Norris

Node Real-time Intelligence

the only framework that eats pie charts

CoffeeStrap

CoffeeStrap

1 Introduzione

Negli ultimi anni l'universo aziendale sta iniziando ad intravedere negli strumenti open source una risorsa estremamente preziosa, sia da un punto di vista economico che da un punto di vista tecnologico. Questa convergenza è dettata in primo luogo dalla necessità di stare al passo con i ritmi e la flessibilità richiesti da esigenze di prototipazione del prodotto, centrate sulla minimizzazione delle risorse da un lato e la massimizzazione dell'efficienza dall'altro.

In particolare, la notevole mole di informazioni rese disponibili dalla velocità di rete e dalla potenza di calcolo dei nuovi sistemi oggi ha impegnato progressivamente gli sviluppatori a rivedere e rivoluzionare i tradizionali paradigmi di analisi dei dati a supporto delle attività di *business*. Questa rivoluzione tecnologica ha permesso di creare strumenti immediatamente utilizzati dagli esperti di dominio all'interno dell'azienda.

Il processo di decisione da parte di queste funzioni aziendali dev'essere fortemente supportato da dati che descrivano il più precisamente possibile il funzionamento dell'azienda in tutti i suoi aspetti. Mancando tali figure di particolare curriculum tecnico, esse hanno bisogno di utilizzare strumenti facilmente fruibili e leggere i dati in output in maniera semplice ed immediata, tramite l'utilizzo di rappresentazioni grafiche quali grafici e tabelle. L'insieme degli strumenti sopra descritti fa parte di un processo decisionale più ampio, detto *Real-Time Business Intelligence*.

Il Real-time *business intelligence* (*RTBI*) è un processo di analisi delle informazioni derivanti da operazioni di *business* guidato da eventi. In questo contesto *real-time* significa che il processamento dell'informazione avviene in un intervallo di tempo che varia dai millisecondi a pochi secondi successivamente allo scatenarsi dell'evento.

2 Real-time Business Intelligence

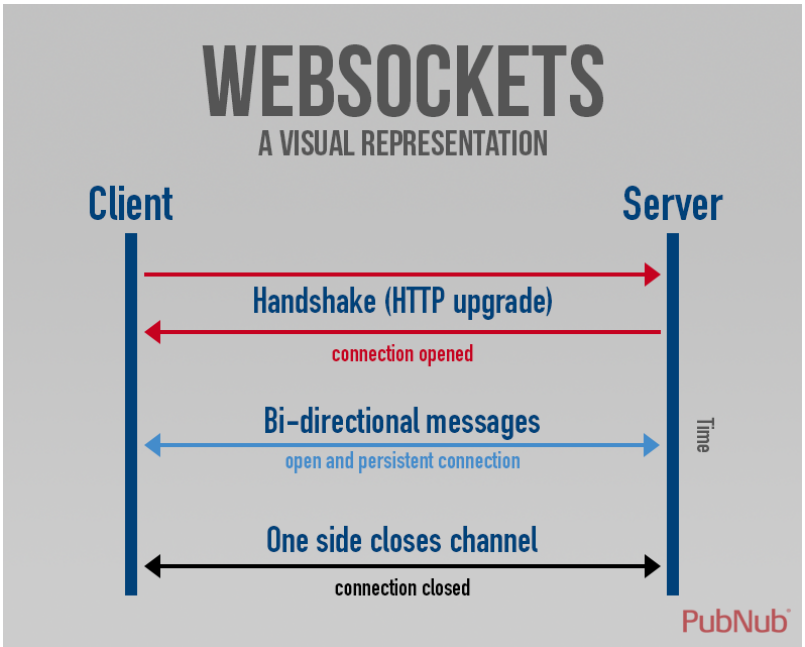
Lo scopo del progetto è di produrre un *framework* per lo *stack* tecnologico formato da Node.js, Express.js e Socket.io in grado di generare grafici i cui dati sono letti da sorgenti arbitrarie. Il fruitore finale dei grafici è l'esperto di dominio.

Norris è un *framework* che permette di raccogliere dati provenienti da sorgenti arbitrarie e visualizzarli come grafici in modo semplice e veloce. La veste grafica e i dati di ciascun grafico sono configurabili programmaticamente usando le API fornite da Norris. Il *framework* mette a disposizione funzioni di aggiornamento dei grafici lato server tramite tecnologia *WebSocket*[1].

WebSocket è una tecnologia web che fornisce canali di comunicazione bidirezionali simultanei attraverso una singola connessione TCP. Le API e il protocollo di comunicazione

[1] http://en.wikipedia.org/wiki/WebSocket

WebSocket sono stati standardizzati dal W3C e da IETF, ed è una tecnologia implementata dai principali *browser*[2].



PubNub, What are websockets: http://www.pubnub.com/blog/what-are-websockets

2.1 Descrizione Grafici

Lo sviluppatore, utilizzando Norris come *framework* di lavoro, deve essere in grado di produrre quattro tipi di rappresentazioni dei dati (per brevità detti grafici):

1. *Bar chart;*
2. *Line chart;*
3. *Map chart;*
4. Stampa di dati in formato tabellare.

Ciascuno dei grafici può essere aggiornato in conseguenza di una alimentazione real-time di nuovi dati. L'aggiornamento può avvenire in tre modi: aggiornamento in *place,* aggiornamento *stream,* aggiornamento *movie*.

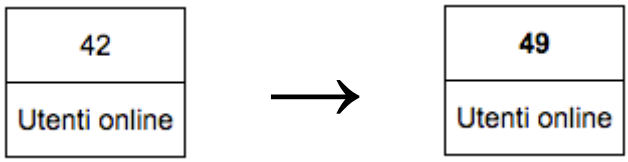
[2] http://en.wikipedia.org/wiki/WebSocket#Browser_support

Tipi di aggiornamento

Aggiornamento in *place*

L'aggiornamento in *place* sostituisce valori già presenti con valori nuovi.

Esempio: aggiornamento di un contatore. Usando una tabella formata da due righe ed una colonna viene creata una componente grafica che per visualizzare un contatore di utenti online. L'aggiornamento della componente non aggiunge nuovi elementi, ma cambia il valore degli elementi presenti.



Aggiornamento *stream*

L'aggiornamento in stream aggiunge nuovi valori al grafico. Il grafico mantiene la visualizzazione dei dati già presenti aggiungendo nuove rappresentazioni grafiche dei nuovi dati aggiunti.

Esempio: una lista di numeri. Ogni nuovo numero aggiunto alla lista viene aggiunto a quelli già presenti.

Timestamp	IdMezzo	Lat	Long	Capolinea	Stato porte
Oct 06 2014 12.44	833	11.884307861328	45.380382537842	Capolinea De Tagg_cimitero	1
Oct 06 2014 12.44	803	11.875713348389	45.387321472168	Capolinea De Lazara	0

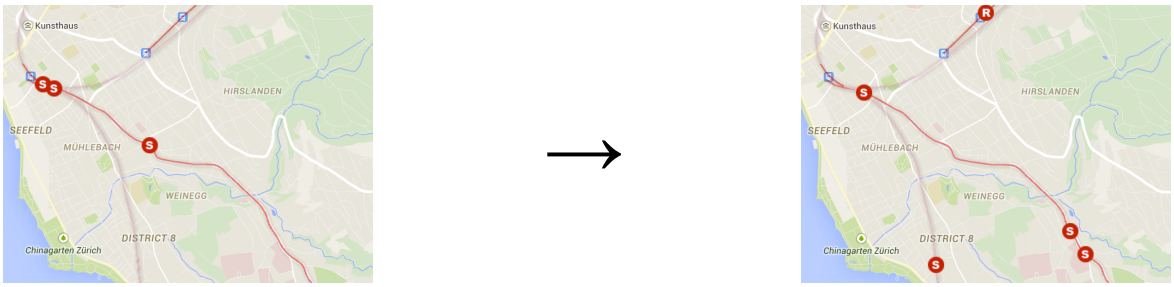
Timestamp	IdMezzo	Lat	Long	Capolinea	Stato porte
Oct 06 2014 12.46	**833**	**11.878223419189**	**45.410736083984**	**Capolinea De Tagg_cimitero**	**0**
Oct 06 2014 12.44	833	11.884307861328	45.380382537842	Capolinea De Tagg_cimitero	1
Oct 06 2014 12.44	803	11.875713348389	45.387321472168	Capolinea De Lazara	0

Aggiornamento *movie*

Il metodo di aggiornamento *movie*, combina i due metodi in *place* e *stream*. Questo aggiornamento gestisce la transizione della rappresentazione del grafico in due momenti successivi. Analogamente ai film, i quali sono composti da sequenze di fotogrammi. Questo tipo di aggiornamento permette di:

- sostituire i valori già rappresentati;
- aggiungere nuovi valori al grafico;
- togliere valori rappresentati nel grafico.

Esempio: una mappa che mostra i percorsi degli autobus che stanno girando nella città e la loro posizione lungo la tratta in tempo reale.



La tabella seguente indica quali grafici devono implementare un aggiornamento di tipo in *place*, di tipo *stream*, o di tipo *movie*.

Grafico	**Tipo di aggiornamento**
Bar Chart	In *place*
Line Chart	In *place*, *stream*
Map Chart	*In place, movie*
Table	In *place*, *stream*

Tipi di grafico

Bar chart

Il grafico *bar chart* è il tipico grafico a barre (i.e.: http://en.wikipedia.org/wiki/Bar_chart). Per ciascun grafico di questo tipo lo sviluppatore può avere almeno i seguenti parametri di configurazione:

- Titolo del grafico;
- Nomi degli assi;
- Colore di ciascun set di dati;
- Formato di stampa dei valori;
- Linee della griglia visualizzate/nascoste;
- Legenda visualizzata/nascosta;
- Posizione della legenda;
- Orientamento delle barre (orizzontali o verticali).

Line chart

Il grafico *line chart* è il tipico grafico a linea (i.e.: http://en.wikipedia.org/wiki/Line_chart). Per ciascun grafico di questo tipo lo sviluppatore può avere almeno i seguenti parametri di configurazione:

- Titolo del grafico;
- Nomi degli assi;
- Colore di ciascun set di dati;
- Formato di stampa dei valori;
- Linee della griglia visualizzate/nascoste;
- Legenda visualizzata/nascosta;
- Posizione della legenda.

Map chart

Il grafico *map chart* è una mappa geografica con dati rappresentati in sovraimpressione, (i.e.: http://stn.spotfire.com/spotfire_client_help/map/map_what_is_a_map_chart.htm). I dati possono essere rappresentati come forme geometriche colorate, come icone o come testo. Generalmente le forme geometriche e le icone possono essere selezionabili, cosi come la mappa può essere ingrandita o rimpicciolita a discrezione dell'utente. Le rappresentazioni dei dati cambiano contestualmente alla porzione di mappa visualizzata.
Per ciascun grafico di questo tipo lo sviluppatore può avere almeno i seguenti parametri di configurazione:

- Titolo del grafico;
- Colore di ciascun set di dati;
- Formato di stampa dei valori;
- Legenda visualizzata/nascosta;
- Posizione della legenda;
- Punto centrale della mappa;
- Dimensioni, espresse in larghezza e altezza, dell'area visualizzata dalla mappa. L' unità di misura delle dimensione è espresso in metri.

Table

La rappresentazione dei dati a tabella è una comune griglia sulla quale vengono visualizzati i valori. Per ciascun grafico di questo tipo lo sviluppatore può avere almeno i seguenti parametri di configurazione:

- Titolo del grafico;
- *Header* di ciascuna colonna;
- Colore del testo all'interno di ogni singola cella;
- Colore dello sfondo di ogni singola cella;
- Formato di stampa dei valori all'interno delle celle;
- Linee della tabella visualizzata/nascosta;
- Solo in *place*: possibilità di ordinare gli elementi di una colonna: si/no (solo per modalita' di aggiornamento in *place*);
- Solo *stream*: posizione di aggiunta dei nuovi dati in testa o in coda;
- Massimo numero di elementi visualizzati contemporaneamente.

2.2 Composizione di grafici in pagine

Più grafici possono composti nella stessa pagina. Ciascun grafico viene aggiornato indipendentemente dagli altri. Il programmatore può decidere la disposizione dei grafici nella pagina. Per determinare la visualizzazione dei grafici nella pagina il programmatore configura almeno i seguenti parametri di configurazione:

- massimo numero di grafici su una riga e/o colonna;
- titolo della pagina;
- dimensioni massime di ciascuna grafico.

3 Requisiti

3.1 Requisiti minimi

Di seguito vengono elencati i requisiti che il prodotto deve obbligatoriamente soddisfare:

1. Sviluppo prototipale di Norris per la generazione rapida di client web per la visualizzazione di grafici aggiornabili in tempo reale. Si richiede l'implementazione delle funzionalità relative ai grafici *Line Chart, Bar Chart, Map Chart, Table,* con aggiornamento in *place,* stream e *movie*, come descritto al punto 2.1 del presente documento.
2. Caso d'uso. Implementazione di una *dashboard* di visualizzazione in tempo reale degli spostamenti degli autobus nel comune di Padova. La *dashboard* sarà un progetto Node.js che utilizzerà Norris per visualizzare informazioni riguardanti le linee degli autobus di Padova con aggiornamento in tempo reale. La sorgente dati è fornita dal sito dell'APS Holding (http://www.apsholding.it/) effettuando una chiamata HTTP con i seguenti parametri:

Indirizzo	http://www.apsholding.it/index.php/informazioni/dov-e-il-mezzo-pubblico-in-tempo-reale?option=com_mappeaps&view=posmezzi&format=raw
Verbo HTTP	POST
Payload	l='NUMERO_LINEA'
Formato *payload*	x-www-form-urlencoded
Esempio	https://www.getpostman.com/collections/b172bde8088685a57489

La *dashboard* deve visualizzare almeno le seguenti informazioni:

1. Posizione degli autobus di una singola linea
2. Numero di autobus attivi per una singola linea

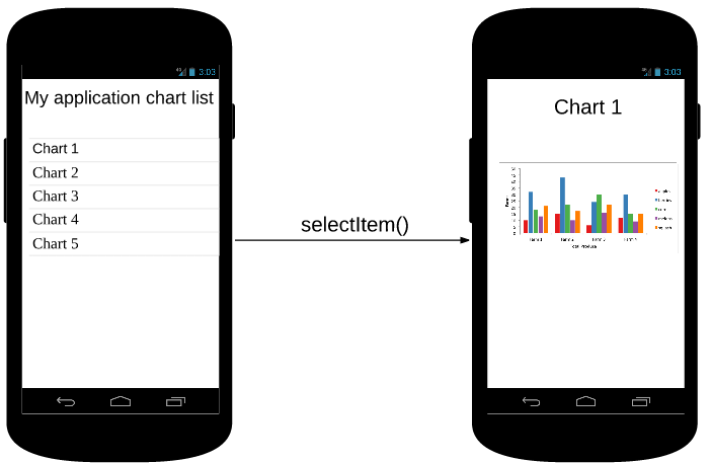
3. Lo stack tecnologico richiesto include:
   a. Node.js (http://nodejs.org): per la realizzazione della libreria come piattaforma di sviluppo.
   b. Express (http://expressjs.com): per la realizzazione dell'infrastruttura web. Il framework dovrà essere utilizzato come middleware di express.
   c. Socket.io (http://socket.io): per la componente WebSocket che realizza le notifiche *push*.

   d. Non è posto alcun vincolo circa la scelta delle componenti grafiche di front-end. Tuttavia si consiglia fortemente l'uso di Angular.js.
4. L'uso del framework dovra essere compatibile con l'utilizzo standard dei *middleware* di Express, versione 4.x (http://expressjs.com/4x/api.html#middleware).
5. Possibilità di effettuare il *deployment* su Heroku (https://www.heroku.com), servizio di cloud di tipo *platform as a service* (PaaS). Tramite Heroku è possibile rendere disponibile *on-line* applicazioni web scritte in vari framework (Node.js, Ruby on Rails e Play). Heroku si rende responsabile di ospitare il codice (con relative dipendenze/librerie) adottando un meccanismo di *deployment* basato su git (http://git-scm.com, software di versionamento distribuito).
6. Pubblicazione del progetto su GitHub o Bitbucket e utilizzo delle issue per la segnalazione di bug.
7. Compatibilità con la versione 38.0.X o superiori di Chrome e la versione 32.X o superiori di Firefox.

3.2 Requisiti opzionali

Creazione di un'applicazione Android che per visualizzare i grafici messi a disposizione da una specifica istanza di Norris.

1. Tale applicazione dovrà utilizzare le API di Norris per visualizzare ciascuno di questi grafici in una pagina separata (in ambito Android si utilizzerà il termine Activity o Fragment);
2. Le funzioni di aggiornamento devono essere disponibile nella versione Android del progetto allo stesso modo della versione web. A tal scopo è consigliato utilizzare il client Java per Socket.IO (https://github.com/nkzawa/socket.io-client.java);
3. L'indirizzo del server ospitante Norris, deve poter essere configurabile usando l'applicazione. In tal modo la singola applicazione può essere utilizzata con diverse istanze di Norris, previa modifica del parametro di configurazione;
4. Un esempio di struttura di applicazione è dato dal seguente *mockup:*



3.3 Variazione dei requisiti

Non sono ammesse variazioni se non a evidente miglioramento di quanto qui specificato. Non è esclusa la comunicazione, da parte del committente, di variazioni ai requisiti sia precedentemente alla consegna delle offerte che durante la realizzazione del sistema.

3.4 Documentazione

La consegna del sistema dovrà essere accompagnata dai necessari manuali d'uso per gli utilizzatori del framework e da ogni altra documentazione tecnica necessaria per l'utilizzo del prodotto da parte del personale operatore del committente. È gradita la versione inglese.

3.5 Garanzia e manutenzione

Il fornitore dovrà dimostrare in sede di collaudo (Revisione di Accettazione) il funzionamento corretto del sistema. L'eliminazione dei difetti e delle non conformità eventualmente emersi in sede di collaudo sono a totale carico del fornitore.

3.6 Rinvio

L'azienda CoffeeStrap è interessata a questo progetto come dimostrazione della fattibilità dell'obiettivo utilizzando le tecnologie indicate. Il progetto verrà distribuito con licenza MIT con copyright degli studenti e menzione del contributo di CoffeeStrap nel *repository* (i.e. sezione *credits* nel *readme*). Per tutto quanto non previsto nel presente capitolato, sono applicabili le disposizioni contenute nelle legge e nei collegati per la gestione degli appalti pubblici.

4 Proponente

CoffeeStrap (http://www.coffeestrap.com) è una piattaforma online che offre a chiunque desideri apprendere un linguaggio straniero la possibilità di entrare in contatto con studenti e insegnanti attraverso sessioni di apprendimento online basate su videochat.

L'azienda ha come base operativa Amsterdam (ex alumni Rockstart 2014, http://rockstart.com/accelerator/2014/03/04/coffeestrap/) e base strategica nell'ecosistema *startup* di San Francisco.

CoffeeStrap è rappresentata da:
Milo Ertola;
Mahesh Casiraghi;
Alessandro Maccagnan (contatto di riferimento per Norris) - alessandro@coffeestrap.com

5 Appendice

5.1 Esempi d'uso

Bar chart con aggiornamento in *place*

Per usare il framework sarà necessario preventivamente installare il pacchetto tramite il gestore dei pacchetti di node (npm [Node Packaged Modules], https://www.npmjs.org/).

Per farlo è sufficiente aggiungere come dipendenza il pacchetto 'norris' nel file 'package.json' e lanciare l'aggiornamento dei pacchetti tramite 'npm install'. Il nome del pacchetto 'norris' è utilizzato solo a titolo esplicativo.

package.json

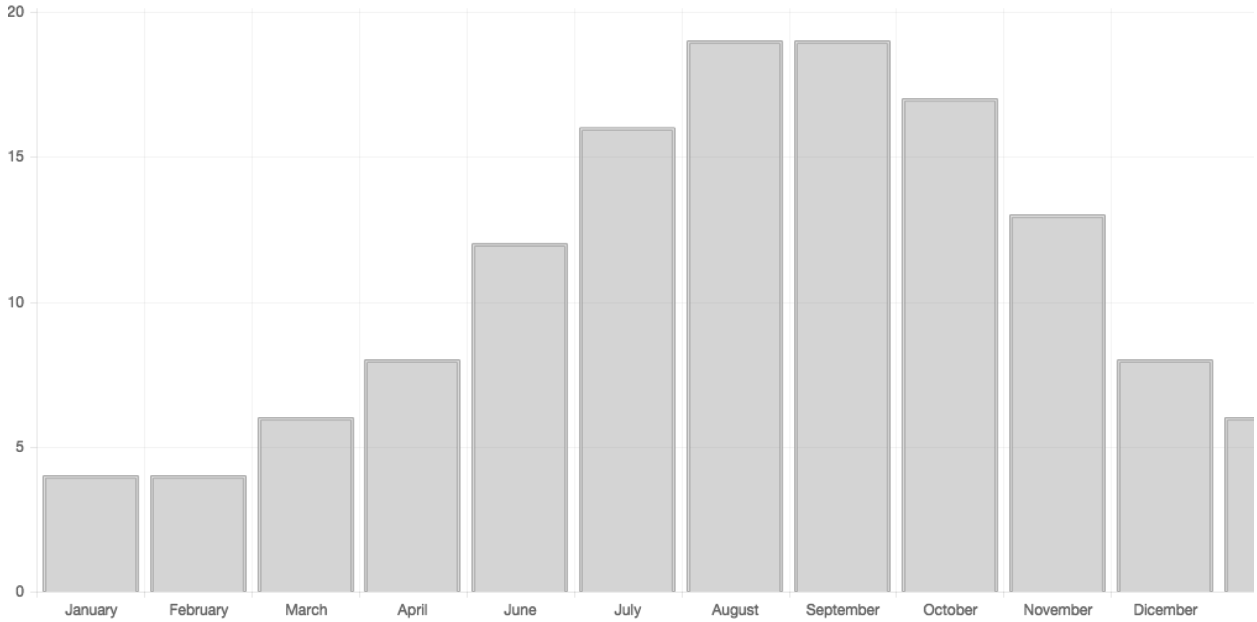
```
{
  "name": "norris",
  "version": "1.0.0",
  "description": "",
  "dependencies": {
    "express": "^4.9.5",
    "norris"   : "latest"
  }
}
```

L'utilizzo della libreria sara' programmatico. Un esempio minimale e' dato di seguito.
In questo esempio i dati sono dichiarati staticamente. La gestione dei dati è a carico dell'utilizzatore del framework. Allo sviluppatore viene data completa libertà riguardo l'origine e il recupero dei dati. Viene invece vincolato il formato dei dati in ingresso al framework.

index.js

```
/*jshint node: true */
'use strict';

var express = require('express');
var app     = express();

var norris    = require('norris');
var Page      = norris.Page;
var BarChart  = norris.BarChart;
var page = new Page();

var barChartDataConnectorConnector = function(done){
    var labels = [
      'January',
      'February',
      'March',
      'April',
      'June',
      'July',
      'August',
      'September',
      'October',
      'November',
      'Dicember'
    ];
    var barChartData = [4,4,6,8,12,16,19,19,17,13,8,6 ];
    return done(null, barChartData);
};

var barChart = new BarChart(barChartDataConnector);
page.addElement(barChart);

app.use('/barchart', page);
app.listen(3000);
```

Un esempio di output per l'utilizzo precedente può essere il seguente. Puntando il *browser* all'indirizzo indicato (http://localhost:3000/barchart) si avrà la seguente rappresentazione:



Volendo aggiornare il grafico, in particolare una barra dell'istrogramma, lo sviluppatore dovrà usare la funzione di aggiornamento fornita dal *framework*. Con riferimento all'esempio precedente, un esempio è il seguente:

```
barChart.updateValue({bar: 5, value:2});
```

A seguito di tale chiamata il grafico verrà cosi aggiornato:

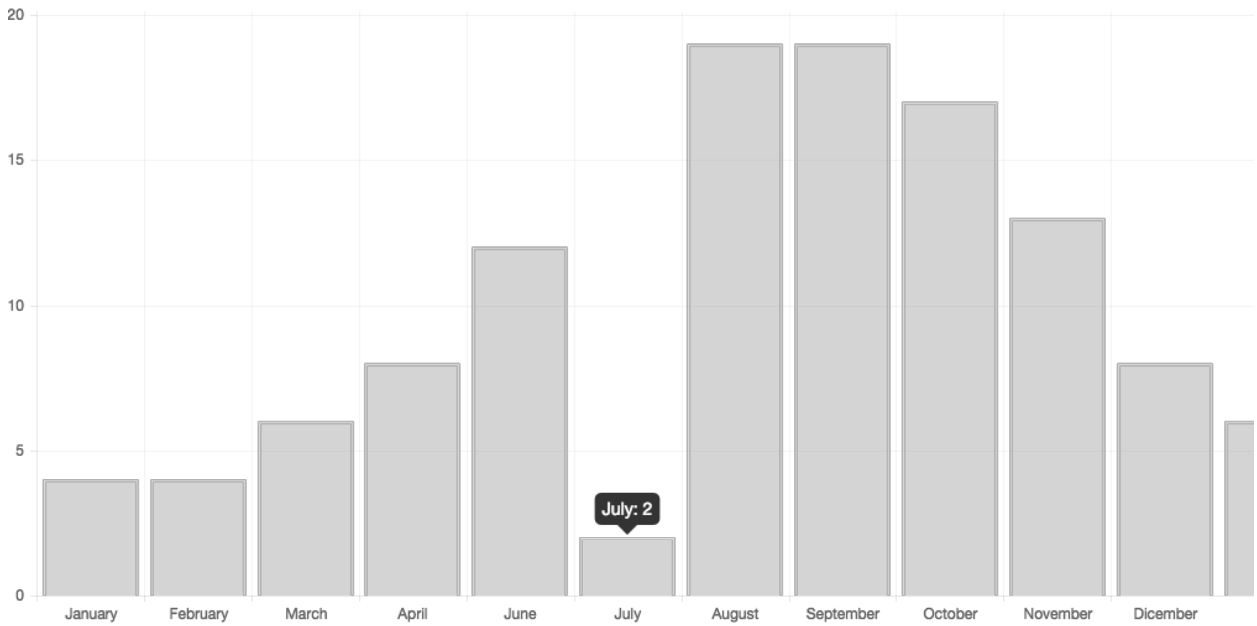


Table con aggiornamento in place

Nel seguente esempio abbiamo invece una visualizzazione a tabella con aggiornamento in *place*.

index.js

```
/*jshint node: true */
'use strict';

var express = require('express');
var app     = express();

var norris  = require('norris');
var Page    = norris.Page;
var Table   = norris.Table;
var page = new Page();

var tableDataConnector = function(done){
    var labels = [
      'January',
      'February',
      'March',
      'April',
      'June',
      'July',
      'August',
      'September',
      'October',
      'November',
      'Dicember'
    ];
    var tableData = [4,4,6,8,12,16,19,19,17,13,8,6 ];
    var table = {labels: labels, data: tableData};
    return done(null, table};
};

var table = new Table(tableDataConnector);
page.addElement(table);

app.use('/table', page);

app.listen(3000);
```

Un esempio di output per l'utilizzo precedente può essere il seguente. Puntando il *browser* all'indirizzo indicato (http://localhost:3000/table) si avrà la seguente rappresentazione:

January	February	March	April	May	June	July	August	September	October	November	December
4	4	6	8	12	16	19	19	17	13	8	6

Volendo aggiornare il grafico, in particolare una casella della tabella, lo sviluppatore dovrà usare la funzione di aggiornamento fornita dal *framework*. Con riferimento all'esempio precedente, un esempio è il seguente:

```
table.updateValue({row: 1, column: 5, value:2});
```

A seguito di tale chiamata il grafico verrà cosi aggiornato:

January	February	March	April	May	June	July	August	September	October	November	December
4	4	6	8	**2**	16	19	19	17	13	8	6

Table con aggiornamento in stream

Nel seguente esempio abbiamo invece una visualizzazione a tabella con aggiornamento in *stream*.

index.js

```
/*jshint node: true */
'use strict';

var express = require('express');
var app     = express();

var norris  = require('norris');
var Page  = norris.Page;
var Table = norris.Table;

var page = new Page();

var tableDataConnector = function(done){
    var busData = [];
    // specific logic to retrieve bus data
    // [...]
    return done(null, busData);// async version
};

var table = new Table(tableDataConnector);
page.addElement(table);

app.use('/table', page);
app.listen(3000);
```

Un esempio di output per l'utilizzo precedente può essere il seguente. Puntando il *browser* all'indirizzo indicato (http://localhost:3000/table) si avrà la seguente rappresentazione:

Timestamp	IdMezzo	Lat	Long	Capolinea	Stato porte
Oct 06 2014 12.44	839	11.837796211243	45.443008422852	Capolinea De Lazara	0
Oct 06 2014 12.44	842	11.876885414124	45.40446472168	Capolinea De Lazara	0
Oct 06 2014 12.44	833	11.884307861328	45.380382537842	Capolinea De Tagg_cimitero	1
Oct 06 2014 12.44	803	11.875713348389	45.387321472168	Capolinea De Lazara	0

Volendo aggiungere righe alla tabella, in particolare una casella della tabella, lo sviluppatore dovrà usare la funzione di aggiornamento fornita dal *framework*. Con riferimento all'esempio precedente, un esempio è il seguente:

```
table.addRow(rowData);
```

A seguito di tale chiamata il grafico verrà cosi aggiornato:

Timestamp	IdMezzo	Lat	Long	Capolinea	Stato porte
Oct 06 2014 12.46	**827**	**11.821853637695**	**45.459915161133**	**Capolinea Due Palazzi**	**1**
Oct 06 2014 12.46	**842**	**11.859348297119**	**45.420551300049**	**Capolinea De Lazara**	**0**
Oct 06 2014 12.46	**839**	**11.871828079224**	**45.413898468018**	**Capolinea De Lazara**	**0**
Oct 06 2014 12.46	**833**	**11.878223419189**	**45.410736083984**	**Capolinea De Tagg_cimitero**	**0**
Oct 06 2014 12.44	839	11.837796211243	45.443008422852	Capolinea De Lazara	0
Oct 06 2014 12.44	842	11.876885414124	45.40446472168	Capolinea De Lazara	0
Oct 06 2014 12.44	833	11.884307861328	45.380382537842	Capolinea De Tagg_cimitero	1
Oct 06 2014 12.44	803	11.875713348389	45.387321472168	Capolinea De Lazara	0

5.2 NodeJS

Node.js è un sistema *run time cross platform* per applicazioni lato *server* e applicazioni di rete. Le applicazioni scritte per Node.js sono progettate per massimizzare l'efficienza di esecuzione, usando un sistema di I/O non bloccante e eventi asincroni. Node.js e' ampiamente usato per applicazioni *real-time* grazie alla sua natura asincrona. Il sistema utilizza l'*engine* Google V8 Javascript[3] per eseguire il codice. Grazie alle elevate performance di V8, molti moduli di Node.js sono scritti in Javascript stesso.
Node.js contiene un modulo nativo asincrono per fare I/O su *file*, *sockets* e *HTTP*, grazie a questo modulo, Node.js puo' essere utilizzato come web *server* senza ricorrere a software quale Apache HTTP Server o Microsoft IIS.

5.3 Socket IO

Socket IO (http://socket.io) è una libreria Javascript per applicazioni *web real-time*. Socket IO permette di creare una comunicazione bi-direzionale tra il web *client* e il *server*. La libreria consta di due parti: una libreria lato *client* che esegue nel browser e una lato *server* per Node.js. Entrambe le componenti hanno API quasi identiche, similmente a Node.js, la libreria è *event-driven*. Esistono implentazioni sia per Android (https://github.com/nkzawa/socket.io-client.java) che per iOS (https://github.com/MegaBits/SIOSocket).
Socket IO usa principalmente il protocollo WebSocket, ma se necessario, può usare altri metodi di comunicazione bi-direzionale quali: *Adobe Flash sockets, JSONP polling* ….
Sebbene possa essere utilizzata come un semplice *wrapper* per WebSocket, essa fornisce molte altre *features* tra le quali: *broadcasting* su *socket* multipli, salvare dati associati a ciascun *client* e I/O asincrono. La libreria può' essere installata utilizzando NPM, il *package manager* di Node.js.

5.4 Fonti informative

- http://chartjs.org: fornisce grafici in HTML5 usando il tag 'canvas'. Sono presenti i grafici *Bar/Line* con metodi *update/removeData/addData.* Tali metodi sono sufficienti per costruire gli aggiornameni in *place* (per Bar) e *stream* (per Line)
- http://code.shutterstock.com/rickshaw: fornisce elementi grafici per costruire grafici interattivi in *real-time.*
- http://d3js.org: libreria per manipolare documenti basati su dati.
- http://nvd3.org: componenti grafiche per d3.
- http://maps.vasile.ch/transit-sbb: esempio di mappa dinamica. La mappa simula la movimentazione dei treni usando una fonte statica di dati e interpolando le posizioni nei vari percorsi.
- https://github.com/moudy/agenda-ui: interfaccia web per Agenda (https://github.com/rschmukler/agenda, schedulatore di *job* con persistenza su mongodb). Utile per verificare l'architettura di un pacchetto *npm* che ha come output

[3] http://en.wikipedia.org/wiki/V8_(JavaScript_engine)

un middleware di express. L'architettura delle componenti di Norris può ispirarsi fortemente ad agenda-ui/agenda.

- https://github.com/steakholders/maap, https://github.com/ApertureSoftware/MaaP : progetti realizzati nell'anno accademico 2012-2013 per l'esame di Ingegneria del Software (corso di Laurea in Informatica, Padova, Docente Tullio Vardanega) derivati dal capitolato proposto da CoffeeStrap. (http://www.math.unipd.it/~tullio/IS-1/2013/Progetto/C1.pdf).