery Mobile

Dallo schema presente in *figura 4.1a* si può notare che le Dashboard sono contenute in Folder, questo avviene per una maggiore organizzazione dell'ambiente di lavoro relativo all'utente che utilizzerà l'applicazione.

Tale struttura richiama quella presente tra Project e Dataview, vista nell'applicazione Qery, con una piccola ma sostanziale differenza. Infatti in tale applicazione la struttura delle cartelle e delle Dashboard è ricorsiva. Questo significa che ogni Folder oltre che poter contenere più Dashboard, può anche contenere altre cartelle al suo interno che di conseguenza possono contenere altre Dashboard o cartelle, e così di seguito.

Inoltre una Dashboard è formata da una o più pagine, le quali contengono una o più parti.

I target o permessi sono legati alla Dashboard mediante un'associazione 1:N, per il motivo che una Dashboard può avere più permessi, e perciò utenti abilitati a visualizzarla, ma un singolo permesso nel momento in cui esso viene creato è legato ad una sola Dashboard.

Infine la tabella JSON permette di tenere memorizzato il JSON della Dashboard in modo da aumentare le prestazioni sui dispositivi mobili, in cui il tempo di risposta aumenta nel momento in cui la stessa Dashboard, per esempio, viene consultata una seconda o terza volta.

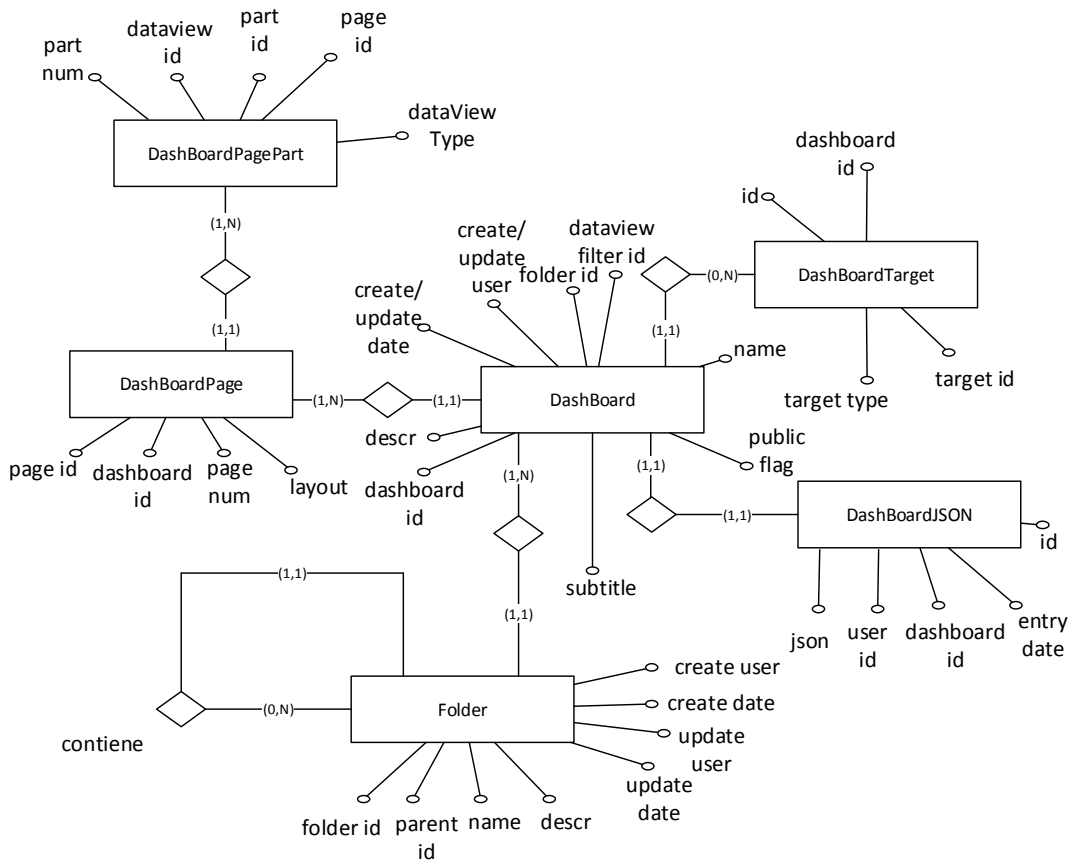

Figura 4.1b: Schema E/R Qery Mobile con attributi

4.1.4 Database a Confronto

Possiamo quasi dire che Qery Mobile sia un'estensione dell'applicazione Qery, in quanto queste appartengono allo stesso schema di dati e sono perfettamente legate tra loro.

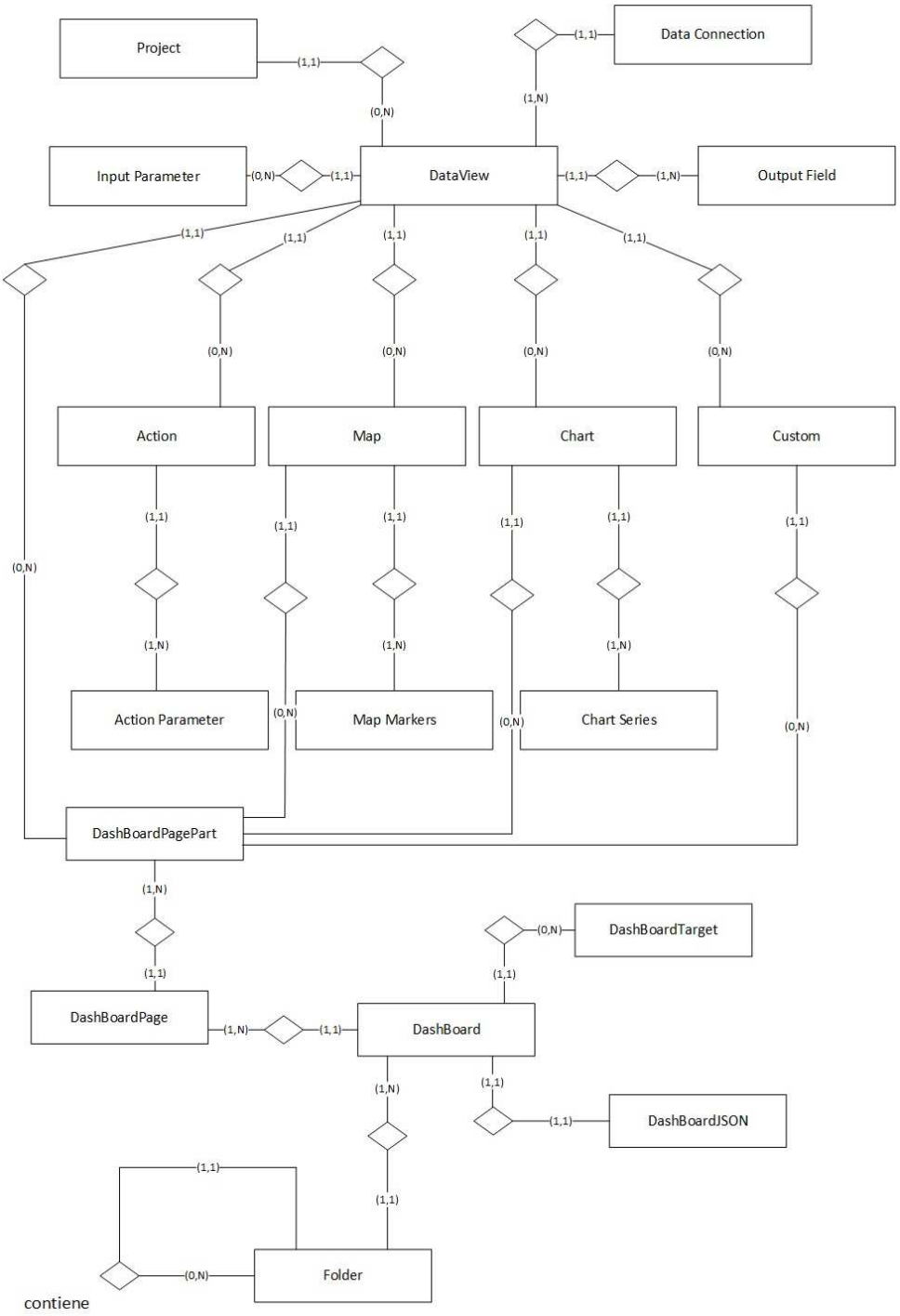

Figura 4.2: Schema E/R Finale

È possibile capire meglio questo concetto di “legame” tra le due applicazioni osservando la *figura 4.2*, da tale immagine vediamo che ogni parte contenuta in una pagina, che a sua volta è contenuta in una Dashboard, è legata ad un Dataview, Un Grafico, una Mappa oppure un Custom.

4.2 Struttura dell'applicazione: Gestione Dashboard

L'applicazione QeryMobileAdmin funge da pannello di amministrazione dell'applicazione qeryMobile sviluppata per Tablet, Smartphone e altri dispositivi mobili.

QeryMobileAdmin si occupa di creare contenitori di Dataview, Grafici, Mappe e Custom tali contenitori vengono chiamati Dashboard.

Iniziamo descrivendo il funzionamento del pannello di controllo delle Dashboard. Tale pannello è stato progettato per essere il più possibile user-friendly, perciò semplice ed intuitivo per l'utente.
Per esporre le funzionalità di tale pannello seguiremo lo schema presentato in *figura 4.3*, questo raffigura l'interfaccia di gestione delle Dashboard ed i punti principali rappresentanti le funzioni che andremo a descrivere.

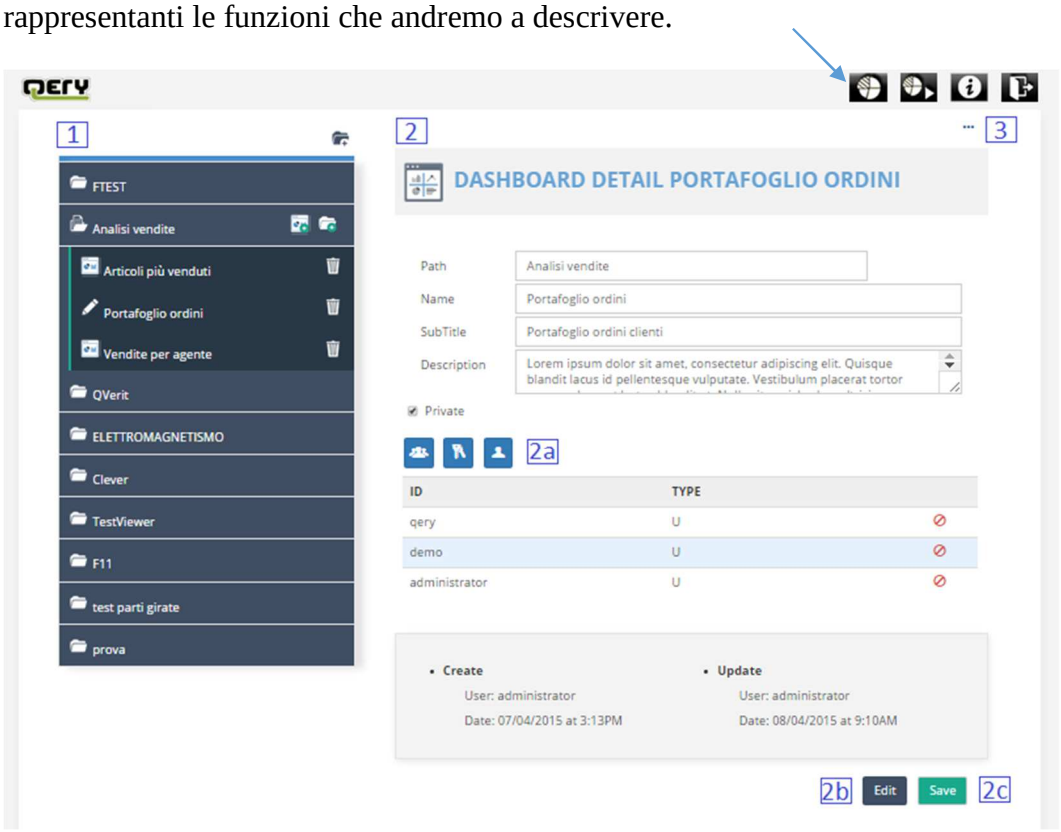

Figura 4.3: Pannello di controllo Dashboard

Innanzi tutto possiamo notare dal logo in alto a sinistra che il nostro pannello di controllo di Qery Mobile si trova all'interno dell'applicazione Qery, infatti cliccando sul logo Qery si passa alla home di Qery con la lista dei progetti descritta nel *capitolo 1*.
Mentre l'icona che contraddistingue la gestione delle Dashboard di Qery Mobile è l'icona contrassegnata dalla freccia blu.
Capiremo il significato della seconda icona, affianco all'icona di QeryMobile, nel paragrafo *4.4 Sviluppi Futuri.*

4.2.1 L'albero di Navigazione

Con l'albero di navigazione, presentato in *figura 4.3*, è possibile decidere in che percorso creare le proprie Dashboard o crearsi il proprio percorso di lavoro mediante la funzione crea cartella.
In questo modo è possibile gestire la propria area di lavoro autonomamente e tutto da un'unica interfaccia.

L'albero di navigazione di Qery Mobile è stato progettato partendo da una struttura ricorsiva delle cartelle nel database.
Tale struttura ha portato alla costruzione di una gerarchia di cartelle e di Dashboard che rende possibile una migliore organizzazione dell'ambiente di lavoro dell'utente.
Oltre alla funzionalità di navigazione l'albero possiede altre funzioni, le troviamo raffigurate in *figura 4.4*, sono:

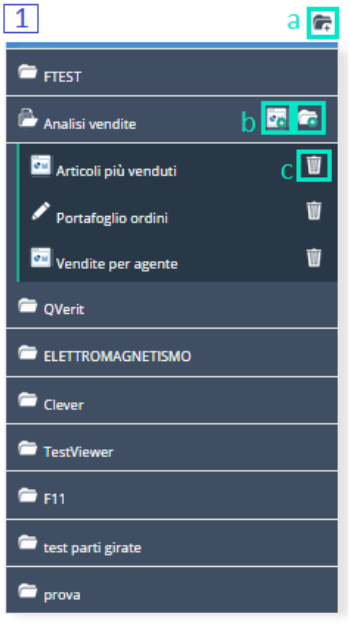

Figura 4.4: Albero di Navigazione

- Pt.a: Creazione di una nuova cartella di root
- Pt.b: Creazione di una nuova Dashboard

- Pt.b: Creazione di una nuova sottocartella

- Pt.c: Eliminazione di un elemento

Nel momento in cui verrà creata una nuova cartella di root, mediante la funzione del pt.a, verrà aggiunto un elemento in fondo all'albero. A questo elemento verrà dato un nome di default e comparirà al suo fianco un asterisco che indica un elemento appena creato ma non ancora salvato.
Le icone di creazione(pt.b), di una nuova cartella o di una nuova Dashboard, aggiungono un elemento all'interno della cartella selezionata, che corrisponde a quella in cui sono posizionati i pulsanti che andremo a cliccare.
Infine l'icona di eliminazione al punto c, serve per eliminare l'elemento della riga selezionata corrispondente all'icona cestino.
Perciò cliccando sulle icone indicate dal punto b verranno aggiunte Dashboard e cartelle all'interno della cartella "Analisi vendite", mentre mediante l'icona cestino del punto c verrà eliminata la Dashboard Articoli più venduti.
A differenza delle prime due operazioni, che per essere registrate necessitano la chiamata all'operazione di salvataggio, l'operazione di eliminazione è definitiva, perciò nel momento in cui si decide di eliminare un elemento questo è eliminato definitivamente dai dati.
Ora passiamo ad analizzare l'implementazione dell'albero di navigazione creato in AngularJS mediante lo script della libreria angular-ui-tree.

Angular Tree è un componente per l'interfaccia utente scritto in AngularJS che rende possibile la rappresentazione di liste nidificate e il drag & drop.
Siccome tale libreria è stata progettata con AngularJS utilizza lo *$scope* e sfrutta, perciò il Data Binding.
Tale componente ha anche una funzione di drag & drop per cui è possibile spostare elementi attraverso l'intero albero. Questa funzione è stata disattivata nella nostra applicazione, mediante la direttiva "`data-nodrag`", in quanto non necessaria.
La struttura dell'albero di navigazione, mediante le direttive, si presenta in modo simile a quella proposta qui di seguito:

ui-tree	Radice dell'albero
ui-tree-nodes	Contenitore di Nodi
ui-tree-node	Nodo Figlio
ui-tree-nodes	Contenitore di Nodi
ui-tree-node	Nodo Figlio
ui-tree-node	Nodo Figlio
ui-tree-node	Nodo Figlio
ui-tree-nodes	Contenitore di Nodi
ui-tree-node	Nodo Figlio

Nella prima parte del codice sopra riportato abbiamo lo script che genera l'albero di navigazione, mentre nella seconda parte è presente il codice dove lo script viene incluso in pagina. Per chiarezza chiameremo lo script della prima parte `items_renderer.html`.
La struttura dell'albero è ricorsiva, il codice al di sotto dello script `items_renderer.html` legge la struttura dell'albero di navigazione dal JSON memorizzato nello *$scope* e mediante la direttiva `ng-repeat` include più volte il codice di tale script. Questo script rappresenta ogni nodo del nostro albero nella nostra pagina.
Mentre nello script `items_renderer.html` è presente un secondo ciclo che controlla se il nodo corrente ha eventualmente altri figli, se è così richiama se stesso per disegnare anche questi.
Il JSON contenente l'albero di navigazione viene popolato nel momento in cui viene caricata la pagina, mentre le informazioni relative ai singoli nodi sono caricate nel momento in cui vengono richieste, perciò al click del singolo elemento sull'albero di navigazione.

```
<script type="text/ng-template" id="items_renderer.html">
  <div ui-tree-handle ng-class="expandLine(item)">
      <a data-nodrag ng-click="toggleItem(item)">

           <span ng-class="iconClass(item)"></span>

      </a>
      <a class="item" ng-click="clickItem(item)">

               <span ng-class="selectItem(item)">
                   {{item.name}}</span> 
               <span ng-show = "item.toSave == true ||
                item.new == true"
                class="glyphicons asterisk trasparent">

      </a>
      <a class="pull-right" data-nodrag ng-click="removeItem(item)"
      ng-show="item.children==null || item.children.isEmpty ||
      item.children.length<1 ">

               <span class="delete-item"></span>
      </a>
      <a class="pull-right" data-nodrag ng-click="newFolder(item)"
      style="margin-right: 8px;" ng-hide="item.type=='dashboard' ||
      item.expand==false || item.expand==null">

               <span ng-class="addFolderIcon()"></span>

      </a>
      <a class="pull-right" data-nodrag ng-click="newDashBoard(item)"
      style="margin-right: 8px;" ng-hide="item.type=='dashboard' ||
      item.expand==false || item.expand==null">

                 <span ng-class="addDashBoardIcon()"></span>

      </a>

   </div>
   <ol  ui-tree-nodes="options" ng-model="item.children"
    ng-class="hiddenItem(item)">
     <li  ng-repeat="item in item.children" ui-tree-node
      ng-include="'items_renderer.html'">
     </li>
   </ol>
</script>


<div ui-tree="options">
  <ol ui-tree-nodes ng-model="list" >
    <li ng-repeat="item in list.navigation" ui-tree-node
        ng-include="'items_renderer.html'">
    </li>
  </ol>
</div>
```

4.2.2 Form di Modifica di Dashboard e Folder

Una volta creato il percorso e la Dashboard per modificarla basta cliccare sul nodo di menu che ci interessa, subito affianco verrà visualizzata l'interfaccia per modificare i campi base della Dashboard.

Facendo riferimento al form in *figura 3.5* notiamo che questo è formato da due parti differenti: una prima parte con i campi relativi alla descrizione della Dashboard, come il percorso in cui è inclusa, il nome, il sottotitolo e la descrizione, ed una seconda parte contenente una tabella dei permessi di visualizzazione della Dashboard.

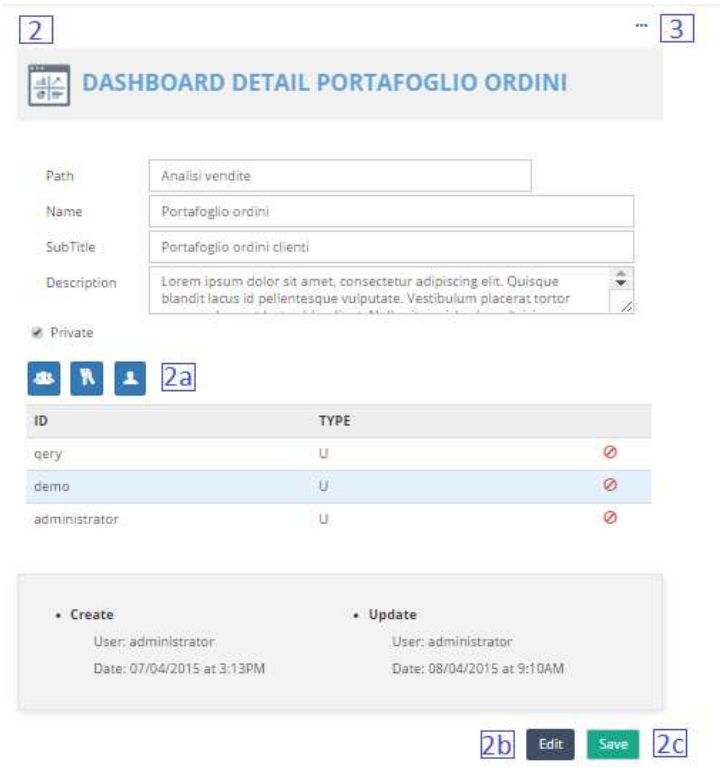

Figura 4.5: Form Modifica Dashboard

Ogni Dashboard può essere con visibilità pubblica o privata: nel primo caso questa può essere visualizzata da tutti gli utenti di qery, mentre nel caso la Dashboard sia privata può essere visualizzata soltanto da determinati utenti ruoli o gruppi decisi dal creatore della Dashboard.
Per questa funzionalità viene inserito nell'interfaccia utente un piccolo pannello presente al punto 2a, inizialmente nascosto in caso di Dashboard pubblica che viene visualizzato solo nel momento in cui la Dashboard è settata a privata.

In tale tabella è possibile aggiungere gruppi, ruoli o utenti che possono visualizzare la Dashboard, ma è anche possibile disabilitare i permessi già assegnati mediante il simbolo rosso di divieto a destra della riga rappresentante l'id e il tipo di ogni permesso (U: utente, G: gruppo, R: ruolo).
Per aggiungere nuovi permessi relativi a gruppi, ruoli o utenti è necessario utilizzare le icone blu al di sopra della tabella dei permessi.

Cliccando su una di queste icone verrà visualizzata una casella di testo implementata mediante la funzione di autocompletamento dello script allmighty-autocomplete scritto in AngularJS.

```
<autocomplete ng-model="searchString" attr-placeholder="type to
              search..."  data="gruList"  on-type="updategru">
</autocomplete>
```

Tale script fa in modo che ad ogni inserimento di caratteri in questa casella di testo, da parte dell'utente, venga visualizzata una tendina che corrisponde ai possibili gruppi, ruoli o utenti del sistema che fanno match con la stringa inserita dall'utente nell'apposita casella.
Riportiamo lo screenshot, in *figura 3.6*, che riporta il funzionamento dell'autocompletamento.

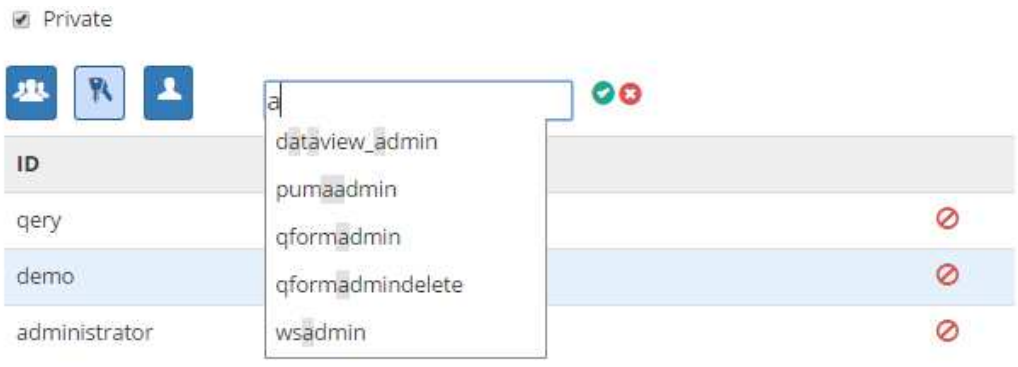

Figura 4.6: Autocompletamento

Dopo aver selezionato il ruolo, cliccando sull'icona verde, questo verrà aggiunto alla tabella.

Anche le cartelle hanno un loro form di modifica che differisce da quello presentato per un minor numero di campi; infatti di ogni cartella è possibile visualizzare il path e modificare soltanto il nome e la descrizione della cartella stessa.
A titolo informativo, sia per Dashboard che per cartelle, in fondo al form di modifica viene inserito:

- L'utente e la data di creazione dell'elemento
- L'utente e la data di ultima modifica dell'elemento

Inoltre il pulsante di modifica indicato nel punto 2b per le cartelle non esiste, ma è presente solo il pulsante di salva.

4.2.3 Composizione di una Dashboard

Una volta creata la Dashboard e affidatogli i vari permessi è possibile modificarne il contenuto mediante il pulsante Edit.
Cliccando su tale pulsante, rappresentato dal punto 2b della *figura 4.5*, si apre una modale (*figura 4.7)*, tale modale viene utilizzata per effettuare la vera e propria composizione delle Dashboard.

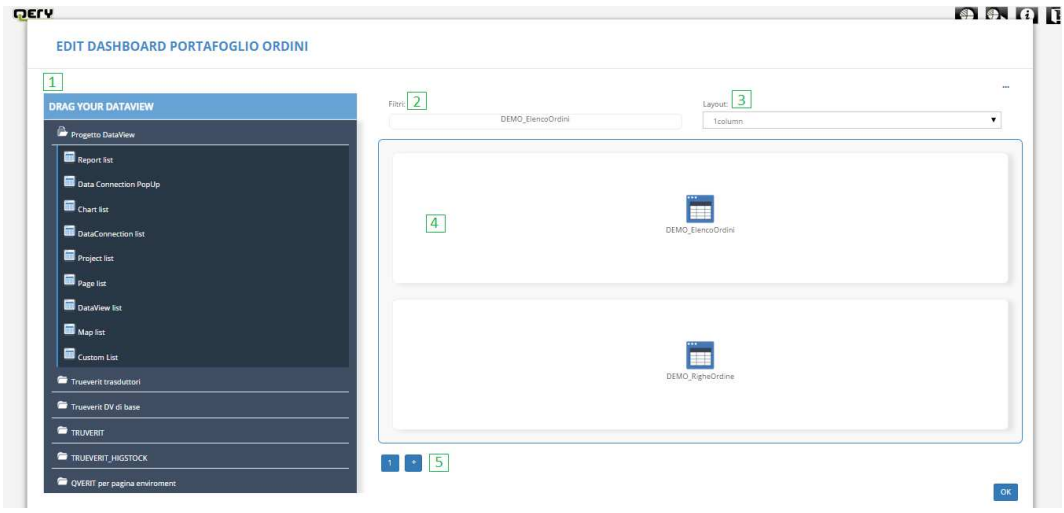

Figura 4.7: Modale di Composizione Dashboard

Sulla sinistra si trova un altro albero di navigazione, a differenza del precedente, questo è a soli due livelli in quanto contiene i progetti e i Dataview che possono essere inseriti nella Dashboard.
Le griglie, i grafici, le mappe e i custom sono contraddistinti da icone diverse e sono trascinabili mediante il drag&drop nelle parti della pagina rappresentata sulla destra.
Il punto nr.2 rappresenta il filtro della Dashboard, questo può essere soltanto un Dataview di tipo griglia e solitamente i suoi dati sono legati ai Dataview della Dashboard. Questo avviene perché nel momento dell'esecuzione questo campo deve filtrare i dati della Dashboard stessa.
La Dashboard è composta da più pagine ed ogni pagina può avere layout differenti. Il terzo punto che l'utente deve selezionare è il layout della nostra pagina: questo serve per decidere il numero e la disposizione delle parti all'interno della pagina. Esistono diversi layout, ed è possibile inserirne a piacere mediante la semplice aggiunta di un file HTML alla cartella template dell'applicazione.
Esempi di Template vengono riportati di seguito, nella *figura 4.8*.

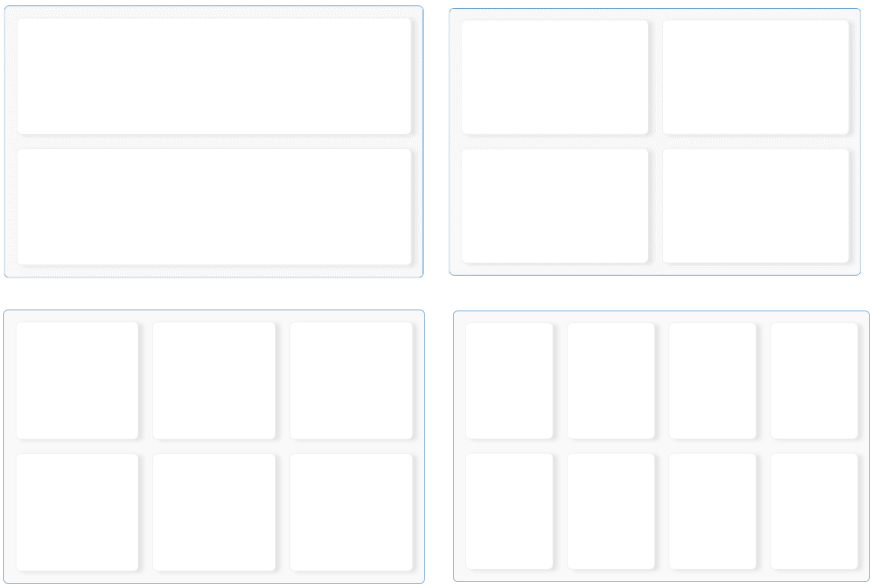

Figura 4.8: Esempi di Layout di una Pagina

La parte principale di questa parte dell'applicazione è il drag&drop.
Questo viene implementato mediante lo script angular-dragdrop scritto in AngularJS.
Tale script rappresenta una libreria esterna che implementa il drag&drop di jQueryUI e serve per trascinare dei componenti all'interno della pagina HTML.
Tali componenti per noi sono i Dataview presenti nell'albero di navigazione, questi possono essere trascinati e rilasciati all'interno delle parti della pagina corrente appartenente alla nostra Dashboard. Nel caso la parte contenga già un elemento questo verrà sostituito con l'elemento che andremo a trascinarci all'interno in un secondo momento.
In *figura 4.9* viene visualizzato il comportamento dell'interfaccia nel momento in cui si effettua il drag&drop di un elemento all'interno di una parte.

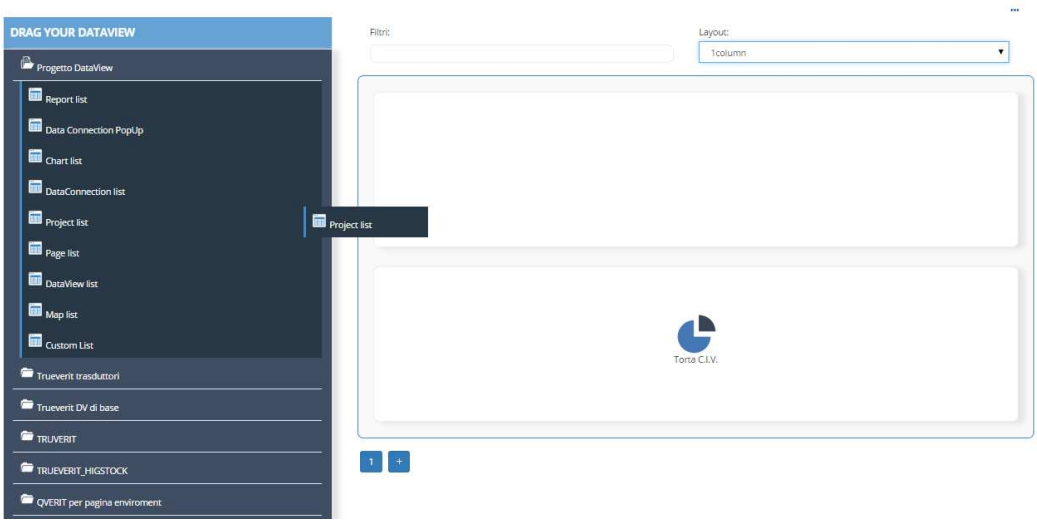

Figura 4.9: Esempio di Drag&Drop

L'elemento "draggato" dall'albero viene perciò duplicato dall'albero stesso e portato nella parte dove, una volta "droppato" ossia rilasciato, verrà rappresentato con un icona significativa rappresentante il tipo di elemento: Griglia, Grafico, Mappa o Custom.
Passando agli aspetti implementativi, tale script rende possibile il drag&drop mediante delle direttive chiamate data-drag, data-drop, jqyoui-draggable e jqyoui-droppable.

Le prime due direttive vanno applicate all'elemento sui cui potrà essere effettuato il drag e il drop, mentre le direttiva jqyoui-draggable e jqyoui-droppable servono per definire altre proprietà del drag&drop come, per esempio, la duplicazione dell'elemento "draggato", descritta in precedenza e l'attivazione o meno dell'animazione, la quale rende visibile il trascinamento dell'elemento.

Il primo script implementato del progetto è simile a quello dell'albero di navigazione delle Dashboard analizzato al *paragrafo 4.1*.

Questo script infatti implementa l'albero di navigazione dei progetti (al punto 1 della *figura 4.1* relativa alla modale), ma a differenza della struttura presentata in Qery all'interno di ogni progetto non sono presenti soltanto i Dataview con al loro interno grafici, mappe e custom, ma sono presenti in modo distinto tra loro ed allo stesso livello griglie, grafici, mappe e custom.

Tale scelta è stata fatta in modo che sia più semplice creare una Dashboard con tutti gli elementi desiderati, di qualunque tipo essi siano.

```
<script type="text/ng-template" id="items_rendererdv.html">

      <div ui-tree-handle ng-class="expandLine(item)" data-drag="true"
                          data-jqyoui-  options="{revert: 'invalid',
                          helper: 'clone'}">

           <a ng-click="toggleItem(item)">
                    <span ng-class="iconClass(item)"></span>
           </a>
           <a>{{item.name}}</a>
      </div>

      <ol ui-tree-nodes="options" ng-model="item.children"
                                  ng-class="hiddenItem(item)">
            <li ng-repeat="item in item.children" ui-tree-node
                            data-drag="true"
                            data-jqyoui-options="{revert: 'invalid',
                            helper: 'clone'}"
                            ng-model ="item" jqyoui-draggable="{index:
                            {{$index}},
                            animate: true, placeholder: 'keep'}"
                            ng-include="'items_rendererdv.html'">
            </li>
      </ol>
</script>
```

In questi script possiamo notare la presenza delle direttive citate in precedenza: data-drag, data-drop, jqyoui-draggable e jqyoui-droppable, le quali definiscono gli elementi che è possibile trascinare all'interno della pagina: i Dataview.

```
<div data-drop="true" ng-model='selectedPage.parts[count.index]'
                      jqyoui-droppable="{multiple:false}" class="drop">

      <div ng-show="selectedPage.parts[count.index].name"
           data-drag="true"
           data-jqyoui-options="{revert: 'invalid', helper: 'clone'}"
           ng-model="selectedPage.parts[count.index]"
            jqyoui-draggable="{index:{{count.index}},animate:true}" >

           <div style="line-height:10px;"
                ng-class="partType(selectedPage.parts[count.index])">
           </div>

            <div style="line-height:10px;">
                {{selectedPage.parts[count.index].name}}
             </div>

       </div>
</div>
```

Nel punto 4 della *figura 4.8* ci vengono indicate le parti: come già visto in precedenza il numero di parti in una pagina dipende dal layout che è stato scelto per la pagina stessa.

Ma come è possibile che un template HTML possa supportare il drag&drop?

Occorrerebbe ripetere il codice relativo al drop dell'elemento per ogni parte e per ogni template, il quale porta a duplicazione del codice e a poca leggibilità dello stesso.

```
<div class="col-md-12 part" id="part0" qx-qery-isolate-scope='0'>


</div>
<div class="col-md-12 part" id="part1" qx-qery-isolate-scope='1'>


</div>
```

Questo problema è stato risolto mediante l'implementazione di una direttiva chiamata qxQeryIsolateScope: tale direttiva fa in modo di assegnare ad ogni parte del template HTML, definito nel codice riportato sopra, un numero identificativo e di includere all'interno di ogni parte lo script di drop definito in precedenza.

In tale modo viene evitata la duplicazione del codice di drop della parte in ogni template, inoltre ogni template rimane pulito e facilmente inseribile anche dall'utente finale.

La direttiva qxQeryIsolateScope, viene definita nello stesso modulo del controller che gestisce la nostra applicazione, ed è riportata qui di seguito:

```
module.directive('qxQeryIsolateScope', function () {
return {
    restrict: 'A',
    scope: true,
    template:'<div
                ng-include="\'../../pages/dashBoard/drop.html\'">
              </div>',
    link: function(scope, element, attributes){
        scope.count = {};
        scope.count.index = attributes.qxQeryIsolateScope;
    }
  };
});
```

Tutta l'applicazione inizialmente gira lato client, perciò le operazioni di creazione e modifica di una Dashboard non vengono memorizzate su database, fino a che l'utente non decide di cliccare sul pulsante salva.
Le uniche operazioni che vengono effettuate prima del salva, mediante una o più chiamate al server, sono quelle di eliminazione di un elemento dall'albero di navigazione o di eliminazione di un permesso dalla tabella dei Target, oltre naturalmente a quelle iniziali di caricamento dei dati da database.

4.2.4 Strumenti di Debug

Infine il punto 3 della *figura 4.3* è stato implementato come strumento di debug e serve per controllare i dati contenuti nello $scope del nostro controller AngularJS.
I dati perciò vengono serializzati in JSON, mediante il filtro JSON di AngularJS (accennato al paragrafo AngularJS del *capitolo 3)*, e possono essere acceduti mediante l'icona rappresentata nel punto 3.
Cliccando su tale icona lo presentato in *figura 4.10*:

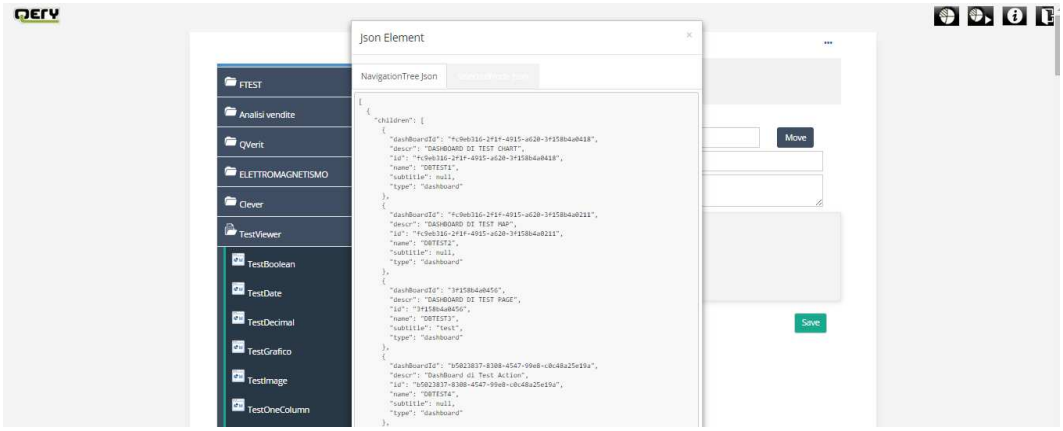

Figura 4.10: Modale di Debug

Cliccando questa icona perciò si aprirà una modale di consultazione dei dati su cui l'applicazione sta lavorando, rappresentata in *figura 4.10*.
In tale modo è semplice capire se i dati sono stati caricati in modo corretto oppure no e quindi se si sta lavorando con dati sporchi oppure se è l'applicazione che si comporta in maniera scorretta.

4.2.5 Altri Aspetti Implementativi

Di seguito riportiamo alcuni aspetti implementativi importanti per l'applicazione.
Perciò elencheremo i nomi e le funzionalità di ulteriori librerie, scritte in AngularJS, che sono state incluse nel progetto.

- angular-busy : script utilizzato per le chiamate al server.
  Nel momento in cui la promise effettua la chiamata per recuperare i dati da database, la direttiva cg-busy si occupa di visualizzare un messaggio di Loading in Pagina fino a che la chiamata non è terminata.

```
<div class="col-md-4"
     cg-busy="{promise:promise,message:'Loading...'}">
```

  La chiamata al server è effettuata con la promise di AngularJS e viene effettuata nel controller della nostra applicazione.

```
$scope.promise = $http({method:'GET',
                  url:$scope.baseUrl
                 +'/admin/dashBoard/dashBoardView.action?
                 task=getNavigationList'});

$scope.promise.success(success).error(_manageError);
```

Il risultato in pagina sarà perciò quello riportato in *figura 4.11*:

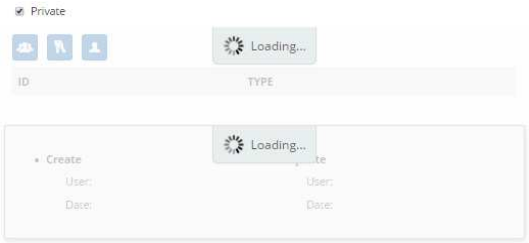

Figura 4.11: Loading di cg-busy

- angular-messages: tale script è utilizzato nell'applicazione per visualizzare eventuali messaggi di errore. Il codice inserito nell'applicazione relativo all'utilizzo di tale script è il seguente:

```
  <div ng-messages="dashBoardForm.itemname.$error" >
      <div ng-message="maxlength">
         <alert class="alert alert-danger" type="danger"
          style="clear:both;">
            <s:text name="dashboard.error.maxlength" />
         </alert>
       </div>
       <div ng-message="required">
          <alert class="alert alert-danger"  type="danger"
           style="clear:both;">
              <s:text name="dashboard.error.required" />
          </alert>
        </div>
        <div ng-message="duplicated">
           <alert class="alert alert-danger" type="danger"
            style="clear:both;">
                 <s:text name="dashboard.error.duplicated" />
           </alert>
         </div>
  </div>
```

Un errore di tale tipo verrà visualizzato nel caso che i campi richiesti ed obbligatori del form di Modifica di Dashboard e Cartelle non siano stati compilati o siano stati compilati in modo sbagliato.

Per esempio in *figura 4.12* viene rappresentato un errore relative al campo "name", il quale è obbligatorio nel form.

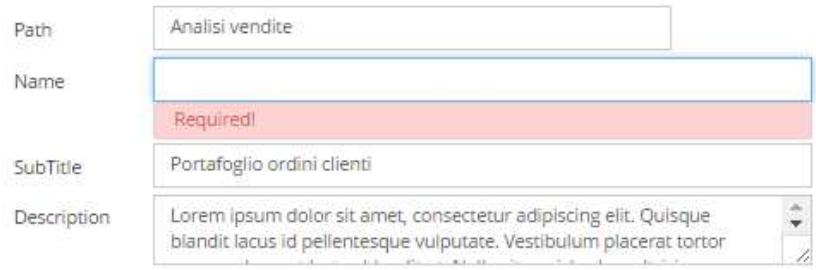

Figura 4.12: Alert Campo Richiesto

Un altro messaggio di errore dello stesso tipo si può avere nel caso in cui un permesso, nella tabella dei target, non sia valido o sia già presente.

- sweet-alert: infine la libreria sweetAlert viene utilizzata per la visualizzazione dei messaggi di dialogo dell'applicazione.

  Un esempio di codice relativo all'utilizzo delle sweetAlert è il seguente:

```
if($scope.selectedNode.filter !=null
   && $scope.selectedNode.filter.type != 'grid'){

       sweetAlert("Error!",
                  "The Filter Must Be of Type DataView", "error");

} else if(complete == true){

       $modalInstance.dismiss('ok');

} else {

       sweetAlert("Warning!","In your DashBoard there are empty
parts!", "warning");

}
```

Una piccola gestione degli errori mediante SweetAlert viene implementata anche per l'albero di navigazione. Infatti nel momento in cui viene creata una nuova cartella e al suo interno viene aggiunta una nuova Dashboard, senza prima salvare la cartella appena creata, verrà visualizzato un messaggio di errore che inviterà a salvare tale cartella prima di effettuare altre operazioni.

Altri messaggi di dialogo sono visualizzati in caso di successo in seguito al salvataggio di una Dashboard, oppure per avvertire l'utente mediante una Warning di eventuali parti della Dashboard non ancora completate.

Nella *figura 4.13* vengono riportati gli esempi relativi ai 4 tipi di sweet-alert utilizzati dall'applicazione.

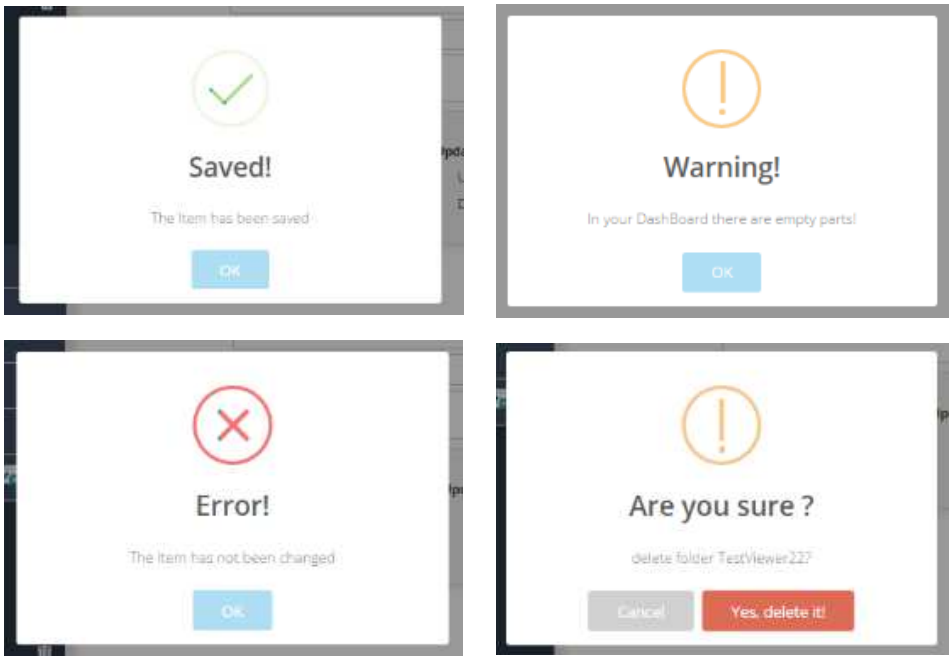

Figura 4.13: Tipi di sweet-alert

4.3 Struttura dell'applicazione: Chiamate REST

Per il funzionamento della parte Mobile dell'applicazione, prima che questa venisse realizzata, è stato necessario implementare delle chiamate API REST lato server che fornissero i dati riguardanti la struttura e i contenuti dell'applicazione stessa.
Per avere un idea di ciò che fa l'applicazione mobile inseriamo, in *figura 4.14*, delle pagine di esempio di QeryMobile su Tablet:

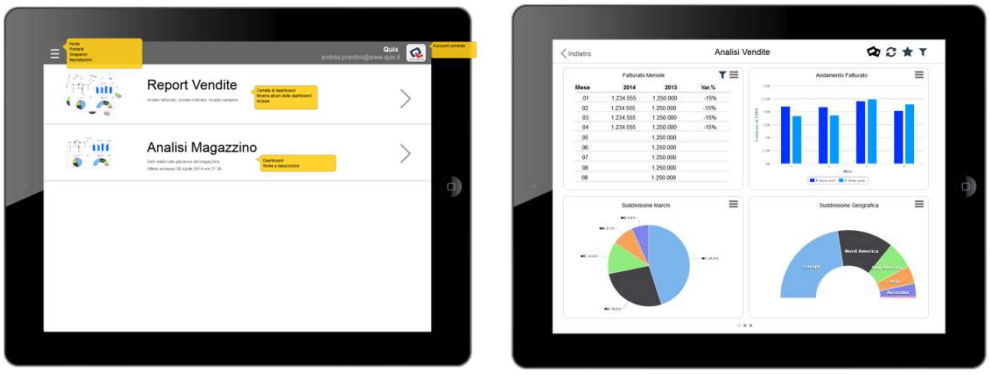

Figura 4.14: Esempio dell'applicazione mobile

Le pagine rappresentate in *figura 4.14* rappresentano un esempio di albero di navigazione sulla sinistra e un esempio di esecuzione di Dashboard sulla destra.
In tale paragrafo parleremo delle varie chiamate API REST che permettono il funzionamento di tale applicazione mobile.

4.3.1 Servizio di Login

Prima di tutto è stato creato un servizio di login, in quanto per tutte le chiamate al sistema di backend è necessaria l'autenticazione.
Tale servizio è stato gestito mediante la classe LoginManagerAction che viene richiamata mediante una richiesta URL contenente nome utente e password. Tale richiesta restituisce un codice di autenticazione necessario per le prossime chiamate al server.
In tale paragrafo presenteremo degli esempi di chiamate al nostro server di test: trueverit.

L'url che verrà chiamato per effettuare il login sarà il seguente:

```
http://trueverit.quix.it/qery/api/login.action?usr=qery&pwd=*****&nocac
he=1435305391093
```

Mentre la pagina visualizzata dal client avrà un contenuto simile a quello rappresentato nella *figura 4.15* riportata di seguito:


```
{
    "expire": "2015-06-26T13:56:31.271+0200",
    "auth_code": "cb720bdf-def7-4905-9f3e-1f4f1359a9e0"
}
```


Figura 4.15: Risposta al servizio di Login

Il codice auth_code sarà attivo solo per un determinato lasso di tempo, questa porzione di tempo è del tutto necessaria per effettuare le varie operazioni con l'applicazione mobile. Dalla *figura 4.15* è possibile osservare che da questa chiamata viene generato, oltre l'auth_code, un campo di tipo expire; Questo è il campo che ci indica la scadenza del nostro codice di autenticazione. Nel caso in cui la validità di questo codice venga a meno viene restituito un codice di errore di tipo 403, l'applicazione in questo caso si occupa di effettuare nuovamente il login per recuperare un nuovo codice di autenticazione. Questa operazione viene effettuata in modo del tutto trasparente per l'utente che utilizza l'applicazione.

A pagina seguente riportiamo il metodo della classe LoginManagerAction che gestisce tale servizio.

Possiamo notare che nel caso che l'utente o la password non siano presenti, o nel caso in cui il codice di autenticazione non sia stato generato in maniera corretta, il metodo restituisce un codice di errore di tipo HttpStatus.SC_FORBIDDEN (ossia la pagina di errore di tipo 403 di cui abbiamo accennato prima).

Mentre nel caso che l'username e la password siano presenti, viene controllata la loro validità mediante Puma (il gestore degli utenti Quix) che, nel caso il controllo sia positivo, genera il codice auth_code mediante il metodo createAuthCode.

```
private static Log log = LogFactory.getLog(LoginManagerAction.class);

private String usr;

private String pwd;

public String execute() throws Exception {

 HttpServletResponse response = ServletActionContext.getResponse();

 if (StringUtils.isEmpty(usr) || StringUtils.isEmpty(pwd)) {
     response.sendError(HttpStatus.SC_FORBIDDEN);
     return null;
}
// verifica autenticazione
try {
       if (PumaManagerFactory.getPumaManager().checkUser(usr, pwd)) {
             // utente valido
              Calendar c = Calendar.getInstance();
              c.add(Calendar.HOUR, 4);


              // genera un nuovo auth code
             String authCode =
             PumaManagerFactory.getPumaManager().createAuthCode(usr,
             c.getTime());

              if (authCode != null) {
                    SimpleDateFormat formatter = new
                    SimpleDateFormat(
                    "yyyy-MM-dd'T'HH:mm:ss.SSSZ");
                    Map<String, Object> jsonMap =
                    new HashMap<String, Object>();
                    jsonMap.put("auth_code", authCode);
                    jsonMap.put("expire", formatter.format(c.getTime()));
                     // genera il JSON
                     return sendAjaxResponseToClient(
                     JsonUtils.getJSONSerializer().serialize(jsonMap),
                     "UTF-8");
              } else {
                     // Un errore non ha permesso la creazione del codice
                     log.error("Error in generating authcode");
                     // in caso di errore torna un errore 503
                     response.sendError(
                     HttpStatus.SC_INTERNAL_SERVER_ERROR);
                     return null;
              }

       } else {
             // in caso di errore torna un errore 403
             response.sendError(HttpStatus.SC_FORBIDDEN);
             return null;
       }

} catch (Exception e) {
       log.error("Error in generating authcode", e);
       // in caso di errore torna un errore 503
       response.sendError(HttpStatus.SC_INTERNAL_SERVER_ERROR);
       return null;
}
```

4.3.2 Servizio di erogazione dell'albero

In seguito si è passati all'ideare un metodo ricorsivo per la costruzione e l'erogazione dell'albero di navigazione delle Dashboard.
Naturalmente la navigazione viene ottenuta in base all'account attivo, ossia in base all'utente che ha effettuato il login. Questo avviene in quanto ogni Dashboard ha i propri permessi di visualizzazione, come abbiamo spiegato nel paragrafo precedente.
La chiamata al servizio avviene sempre mediante un URL. Questo contiene il codice di autenticazione e l'identificativo del servizio che in questo caso è navigation.

```
http://trueverit.quix.it/qery/api/navigation.action?auth_token=cb720bdf-def7-4905-9f3e-1f4f1359a9e0
```

Il risultato che ci viene restituito da questa chiamata è in formato JSON ed è presentato in *figura 4.16*, perciò la serializzazione dell'albero avviene mediante il JSONSerializer di flexJSON.
Per un'anteprima più chiara del risultato ottenuto dalla chiamata utilizziamo l'applicazione di Google Chrome *AdvancedRestClient,* per effettuare la formattazione del nostro JSON.

Nell' anteprima in *figura 4.16a* possiamo notare la prima parte del JSON.
L'albero di navigazione è incluso sotto la voce "navigation", mentre lastupdate, ci informa sulla data di ultima modifica dell'albero.

Figura 4.16a: 1° Parte del JSON dell'albero di navigazione

Mentre nella *figura 4.16b* è possibile osservare la struttura ricorsiva dell'albero: abbiamo la cartella prova che al suo intero contiene la cartella prova2, la quale a sua volta contiene due Dashboard (D5 e D6) ed una terza cartella F211 la quale, infine, contiene la Dashboard D211.

Figura 4.16b: 2° Parte del JSON dell'albero di navigazione

La classe che effettua tale servizio è la NavigationManagerAction, tale classe effettua una chiamata esterna al DahboardHandler per la costruzione dell'albero di navigazione, una volta popolato l'albero questo verrà serializzato mediante JSONSerializer ed inviato al client.

```
return sendAjaxResponseToClient(new JSONSerializer().
                                exclude("*.class").
                                serialize(navigationView), "UTF-8");
```

Il DashboardHandler è una classe di generica utilità la cui istanza è creata una sola volta da Spring e può essere utilizzata e richiamata da tutta l'applicazione mediante la notazione *@Resource*.

```
@Resource (name = "dashBoardHandler")
public DashBoardHandler dashBoardHandler;
```

La classe DashboardHandler contiene metodi di utilità generale come il buildnavigationtree ed è richiamabile in tutta l'applicazione, come per esempio per la costruzione dell'albero di navigazione nella Gestione delle Dashboard vista al paragrafo precedente.

Analizziamo le parti più importanti del metodo che costruisce l'albero di navigazione: buildnavigationtree.

1°Parte: *Analisi delle cartelle*

```
for (DashBoard dashBoard : dashBoardList) {

        String folderId = dashBoard.getId_folder();
        FolderRootView folderRootView = new FolderRootView(folderId, true);
        Folder folder = navigationMap.get(folderRootView);

        if(folder == null) {
                folder = DAOFactory.getFolderDAO().get(dashBoard.getId_folder());
        }

        //Controllo la data di ultima modifica dell'albero per inserirla poi nel
          Json
        if (folder.getUpdateDate()!=null &&
            folder.getUpdateDate().getTime()>lastUpdate){

                lastUpdate=folder.getUpdateDate().getTime();

        } else if (folder.getCreateDate().getTime()>lastUpdate){

                lastUpdate=folder.getCreateDate().getTime();
        }
        if(dashBoard.getLastupdate()!=null &&
           dashBoard.getLastupdate().getTime()> lastUpdate){
                lastUpdate=dashBoard.getLastupdate().getTime();

        } else if(dashBoard.getCreateDate().getTime()>lastUpdate) {

                lastUpdate=dashBoard.getCreateDate().getTime();

        }
        if(folder.getParent()==null && !(rootFolder.contains(folder))){
                folderRootView.addAll(folder.getFolderId(), false);
                folder.setRoot(true);
                rootFolder.add(folder);
        } else {
                folderRootView.addAll(folder.getFolderId(), true);
```

```
        }

        folder.getChildren().add(dashBoard);
        navigationMap.put(folderRootView, folder);

}
```

Per prima cosa viene effettuato un ciclo sulle Dashboard, in tale metodo vengono utilizzate: una vista chiamata folderRootView ed una mappa navigationMap.
I passi effettuati dalla prima parte dell'algoritmo sono i seguenti:

- Recupero della cartella in cui è contenuta la Dashboard
- Controllo che la cartella non sia già presente nella navigationMap
- Se la cartella non è nella navigationMap la recupero da database
- Controllo la data di ultima modifica dell'albero per inserirla poi nel Json
- In seguito controllo se la cartella in cui è contenuta la Dashboard è una cartella di root, se è così e se questa non è ancora presente nella lista delle cartelle di root
    - Aggiungo l'id della Folder a FolderRootView con proprietà *incluse = false*
    - Aggiorno la proprietà di root della Folder a *true*

  Aggiungo la folder alla lista di rootfolder delle cartelle di root
  Altrimenti:
    - Aggiungo l'id della Folder a FolderRootView con proprietà *incluse = true*

  Tale passaggio ci aiuterà a tenere traccia delle folder non ancora aggiunte all'albero.
- Si aggiorna la proprietà children della cartella con l'istanza della Dashboard corrente
- Infine si inserisce nella navigationMap la nostra vista come chiave e l'istanza della nostra cartella.
  Perciò la folderRootView viene utilizzata come indice delle varie cartelle all'interno della navigationMap.

2°Parte: Visualizzazione Cartelle Vuote

```
if(emptyFolder == true){
        List <Folder> folderList = DAOFactory.getFolderDAO().getEmptyFolderList();

        for (Folder folder : folderList) {
                String folderId = folder.getId_parent();
                FolderRootView folderRootView = new FolderRootView(folderId, true);
                Folder folderR = navigationMap.get(folderRootView);

                if(folder == null) {
                        folder = DAOFactory.getFolderDAO().get(folderId);
                }
                if (folder.getUpdateDate()!=null &&

                    folder.getUpdateDate().getTime()>lastUpdate){

                        lastUpdate=folder.getUpdateDate().getTime();
                } else if (folder.getCreateDate().getTime()>lastUpdate){
                        lastUpdate=folder.getCreateDate().getTime();
                }
                if(folder.getParent()==null && !(rootFolder.contains(folder))){
                        folderRootView.addAll(folder.getFolderId(), false);
                        folder.setRoot(true);
                        rootFolder.add(folder);
                } else {
                        folderRootView.addAll(folder.getFolderId(), true);
                }
                folder.getChildren().add(folder);
                navigationMap.put(folderRootView, folder);
        }
}
```

Tale parte di codice viene eseguita soltanto se è stato richiesto di includere anche le cartelle vuote nell'albero di navigazione.

È stato aggiunto in quanto non tutte le classi che richiamano questo metodo volevano escludere le cartelle vuote.

Per esempio al DashBoardViewManagerAction, l'Action che si occupa della gestione del pannello di controllo delle Dashboard, occorreva che le cartelle vuote fossero visibili per una corretta costruzione dell'ambiente di lavoro (ricordiamo che prima di salvare una Dashboard in una nuova cartella è necessario creare e salvare la cartella nuova). Perciò se questo metodo non restituisse anche le cartelle vuote non si avrebbe la stessa libertà di modifica dell'albero di navigazione.

Invece dal lato del NavigationManagerAction è necessario che le cartelle vuote non vengano restituite in quanto lo strumento a cui si eroga l'albero di navigazione è uno strumento di visualizzazione delle Dashboard, è non servirebbe a nulla visualizzare cartelle che all'interno non contengono nessuna Dashboard.

3°parte: *Costruzione ricorsiva dell'albero di navigazione*

```
int i = 0;
boolean find = true;
while(find) {
        if(i>MAX_FOLDER_LEVEL) {
        throw new SystemException("Max number of Iteration reached in execute");
        }
        find = false;
        Set<FolderRootView> folderRootViewSet = new HashSet<FolderRootView>();
        folderRootViewSet.addAll(navigationMap.keySet());
        for(FolderRootView folderRootView:folderRootViewSet) {
                if(!folderRootView.isIncluse()) {
                        continue;
                }
                Folder folder = navigationMap.get(folderRootView);
                Folder parent = folder.getParent();
                FolderRootView parentFolderRootView = new

                FolderRootView(parent.getFolderId(), true);

                if (folder.getUpdateDate()!=null &&
                    folder.getUpdateDate().getTime()>lastUpdate){

                        lastUpdate=folder.getUpdateDate().getTime();

                } else if (folder.getCreateDate().getTime()>lastUpdate){

                        lastUpdate=folder.getCreateDate().getTime();
                }
                if(navigationMap.containsKey(parentFolderRootView)) {

                         navigationMap.get(parentFolderRootView).
                         getChildren().add(folder);
                         folderRootView.setInclude(false);
                } else {
                        if(parent.getId_parent()==null) {
                // Sono una cartella figlia di una cartella root non presente
                nella mappa
                                        parentFolderRootView.setInclude(false);
                                        parent.getChildren().add(folder);
                                        navigationMap.put(parentFolderRootView, parent);
                                        rootFolder.add(parent);
                        } else {
                                        parent.getChildren().add(folder);
                                        navigationMap.put(parentFolderRootView, parent);
                                        find = true;
                        }
                }
        }
        i++;
}
Collections.sort(rootFolder);
```

La terza e ultima parte del metodo buildnavigationtree di DashboardHandler è la parte ricorsiva.

Infatti come possiamo osservare viene utilizzato un ciclo while infinito che termina soltanto nel momento in cui si avvera una certa condizione ossia find diventa *false*.

In questo ciclo è stato anche inserita una condizione di terminazione alternativa, infatti per evitare un loop infinito è presente una condizione di terminazione del ciclo basata su un numero di iterazioni molto alto memorizzato nella costante MAX_FOLDER_LEVEL.

L'algoritmo iterativo procede in tale modo:

• La variabile folderRootViewSet viene popolata con la mappa navigationMap, creata nella prima parte.

Tale set viene creato come variabile d'appoggio del FolderRootView per la navigationMap, in questo modo è possibile ciclare sugli elementi e riuscire a modificare la mappa stessa contemporaneamente.

• Viene, perciò effettuato un ciclo su tale Set di tipo FolderRootView, ed in tale ciclo:

- Si controlla che l'elemento corretto sia una root, in questo caso si passa all'elemento successivo, tale passaggio si ripete finché non viene trovata una sottocartella.
- Per ogni sottocartella si controlla che il padre non sia già contenuto nella navigatioMap.

  Se questo è già presente nella mappa setto il flag della sottocartella a *false*, in quanto ho già tenuto traccia di tale elemento mediante il padre, in modo da non considerare più tale cartella nei prossimi cicli.

  - Se il padre non è presente nella mappa e questo è una root:
    aggiungo la sottocartella alla lista dei figli della cartella padre
    aggiungo il padre alla mappa
    setto il flag della vista a *false*, in modo da non considerare più tale cartella nei prossimi cicli.
  - Se il padre non è presente nella mappa e questo non è una root
    aggiungo la sottocartella alla lista dei figli della cartella padre
    aggiungo il padre alla mappa
    ma mantengo il flag della vista a *true*, in quanto in futuro sarà necessario trovare il padre del padre di tale sottocartella.

- Il ciclo prosegue in tale modo fino a che tutte le cartelle non hanno il flag della vista impostato a *false* e perciò fino a che l'intero albero non è stato creato.
- Il metodo termina ordinando la lista delle cartelle radice: rootFolder, ogni cartella di root al suo interno conterrà una lista di children che mediante la serializzazione in JSON creerà l'albero di navigazione.

4.3.3 Servizio di definizione Dashboard

Infine analizziamo il servizio che eroga la definizione di una Dashboard.
Tale definizione viene utilizzata dall'applicazione Mobile, in seguito alla selezione di una Dashboard dall'albero di navigazione.
L'URL che viene utilizzato per la chiamata a tale servizio è il seguente:

```
http://trueverit.quix.it/qery/api/dashBoard.action?auth_token=f9b09391-
30bc-4fe1-bbb0-ca6589b005ca&id=22b56110-d0e1-445f-964c-26888ed81515
```

L'applicazione mobile effettua questa chiamata per recuperare il numero di pagine, il layout e i contenuti della Dashboard: i Dataview, eseguirli e visualizzarne il risultato nella pagina.
Troviamo un esempio di definizione di una Dashboard della *figura 4.17*.
Da tale figura possiamo vedere che tale Dashboard ha una pagina con layout a 1 colonna e contenente due parti.
La prima parte, sulla sinistra contiene un Dataview di tipo GRID, perciò in formato tabellare.
Mentre la seconda parte, presente sulla destra della *figura 4.17* contiene un Dataview di tipo MAP, tale mappa è la stessa di cui è stata proposta la visualizzazione nel capitolo 1 riguardante il Viewer di Qery.

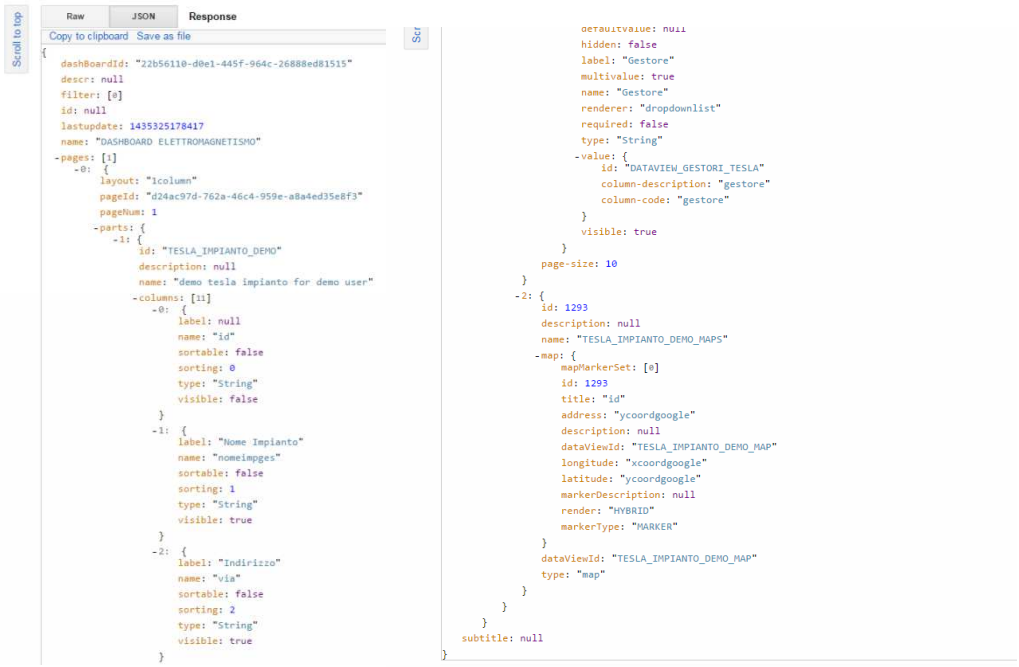

Figura 4.17: JSON di definizione Dashboard

La definizione delle Dashboard viene gestita dal DashboardManagerAction, la quale utilizza il metodo generateDashboardJson del DashBoardHandler.

Questo metodo non fa altro che recuperare le informazioni della Dashboard e dei Dataview contenuti all'interno di essa e costruirne un oggetto di tipo mappa contenente tutte le informazioni necessarie per l'esecuzione della Dashboard. Questo oggetto di tipo mappa verrà poi serializzato in JSON e restituito al client dal DashBoardManagerAction.

Il Metodo generateDashboardJson viene anche utilizzato per memorizzare la definizione della DashBoard in JSON su Database nella tabella apposita (DashBoardJson).

In questo modo è possibile recuperare la definizione ogni volta in cui la Dashboard viene visualizzata rendendo più veloce le future visualizzazioni.

4.4 Sviluppi Correnti e Futuri

In tale paragrafo accenneremo ad alcuni nuovi sviluppi che si stanno eseguendo attualmente e che verranno eseguiti nel prossimo futuro riguardanti l'applicazione Qery.
Inizialmente parleremo delle modifiche correnti che si stanno attuando, in particolare parleremo di un nuovo visualizzatore di Dashboard interno a Qery.
In seguito accenneremo a modifiche future che verranno effettuate sull'applicazione.

4.4.1 Sviluppi Correnti: Dashboard Viewer

Riproponiamo la *figura 4.3* del paragrafo 3.1, in quanto è stato accennato che avremo parlato dell'icona evidenziata ora dalla freccia blu.

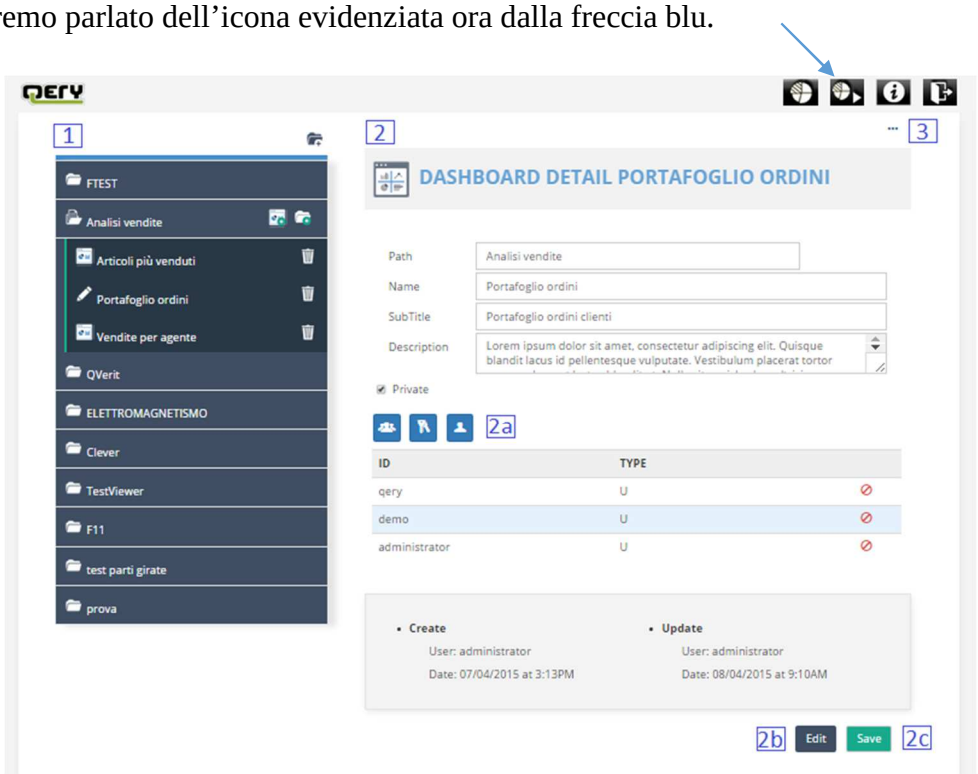

Figura 4.3: Pannello di controllo Dashboard

Mediante questa icona è possibile entrare nel visualizzatore di Dashboard interno all'applicazione.

Il Dashboard Viewer è nato per rendere necessario il test delle Dashboard che l'utente crea mediante il pannello di gestione delle Dashboard di QeryMobile e come strumento alternativo all'applicazione mobile per la visualizzazione delle Dashboard.

Come possiamo vedere dalla *figura 4.18* anche per questo visualizzatore è stato utilizzato il nostro albero di navigazione. L'unica differenza sta nel fatto che il menu in questo caso, è impostato su livelli di più pagine invece che essere come nei precedenti casi. Si è resa la navigazione più semplice implementando la barra di navigazione presente al di sopra dell'elenco delle cartelle ed evidenziata nel rettangolo azzurro.

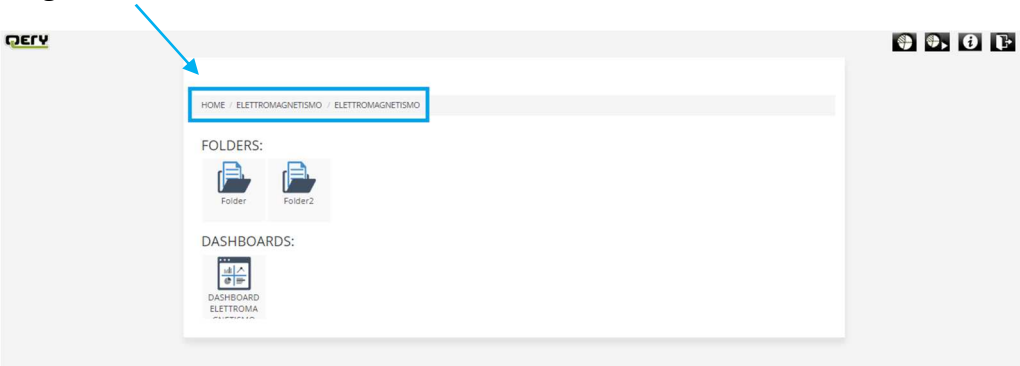

Figura 4.18: Barra di Navigazione

Infine cliccando sull'icona della Dashboard questa verrà eseguita e visualizzata, in *figura 4.19* troviamo la Dashboard di cui si è analizzata la definizione nel paragrafo precedente.

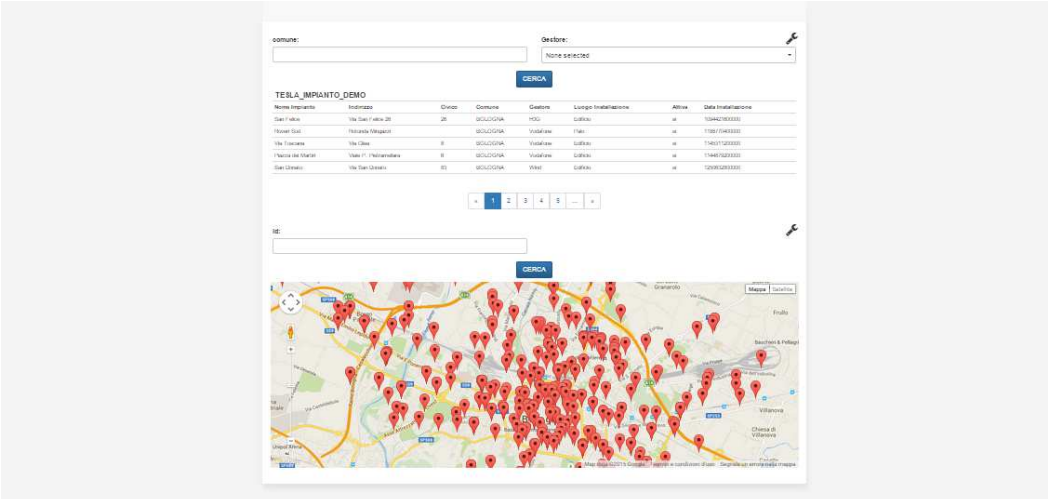

Figura 4.19: Esecuzione Dashboard

4.4.2 Sviluppi Futuri

Sono in programma nuovi sviluppi per l'applicazione Qery analizzata nel primo capitolo.

L'applicazione precedentemente non conteneva il framework Quix, utilizzava Struts1 e non utilizzava Spring.

Successivamente tali tecnologie sono state inserite ed ora si è iniziato a pensare ad eventuali modifiche lato client.

Tali modifiche riguarderanno l'interfaccia grafica dell'applicazione, verranno perciò modificate le pagine JSP e le tecnologie lato client.

Verrà inserito AngularJS ed una gestione dei Dataview simile a quella presentata nelle Dashboard.

L'interfaccia grafica cambierà del tutto, verrà infatti adottata una grafica più moderna e compatta, adatta anche ai dispositivi mobili.

Un primo esempio di come risulterà l'applicazione finale è mostrato nella *figura 4.20*.

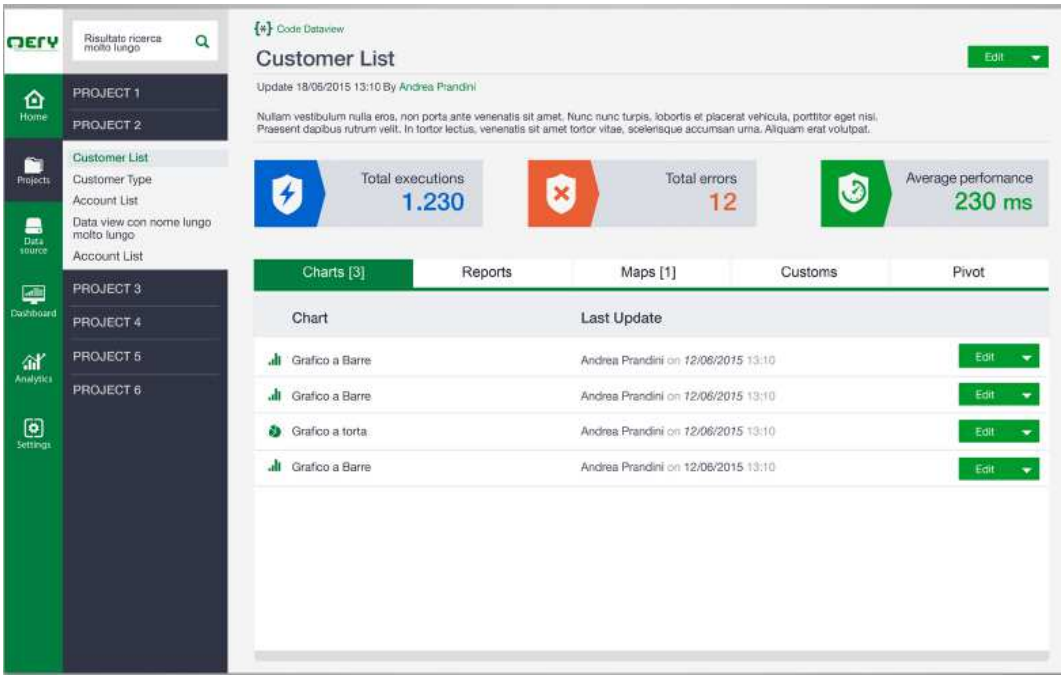

Figura 4.20: Il nuovo Qery

Bibliografia

[1] Struts2 in Action di Donald Brown, Chad Michael Davis e Scott Stanlick

[2] ServletJsp: Diapositive Quix s.r.l

[3] Spring in Action di Craig Walls

[4] Manuale Framework: Quix s.r.l

[5] JavaScript: The Definitive Guide di David Flanagan

[6] Wikipedia voce Javascript: https://it.wikipedia.org/wiki/JavaScript

[7] jQuery in Action Second Edition di Bear Bibeault,Yehuda Katz

[8] Documentazione ufficiale jQuery: https://jQuery.com

[9] Guida jQuery: http://www.html.it/guide/guida-jquery/

[10] Guida AngularJS : http://www.html.it/guide/guida-angularjs/

[11] Documentazione ufficiale AngularJS: https://angularjs.org/

[12] Documentazione ufficiale FlexJSON: http://flexjson.sourceforge.net/