|  |
| --- |
| **Programmer Guide**  ***Istat - WebClient***  Versione 1.0 del 29/07/2015 |

|  |  |  |
| --- | --- | --- |
| **Redatto da** | Proietti Valerio | IT Engineer |
| **Revisionato da** | <Nominativo> | <Qualifica> |
| **Committente** | | |
| **Dipartimento** |  | |
| **Direzione** |  | |
| **Servizio** |  | |

Sommario

[***Istat - WebClient*** 1](#_Toc427167036)

[1. Introduzione 3](#_Toc427167037)

[1. Action dell’applicazione 4](#_Toc427167038)

[2. Architettura Applicazione 5](#_Toc427167039)

[3. Utilizzo Widget 6](#_Toc427167040)

[4. Implementare nuovi Widget 9](#_Toc427167041)

[5. Query Utente 10](#_Toc427167042)

[6. Template dati 12](#_Toc427167043)

[7. Dashboard 14](#_Toc427167044)

# Introduzione

Il presente documento, descrive l'implementazione tecnica del software ISTAT.WebClient.

ISTAT.WebClient è un’applicazione web sviluppata in Asp.Net MVC con l’utilizzo di Javascript (frame work jQuery) si interfaccia a Web Services SDMX-RI.

# Action dell’applicazione

|  |  |
| --- | --- |
| Controller | Action |
| MainController | GetTree |
|  | GetMetadata |
|  | GetChartData |
|  | GetDefaultLayout |
|  | GetLayout |
|  | GetSpecificCodemap |
|  | GetCodemap |
|  | GetDataTable |
|  | GetData |
|  | IsCachingDataSet |
|  | ClearSession |
|  |  |
| DasboardController | GetDashboards |
|  | GetDashboardActive |
|  | PreviewDashboard |
|  | CreateDashboard |
|  | DeleteDashboard |
|  | CloseDashboard |
|  | ActiveDashboard |
|  | AddRow |
|  | UpdateRow |
|  | DeleteRow |
|  | AddWidget |
|  | UpdateWidget |
|  | DeleteWidget |
|  | AddWidgetText |
|  |  |
| ProfileController | Login |
|  | Logout |
|  | IsLogin |
|  | GetUserList |
|  | ModUserRole |
|  |  |
| QueryController | Get |
|  | GetSigleQuery |
|  | Add |
|  | Del |
|  |  |
| SettingsController | SetLocale |
|  | GetMessages |
|  | GetSettings |
|  | SetSettings |
|  | GetUserSettings |
|  | SetUserSettings |
|  |  |
| WebClientController | Contiene Le view (parti HTML) dell’aplicazione |
|  |  |

# Architettura Applicazione

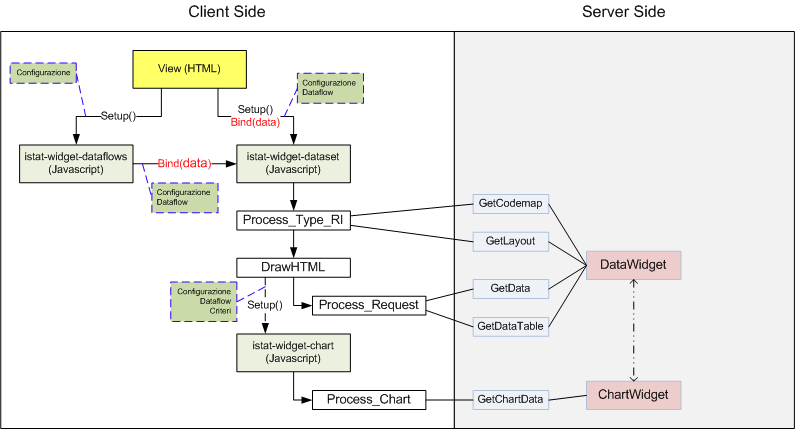


Figura - Comunicazione Client/Server tramite widget dinamici.

# Utilizzo Widget

I file javascript necessari per il corretto funzionamento dei widget sono:

<script src="<%=Url.Content("~/Scripts/istat-widget-manager.js")%>"></script>

<script src="<%=Url.Content("~/Scripts/istat-widget-dataflows.js")%>"></script>

<script src="<%=Url.Content("~/Scripts/istat-widget-dataset.j")%>s"></script>

<script src="<%=Url.Content("~/Scripts/istat-widget-text.j")%>s"></script>

<script src="<%=Url.Content("~/Scripts/istat-widget-chart.js")%>"></script>

<script src="<%=Url.Content("~/Scripts/istat-widget-dashboard.js")%>"></script>

<script src="<%=Url.Content("~/Scripts/istat-client.js")%>"></script>

Ovvero:

istat-widget-manager.js

Classe javascript che esegue l’inizializzazione dei componenti widget.

Per ogni widget ne legge gli attributi e li trasforma in parametri, passati al costruttore della relativa classe widget.

Funzione

* SetupWidgets (selector, locale, messages)

Parametri

* selector – (stringa) classe che identifica gli oggetti widget (al momento .dinamic-widget )
* locale – (stringa due lettere) es: en
* messages – (oggetto js) oggetto js che contiene le stringhe localizzate per le descrizioni

Utilizzo:

Viene richiamata dal client allo scaturire dell’evento ready del DOM, per ogni tag div trovato nella pagina con classe css uguale a selector istanzia la classe del widget corrispettivo contenuto nell’attributo data-widget-template

Widget ad ora implementati:

istat-widget-dataflows: Data un endpoint visualizza sotto forma di lista gerarchica tutti i CategoryScheme e i Dataflows annidati.

istat-widget-dataset: Dato un endpoint ed un dataflow visualizza in forma di tabella i dati del dataset.

istat-widget-text: Visualizza una stringa HTML.

istat-widget-chart: Dato un endpoint, un dataflow e dei costraint visualizza in formato grafico i dati del dataset.

istat-widget-dashboard: permette la visualizzazione delle Dashboard create.

Ogni Widget possiede un factory di partenza simile agli altri, tale struttura è contenuta nel file istat-widget-template.js

Un widget viene definito in come un tag HTML (div) avente classe css dinamic-widget e attributi data.

Es:

<div id="widget\_1 " class="dinamic-widget"

data-widget-template="treeview" // Testo

data-widget-stylecss="treeview-dataflows" // Testo

data-widget-data='{"ContainerMenuSelect":"#menu-left”}' // Testo format Json

data-widget-target="#main-table-dataset"> // Testo selettore css

</div>

Ogni widget a seconda del tipo di template indicato validerà ed utilizzerà in modo diverso i suoi attribute data e configuration.

Sono riportati di seguito i templati utilizzabili e la semantica dei loro attributi.

**Treeview**

data-widget-template="treeview"

data-widget-stylecss ="style\_css"

data-widget-target="selettore\_css"

data-widget-cofiguration=

[{

Title: string

active: bool,

DecimalSeparator: char(2)

EndPoint string

EndPointType: string

EndPointV20: string

EndPointSource: string

Domain: string,

EnableHTTPAuthentication: bool,

EnableProxy: bool,

Password: string,

Prefix: string,

ProxyPassword: string

ProxyServer: string

ProxyServerPort: string

ProxyUserName: string

UseSystemProxy: bool,

UserName: string

Wsdl: string

}]

data-widget-data=

{

ContainerMenuSelect: string

CurrentSelectItem: string

}

**Table**

data-widget-template="table"

data-widget-stylecss ="style\_css"

data-widget-target="selettore\_css"

data-widget-cofiguration=

{

dataflow: {

id: string,

agency: string

version: string

},

configuration: {

*// La stessa del template treeview*

},

layout: string,

criteria: string

}

data-widget-data=

{

query: string *// Json contenente la stessa struttura di configuration*

automation:bool

widgetId: int *// Valorizzato runtime indica se il widget è figlio di una dashboard*

}

**Chart**

data-widget-template="chart"

data-widget-stylecss ="style\_css"

data-widget-target="selettore\_css"

data-widget-cofiguration=

{

dataflow: {

id: string,

agency: string

version: string

},

configuration: {

La stessa del template treeview

},

layout: string,

criteria: string

}

data-widget-data=

{

criteria: string

chartType:bool

obsValue: string

dimensionAxe: string

customKey: string

customChartType: string

widgetId: int valorizzato runtime indica se il widget è figlio di una dashboard

}

[INSERIRE GRAFICO ARCHITETTURA INIZIALIZZAZIONE WIDGET]

# Implementare nuovi Widget

La creazione di nuovi widget è possibile utilizzando il file javascript istat-widget-template.js

Esso contiene il factory dei widget utizzato, è possibile rinominare la classe e salvare con nome diverso.

Per permettere la localizzazione e inizializzazione corretta del widget impostare una nuova scelta di template nel file istat-widget-manager.js che andra ad instaziare il nuovo widget al template corretto.

Es:

this.SetupWidgets = function (selector, locale, messages) {

… codice vario…

switch (\_template){

case "dashboard":

widgets["#" + \_idCSS] =

new WidgetDashBoard({

managerWidgets: \_manager,

idCSS: \_idCSS,

classCSS: \_classCSS,

template: \_template,

target: \_target,

configuration: \_configuration,

messages: \_messages,

locale: \_locale,

data: \_data

});

if (q == undefined) widgets["#" + \_idCSS].Setup();

break;

case "NEW\_TEMPLATE":

widgets["#" + \_idCSS] =

new Widget NEW\_TEMPLATE ({

managerWidgets: \_manager,

idCSS: \_idCSS,

classCSS: \_classCSS,

template: \_template,

target: \_target,

configuration: \_configuration,

messages: \_messages,

locale: \_locale,

data: \_data

});

widgets["#" + \_idCSS].Setup();

break;

Si dovranno implentare i metodi Setup e Bind, dove Setup viene richiamato dal SetupWidgets una volta, al caricamento della pagina e Bind viene richamato principalmente da altri widget che condividono l’elemento data-widget-target.

# Query Utente

L’Oggetto client.main.manager.widget contiene l’istanza della classe WidgetManager (istat-widget-manager.js), è possibile impostare query di partenza o configurazione automatica.

Nel codice seguente i widget treeview e table identificati usando il loro ID CSS vengono forzati ad effettuare una query specifica nei valori di ViewBag, oggetto asp MVC usato per trasportare dati dal controller alle view.

Il codice è preso dalla view index ed è utilizzato per caricare le query utente salvate.

<% if (ViewBag.query != null) {%>

<script type="text/javascript">

$(document).ready(function () {

client.main.manager.widget.query = {

"#main-treeview": {

dataflow: {

id: "<%= ViewBag.query.Dataflow.id %>",

agency: "<%= ViewBag.query.Dataflow.agency %>",

version: "<%= ViewBag.query.Dataflow.version %>"

}

},

"#main-table-dataset": {

dataflow: {

id: "<%= ViewBag.query.Dataflow.id %>",

agency: "<%= ViewBag.query.Dataflow.agency %>",

version: "<%= ViewBag.query.Dataflow.version %>",

configuration: {

IDNode: "<%= (ViewBag.query.Configuration.IDNode!=null)?ViewBag.query.Configuration.IDNode:"" %>",

Locale: "<%= (ViewBag.query.Configuration.Locale!=null)?ViewBag.query.Configuration.Locale:"" %>",

Title: "<%= (ViewBag.query.Configuration.Title!=null)?ViewBag.query.Configuration.Title:"" %>",

EndPoint: "<%= (ViewBag.query.Configuration.EndPoint!=null)?ViewBag.query.Configuration.EndPoint:"" %>",

EndPointType: "<%= (ViewBag.query.Configuration.EndPointType!=null)?ViewBag.query.Configuration.EndPointType:""%>",

EndPointV20: "<%= (ViewBag.query.Configuration.EndPointV20!=null)?ViewBag.query.Configuration.EndPointV20:"" %>",

EndPointSource: "<%= (ViewBag.query.Configuration.EndPointSource!=null)?ViewBag.query.Configuration.EndPointSource:"" %>",

DecimalSeparator: "<%= (ViewBag.query.Configuration.DecimalSeparator!=null)?ViewBag.query.Configuration.DecimalSeparator:"" %>"

}

},

key\_time\_dimension: "TIME\_PERIOD",

layout: {

axis\_x: [

<% foreach (var criterio in ViewBag.query.Layout.axis\_x)

{ %>

"<%= criterio %>",

<% } %>],

axis\_y: [

<% foreach (var criterio in ViewBag.query.Layout.axis\_y)

{ %>

"<%= criterio %>",

<% } %>],

axis\_z: [

<% foreach (var criterio in ViewBag.query.Layout.axis\_z)

{ %>

"<%= criterio %>",

<% } %>],

},

criteria: {

<% foreach (var criterio in ViewBag.query.Criteria){ %>

"<%= criterio.Key %>": [

<% foreach (var value in criterio.Value){ %>

"<%= value %>",

<% } %>

],

<% } %>

}

}

};

});

</script>

<% } %>

# Template dati

Un utente con privilegi di amministratore può definire dei “Template sui dati”.

Ovvero:

Eseguita una query su un determinato dataflow un amministratore può salvare questa come query di default, ciò significa che un altro utente qualora richiedesse il dataflow su qui è applicato un template vedrà direttamente i dati in forma tabellare e potrà tornare alla selezione dei criteri su richiesta.

La lettura e il salvataggio delle query avviene lato server poiché alla richiesta di salvataggio il client invia al server informazioni formato json che descrivono il template:

Oggetto Json passato al metodo Template/Add

var data = {

UserCode: cod,

Template: {

Title: title, *// Titolo DF*

DataFlow: dataflow, *// Oggetto Identificable DF*

Layout: layout, *// Oggetto layout*

Criteria: filters, *// Dictonary contente chiave=>valori costraint*

Configuration: configuration, *// Oggetto configurazione*

HideDimension: cbox\_sel, *// Array che contiente gli Id delle dimensioni*

BlockXAxe: (inArray('x', block\_sel) < 0)?false:true, *// Vero o Falso se x è contenuta in block\_sel*

BlockYAxe: (inArray('y', block\_sel) < 0)?false:true, *// Vero o Falso se y è contenuta in block\_sel*

BlockZAxe: (inArray('z', block\_sel) < 0) ? false : true, *// Vero o Falso se z è contenuta in block\_sel*

EnableCriteria: isCri, *// Vero o Falso se il checkbox è checcato*

EnableDecimal: isDec, *// Vero o Falso se il checkbox è checcato*

EnableVaration: isVar, *// Vero o Falso se il checkbox è checcato*

}

}

Metodo lato server Template/Add c#

public ActionResult Add()

{

try

{

*// Lettura parametri tramite cast di tipi*

GetTemplateObject PostDataArrived =CS.GetPostData<GetTemplateObject>(this.Request);

*// Lettura connection string*

ConnectionStringSettings connectionStringSetting=

ConfigurationManager.ConnectionStrings["ISTATWebClientConnection"];

*// Check sulla connection string*

if (connectionStringSetting == null || string.IsNullOrEmpty(connectionStringSetting.ConnectionString))

throw new Exception("ConnectionString not set");

*// Inizializzazione Service dei template*

TemplateWidget qw = new TemplateWidget(connectionStringSetting.ConnectionString);

*// Ritorna Json al client con il risultato della chiamata TemplateWidget.Add(GetTemplateObject* arg)

return CS.ReturnForJQuery(qw.Add(PostDataArrived));

}

catch (Exception ex)

{

*// Ritorna Json al client con messaggio di errore*

return CS.ReturnForJQuery(ex.Message);

}

}

Il caricamento dei template avviene lato server, in quanto ogni volta che un utente richiede un dataflow il sistema esegue una query su ISTAT.WebClient.DB nella tabella template e cerca un record corrispondente alla chiave DF+Endpoint. Se viene trovato un template questo valorizzerà due proprietà dell’oggetto tornato al client: costraint( oggetto con la stessa struttura di Template.Criteria contiene i costraint imostati) e hideDimension (lista di concetti o dimensioni che non si vogliono mostrare all’utente).

Metodo public SessionImplObject GetCodemap(ConnectionStringSettings connectionStringSetting) su CodemapWidget.cs

*// Ricerca record in Template con id+agency+version+endpoint*

System.Data.SqlClient.SqlConnection Sqlconn = new System.Data.SqlClient.SqlConnection(connectionStringSetting.ConnectionString);

Sqlconn.Open();

string sqlquery = string.Format("Select \* from Template where [tmplKey]='{0}'",

new System.Web.Script.Serialization.JavaScriptSerializer().Serialize(

CodemapObj.Dataflow.id + "+" + CodemapObj.Dataflow.agency + "+" + CodemapObj.Dataflow.version + "+" + CodemapObj.Configuration.EndPoint).Replace("'", "''"));

using (System.Data.SqlClient.SqlCommand comm = new System.Data.SqlClient.SqlCommand(sqlquery, Sqlconn))

{

var reader = comm.ExecuteReader();

if (reader.Read())

{

costraint = reader.GetString(reader.GetOrdinal("Criteria"));

hideDimension = reader.GetString(reader.GetOrdinal("HideDimension"));

}

}

Sqlconn.Close();

*// Oggetto restituito se costraint e hideDimension sono valorizzati si ha un template*

CodemapResponseObject codemapret = new CodemapResponseObject()

{

codemap = ParseCodelist(ConceptCodelists),

costraint = (Dictionary<string, List<string>>) new JavaScriptSerializer().Deserialize(

costraint, typeof(Dictionary<string, List<string>>)),

hideDimension = (List<string>)new JavaScriptSerializer().Deserialize(

hideDimension, typeof(List<string>)),

key\_time\_dimension = kf.TimeDimension.Id,

freq\_dimension = kf.FrequencyDimension.Id,

dataflow = new MaintenableObj()

{

id = df.Id,

agency = df.AgencyId,

version = df.Version,

name = TextTypeHelper.GetText(df.Names, this.CodemapObj.Configuration.Locale),

description = TextTypeHelper.GetText(df.Descriptions,this.CodemapObj.Configuration.Locale)

}

};

# Dashboard

Per dashboard all’interno del contesto ISTAT Web Client si intendo un contenitore di widget ognuno indipendente dall’altro e relazionato alla dashboard solo nella sua posizione (riga/colonna).

Le dashboard e i widget associati vengono salvati sul supporto ISTAT.WebClient.DB

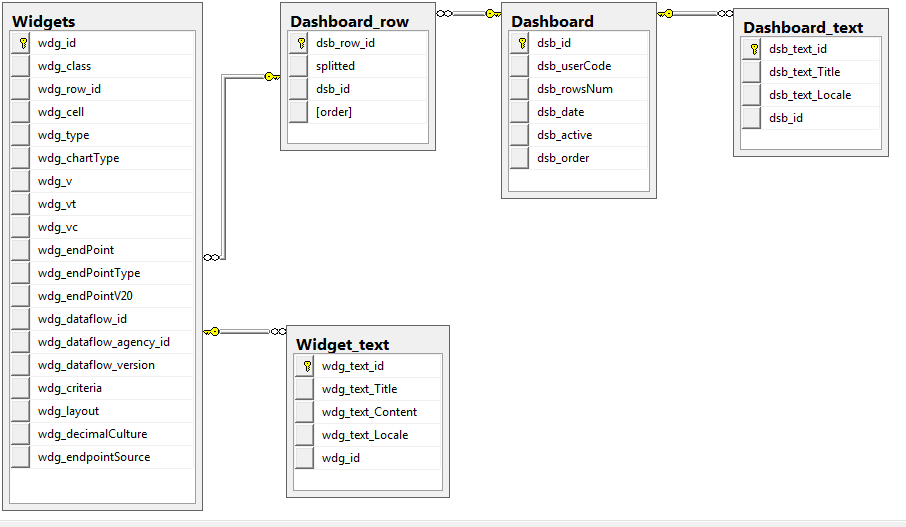


Figura - Schema tabelle interessate gestione Dashboards.

Le dashboard vengono salvate nel db in modalità “Real Time”, ad ogni fine operazione sostanziale viene richiamato il corrispettivo metodo di save su DashboardController, le azioni per qui e’ possibile definire un salvataggio di informazioni autonome possono essere riassunte in:

* Creazione Nuova Dashboard => Dashboard/CreateDashboard
* Eliminazione Dashboard => Dashboard/DeleteDashboard
* Chiusura Dashboard => Dashboard/CloseDashboard
* Aggiunta riga => Dashboard/AddRow
* Eliminazione riga => Dashboard/DeleteRow
* Modifica numero colonne => Dashboard/UpdateRow
* Aggiunta riga => Dashboard/AddRow
* Eliminazione riga => Dashboard/DeleteRow
* Modifica numero colonne (split) => Dashboard/UpdateRow
* Aggiunta Nuovo Widget => Dashboard/AddWidget
* Modifica configurazione Widget => Dashboard/UpdateWidget
* Eliminazione Widget => Dashboard/DeleteWidget

Il file dashboardElements.js contiene le funzioni che costruiscono l’editor delle dashboard.

Il file istat-widget-dashboard (compliant con factory widget) definisce il widget che visualizza le dashboard ad ora utilizzato nella pagina Index.

[INSERIRE GRAFICO ARCHITETTURA SAVATAGGIO DASHBOARD]

**dashboardElements.js**

Allo scaturire dell’evento ready di javascript ottenuto tramite jQuery con la sintassi:

$(document).ready(function () {

*// Code for ready event*

});

viene richiesto al server la lista delle dashboard e vengono rappresentate tramite la func. ReloadListDashboards(dest), che disegna una lista di dashboard, ognuna con due pulsanti per eliminarla o modificarla.

Tramite il bottone “Create” viene chiamata la func. AddDashboard(dest) che richiederà il titolo della dashboard prima di poter continuare.

Al click su “Continua” verrà inviata una richiesta al server per creare un record nella tabella Dashboard e salvare registrare di già la sua presenza (di default vengono inserite dashboard disattivate).

La funzione DrawTable(objDashBoard, dashBoardId) viene utilizzata sia per bashboard nuove sia per dashboard in modifica. Se il parametro objDashBoard contiene già delle righe e dei widget siamo in modifica e quindi occorre riportare le informazioni contenute, se objDashBoard e’ null, verrà presentata una maschera vuota da poter personalizzare.